Dullin/Strassenburg

Tips en trucs voor de MSX-computers

MSX Bibliotheek 5

DATA BECKER NEDERLANDS*

Tips en trucs voor de MSX-computers

MSX
B-5

DATA BECKER NEDERLANDS*

Tips en trucs voor de MSX-computers

Tips
en trucs
voor de
MSX-
computers

Dullin/Strassenburg

Tips en trucs voor de MSX-computers

MSX Bibliotheek 5

DATA BECKER
NEDERLANDS*

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

Oorspronkelijke titel: MSX Tips & Tricks

Basisvormgeving omslag: Studio AWB/Martin van Keulen
Vertaling: J. Meijers, Tilburg

Zetwerk: Euroset bv, Amsterdam
Druk: Drukkerij Vonk, Zeist

ISBN 90 229 3371 7
D/1986/0939/279
NUGI 857

Voorwoord

Eindelijk is hij er!

De MSX standaard voor homecomputers. Voor ons is een lang gekoesterde wens in vervulling gegaan. Er zijn eindelijk meer dan 20 computers, waarvan de soft- en hardware compatibel, d.w.z. uitwisselbaar is. Het voordeel bestaat nu hierin dat programma's, geschreven op andere MSX computers, nu zonder problemen op de eigen MSX computer gebruikt kunnen worden. Hierdoor kan binnen korte tijd een onbegrensde hoeveelheid software voor de gebruikers beschikbaar worden gesteld.

De letters MSX staan voor 'Microsoft Extended Basic'. Hieruit blijkt dat de gerenommeerde firma Microsoft de Basic voor deze computer ontwikkelde. De MSX-BASIC munt uit door een veelheid aan nuttige instrukties. Noch de instrukties voor ondersteuning van grafiek en geluid, noch de instrukties die het weergeven aangenaam verlichten zoals bijvoorbeeld >TRON< en >TROF< ontbreken.

Om het programmeren met deze computer nog aangenamer te maken, hebben we dit boek geschreven. In dit boek vindt U nuttige 'tips en trucs', die U in Uw eigen programma's zinvol kunt gebruiken. Bovendien vindt U belangrijke informatie over de interne opbouw en de machinetaal van deze computer.

Wij wensen U bij de omgang met dit boek veel succes en genoegen.

De auteurs.

Inhoud

Hoofdstuk 1: MSX standaard

Hoofdstuk 2: tekenen met de TMS9918A

Hoofdstuk 3: I/O

Hoofdstuk 4: geluid met de soundchip

Hoofdstuk 5: machinetaal

Hoofdstuk 6: systeemroutines

Hoofdstuk 7: peeks en pokes

Hoofdstuk 8: basis intern

Hoofdstuk 9: de snuffelhoek

Hoofdstuk 10: programma's

1: de MSX standaard

1 Wat is een standaard?

Er is geen sterveling meer die, bij het kopen van radio of TV, eraan denkt dat dit apparaat thuis aangesloten moet kunnen worden op het elektriciteitsnet. Het is vanzelfsprekend dat de stekker van dit apparaat in het stopcontact past. Op dat moment wordt U duidelijk dat, op z'n minst landelijk, afspraken gemaakt zijn over een standaard of norm, die geldig is tussen leveranciers.
Een standaard is, nagenoeg op elk technisch gebied, vanzelfsprekend. Alleen op het gebied van de homecomputers was dit, wegens de snelle ontwikkeling, maar meer nog uit concurrentieoverwegingen, niet het geval. Voor de gebruiker is het nagenoeg ondoenlijk, zo niet onmogelijk, een computer met periferie naar eigen keuze aan te schaffen. Meestal passen alleen die apparaten van de firma, die ook de computer vervaardigde, zonder al te veel complicaties.
Vooral bij software brengt dit feit veel ergernis met zich mee. Men hoort van excellente programma's, bijv. tekstverwerkers, die echter op de eigen computer niet lopen. Compromissen met betrekking tot het te gebruiken programma moeten dan worden gesloten.
Het is dan ook een prettige en gebruikersvriendelijke ontwikkeling dat meer dan twintig ondernemingen besloten hebben de MSX standaard te voeren.

1.2 De standaard

- Zilog Z80A 8 bits microprocessor
- Texas Instruments TMS 9918A video chip
- General Instruments AY-3-8910 Audiochip
- 32 Kilobyte ROM
- Microsoft Extended BASIC, ingeprogrammeerd in ROM
- 8 kilobyte RAM minimaal, 16 kilobyte gewenst
- 40 tekens per beeldschermregel
- 16 verschillende kleuren
- Aansluiting voor datarecorder

- Uitbreidingsruimte voor RAM geheugenmodule
- Genormeerde ruimte voor softwaremodule
- Twee joystick ingangen
- Centronics parallel aansluiting voor printer
- Per land gespecificeerd toetsenbord met 5 funktietoetsen (4 voudig uitgerust)
- Vier cursor besturingstoetsen.
- Genormeerde aansluiting van alle poorten.
- Monitoruitgang

We zullen slechts kort op de funktie van de belangrijkste bouwstenen van deze computer ingaan:

De Z80A processor

Zoals U wel zult weten bevat elke computer een microprocessor, die men de 'hersens' van de computer kan noemen. Dit IC (Integrated Circuit) noemt men de CPU (Central Processing Unit) of centrale eenheid. De CPU voert machine-instrukties uit, bestuurt het verloop in de computer en de extern aangesloten apparaten (periferie). De centrale eenheid is de belangrijkste bouwsteen in een computer.

De MSX computer bezit een Z80A processor, die ook in veel andere computers gebruikt wordt. De Z80A is een krachtige centrale eenheid, die meer dan 600 instrukties herkent, die bij de MSX computers met grote snelheid verwerkt worden. Fabrikant van deze microprocessor is de firma Zilog, uit de Amerikaanse Silicon Valley in Californië.

De video chip TMS 9918A van Texas Instruments

De video chip heeft tot taak een beeld op de monitor, resp. het TV-apparaat te geven. Bovendien is hij verantwoordelijk voor de grafiek. Ook de verschillende sprite-nivo's en het 16K grote video RAM wordt door hem beheerd. Door zijn aanwezigheid wordt de taak van de Z80A processor verlicht, die bij sommige computers deze taak gedeeltelijk moet vervullen.

De sound chip AY-3-8910 van General Instruments

Deze chip is, zoals de naam reeds aangeeft, toegevoegd voor de 'sound', de muziek. De sound chip heeft een omvang van 8 octaven. Veel huisorgels staat een wezenlijk kleinere toonomvang ter beschikking. Bovendien zijn drie regelbare geluidskanalen ingebouwd. Het is daardoor in verhouding

zeer eenvoudig om driestemmig te spelen. Vervolgens is deze chip verantwoordelijk voor de klankkleur, d.w.z. instrumenten en geluiden kunnen bedrieglijk echt worden nagebootst.

Het Microsoft Extended BASIC

Het MSX-BASIC stelt meer dan 100 instrukties ter beschikking. Hieronder bevinden zich instrukties die het invoeren en editeren van programma's comfortabel vorm geven. Ook instrukties die speciaal op de sound- of video chip betrekking hebben. Voorts instrukties voor het laden en opslaan op cassette, diskette, resp. voor het aansturen van een printer. Speciaal te vermelden zijn de mogelijkheden tot interruptbesturing.
In totaliteit bezien is dit BASIC goed doordacht, waardoor er aangenaam mee te werken valt. Uitvoerige instruktie-uitbreidingen zoals bijv. bij Commodore 64, zijn bij de MSX computers niet noodzakelijk.

8K RAM minimaal, 16K RAM gewenst.

De kleinste MSX computers hebben 8K RAM ter beschikking. In deze standaard is 16K gewenst. De trend leidt echter tot de 64K versies, d.w.z. bij deze computers is, samen met de Video RAM, 80K RAM beschikbaar.

1.3 Aansluitmogelijkheden van de MSX computer

De MSX computers kunnen natuurlijk niet zonder peripherie (randapparatuur), net zomin als alle andere computers. Daarom is het nuttig te weten welke aansluitingsmogelijkheden voor welke apparaten ter beschikking staan.

TV apparatuur

De MSX standaard maakt het aansluiten van TV apparaten via de antennebus mogelijk. De ingang bij de computer komt dan ook overeen met een normale TV-bus met een weerstand van 75 Ohm. Jammer genoeg is het beeld van een TV apparaat slechts van middelmatige kwaliteit.

Monitor

De MSX computers bezitten normaal een standaard-monitor aansluiting, die het beeld, zonder een omzetting van de frequentie, op het scherm brengt. De overeenkomende busaansluiting 1 t/m 8 is weliswaar genormaliseerd, maar jammer genoeg niet het stekkertype. Men is daardoor vaak op een adapter aangewezen, die in de vakhandel zonder moeilijkheden verkrijgbaar is.
Aanvullend is nog een zg. RGB aansluiting aanwezig. Bij deze vorm van aansluiten wordt het beeld in zijn drie grondkleuren (rood, groen, blauw) verdeeld en pas in een daarvoor geëigende monitor weer samengevoegd. Bovendien wordt een RGB status, alsmede een luminantiesignaal meegegeven. Hierdoor wordt een zeer goede beeldkwaliteit gegarandeerd.

Cassetterecorder

Voor het opslaan en laden van programma's is een 5-polige DIN bus aan de achterzijde van de computer aangebracht. Daarbij worden de van de computer komende signalen op pen 4 gelegd. De 'pen' is het tegenstuk van de stift, dus een contact binnen de bus. Bus en stekker vormen een samen te voegen eenheid. Het overdragen van gegevens van de cassetterecorder, vaak datasette of datarecorder genaamd, vindt via pen 5 plaats. Voor het starten en stoppen van de recordermotor kan een overeenkomstig signaal van pen 6, resp. pen 7 gezonden worden. Alle andere pennen vormen de massa en hebben geen bijzondere funktie.

Joystick

De MSX computers stellen 2 aansluitingen ter beschikking, die zowel voor joysticks als voor paddles gebruikt kunnen worden. Wat velen niet weten is dat deze aansluitingen ook voor andere apparaten gebruikt kunnen worden, zoals bijv. relaisbesturing. Dit vraagt echter enige elektronische kennis.

Diskdrive

Aan de pennen van de ROM cartridge worden de diskdrives d.m.v. een interface aangesloten. De meeste MSX fabrikanten bieden deze apparaten reeds aan. Naar onze mening zal de diskunit van Sony wel de boventoon gaan voeren. Het is een extreem snel en solide gebouwde 3 1/2 inch drive. De 3 ½ inch heeft betrekking op de grootte van de diskette. Deze diskettes passen makkelijk in een broekzak en zijn bovendien door een bijzondere sluiting beschermd tegen stof.
Omdat bij periferieapparaten de MSX norm niet geldig is, is het aan te

bevelen bij de aanschaf van deze apparaten eerst nauwkeurige informatie in te winnen.
Een speciale mogelijkheid, in samenhang met opslaan en laden van gegevens bij MSX computers, zal binnen niet al te lange tijd in de vorm van CD's (Compact Disk) op de markt komen. Omdat de CD's in zuivere digitaaltechniek werken, kunnen hierop onnoemlijk veel gegevens opgeslagen worden. Bovendien kan de computer zonder al te veel problemen de CD aansturen. Ter verduidelijking van de opslagcapaciteit: een CD kan ongeveer 200 microfilms bevatten. Een aantal firma's is al bezig een dergelijk apparaat te ontwikkelen.

Printer en plotter

Voor het aansluiten van printers en plotters stelt de MSX standaard een Centronics parallelaansluiting ter beschikking. Deze aansluiting is reeds geruime tijd een normaansluiting voor printer en plotter. Hierdoor bestaat er een breed assortiment in deze apparaten. De centronicsaansluiting bevat 36 pennen, waarvan de meeste aansluitingen niet gebruikt worden. De gegevens voor de printer, resp. plotter, liggen aan de pennen 2 tot 9 (8 bit). Daarnaast zijn enige aansluitingen voor besturingssignalen. De rest van de pennen is ofwel niet aangesloten, ofwel vormt de massa.

1.4 Specialisten onder de MSX computers

De MSX standaard heeft een basisconcept vastgelegd, waaraan de firma's, die tot de aanbieders van MSX computers gerekend willen worden, zich moeten houden. Niet vastgelegd is een specialisatie binnen de computerspecifieke gebieden. Het is dus mogelijk dat er computers met deze standaard zijn, die nog speciale, buiten deze standaard liggende, capaciteiten bezitten.

De fabrikant YAMAHA biedt een MSX computer aan die bijzonder 'muzikaal' is. Deze computer wordt in het kader van een CX-5 computer muzieksysteem aangeboden. De basisset bestaat daarbij uit de MSX computer, een FM module SGF-01, het mini-toetsenbord YK-10 met 44 toetsen of het normale toetsenbord met 49 toetsen.
Met deze set heeft men in principe een preset synthesizer met begeleidingsautomaat, sequenties en manuaalverdeling. Beslissend is echter de aange-

boden software als Rom-uitbreidingsmodule die in het moduleslot ingestoken kan worden. Dit slot bevindt zich aan de onderzijde van de computer en is niet MSX compatibel. Wie zich aldus diepgaand met muziek bezig wil houden, heeft hierbij een goed advies. De computer bezigt hoedanigheden, die zelfs enige grotere synthesizers niet beschikbaar hebben.

Te noemen valt hier ook de MSX computer van fabrikant SANYO. Voor deze computer is een lichtpen ontwikkeld die, met overeenkomstige software, tot verbluffende mogelijkheden leidt. Er kunnen videobeelden van een videorecorder, TV-apparaten en videocamera's op het beeldscherm gebracht worden en met behulp van de lichtpen gemanipuleerd worden. Met een beetje fantasie en handigheid kunnen daarmee fantastische resultaten bereikt worden.

Ook de SONY Hit Bit is een specialist. Hij biedt een ingebouwd adressenprogramma met notitieboekje en afsprakenkalender, die gebruikers vriendelijk via menubesturing bediend worden.
Zoals U ziet zijn er een aantal interessante ontwikkelingen binnen het bereik van de MSX computers.

2: Tekenen met de TMS9918A

2.1 Inleiding

Voordat we ons met de speciale bekwaamheden van de MSX computer op het terrein van de grafieken bezig gaan houden, zullen we eerst eens kijken hoe tekeningen door een computer in principe verwerkt worden.

Zoals U weet bestaat het beeld van een monitor, resp. TV-ontvanger, uit meerdere duizenden afzonderlijke punten. De computer moet elke punt afzonderlijk opslaan. De informatie die van een beeldpunt moet worden opgeslagen is:
- Toestand van de punt, dus geplaatst of niet geplaatst
- Kleur van de punt

Omdat het beeld voortdurend verandert, moet deze informatie in de RAM (veranderlijk schrijf/leesgeheugen) van Uw computer opgeslagen worden.

Hoe wordt nu gewoonlijk de benodigde informatie opgeslagen?

Om vast te stellen of een beeldpunt geplaatst of niet geplaatst is, is slechts een bit nodig. Een bit is de kleinste informatie-eenheid. Elektronisch komt een bit overeen met spanning op een kabel: AAN of UIT. In de computer wordt deze situatie door de beide cijfers van het binaire stelsel weergegeven. Een 0 of een 1. Voor elke beeldschermpunt wordt een 1 genomen als hij geplaatst is, een 0 als de punt niet geplaatst is.

Als een TV-beeld met 640 × 400 punten opgeslagen wordt, dan zijn daarvoor 256000 bits nodig.

Bovendien moet nog de kleurinformatie opgeslagen worden. De ruimte, benodigd voor de kleur, staat in nauw verband met het aantal ter beschikking staande kleuren. Elke beschikbare kleur wordt een bepaalde kode toegewezen. Deze kode wordt dan toegevoegd aan elke geplaatste (of niet geplaatste) punt en opgeslagen.
Omdat de geheugenruimte afhankelijk is van de kleurkode, en dus afhan-

kelijk van het aantal kleuren, is het opslaan van de kleur veel ruimteroven-der dan de puntenopslag.
Om de RAM van de computer niet voortijdig met kleurkodes te vullen, zijn beperkingen zeer zinvol. Deze beperking kan via 2 wegen bereikt worden:

1. Bij een hoog oplossend vermogen van de punten wordt het aantal weer te geven kleuren drastisch verminderd. Dikwijls staan in de hoogst mogelijke puntenoplossing slechts twee kleuren ter beschikking. Een vastgelegde puntkleur en een kleur als achtergrond. Voor nauwkeurige detailtekeningen, zoals bijv. schakelplannen, is dit hoog-oplossend vermogen zinvol. Als daarentegen meerdere kleuren gewenst zijn dan wordt de volgende mogelijkheid gebruikt.

2. Als veel verschillende kleuren weergegeven moeten worden, dan is dit, bij gelijkblijvende geheugenruimte slechts mogelijk ten koste van de oplosbaarheid van de punten. Dat betekent dat bij deze weergave telkens een aantal van de kleinst mogelijke punten (pixel) samengevoegd wordt tot een grotere punt. Deze 'grote' punt kan dan alleen maar gesloten, (dus alle pixels samen) als één punt opgeslagen worden. De daardoor bespaarde ruimte kan nu als kleuropslag voor deze 'grote' punt gebruikt worden.

Een mix uit beide bovengenoemde mogelijkheden kan worden gebruikt.

De besproken principes voor het opslaan van tekeningen worden zowel bij de tekstweergave als bij de hoogoplossende weergave van een tekening gebruikt. Op deze, voor de beeldschermmode binnen de MSX gebruikte methoden, zullen we in dit hoofdstuk nader ingaan.

Laat ons eerst eens naar de bijzondere grafische eigenschappen van de MSX computer kijken. Zoals we reeds in het eerste hoofdstuk aanhaalden wordt de beeldschermopbouw door een daartoe speciaal ontworpen chip uitgevoerd, de VDP (Video Display Processor).
De in de MSX standaard vastgelegde VDP is de door firma Texas Instruments vervaardigde videoprocessor TMS9918A. Deze processor munt uit door een aantal technische eigenschappen:

- oplossend vermogen van 256*192 beeldpunten op een normale TV ontvanger
- 16 verschillende kleuren en transparanten
- weergeven van maximaal 32 sprites
- door overlapping van 35 vlakken is 3D-simulatie mogelijk
- adressering van zelfstandige video RAM voorzien
- 4 verschillende beeldschermmodi t.w.:
- tekstmode: 40 tekens breed, 256 verschillende tekens
- grafiek 1: 32 tekens breed, sprites en kleuren
- grafiek 2: Hoog oplossende grafiek met 256*192 punten
- multicolor mode: 64*48 punten met 16 kleuren

De 4 verschillende modi zullen we uitvoerig bespreken. Bovendien is aan de sprites een eigen paragraaf toegekend. Hier gaan we in het kort nog even op punt 5 in.

Elke MSX computer bezit, naast de gebruikelijke RAM, die voor het opslaan van programma's e.d. gebruikt wordt, een video RAM, afgekort als VRAM. Deze VRAM dient uitsluitend voor de VDP om de, voor het beeldscherm noodzakelijke, gegevens op te slaan.
Het voordeel van deze methode is enerzijds dat geen byte van de kostbare RAM voor tekeningen gebruikt wordt, anderzijds dat voor het opslaan van tekeningen uitsluitend 16 Kilobyte (= 16*1024 byte = 16*1024*8 bits) ter beschikking staan. Voorts wordt het interne beheer van de VRAM door het VDP zelfstandig uitgevoerd, waardoor de processor meer tijd voor andere opdrachten heeft. Omdat de toegang voor lezen en schrijven op de beide RAM's verschillend uitgevoerd is bestaan voor de VRAM speciale BASIC-instrukties:

>VPOKE<: schrijven van waarden in de VRAM en
>VPEEK<: lezen van waarden uit de VRAM.

Met deze beide instrukties is een direkte manipulatie van de inhoud van de VRAM, en daardoor ook de aktuele beeldschermuitvoer, mogelijk.

Geeft U in:

```
SCREEN 0:For I=0 TO 255:VPOKE I,I:NEXT
```

Voordat we de diverse grafische modi afzonderlijk behandelen nog een voorbeeld van het vermogen van een MSX computer op het terrein van tekeningen.

Eerste tekenprogramma:

```
10 SCREEN 2
20 FOR I=0 TO 255 STEP 2:REM Step variëren
30 'LINE(0,96)-(I,96+95*SIN(I/33)):REM 33 variëren
31 'LINE(0,96)-(I,96+95*SIN(I/33)),I MOD 16:REM 16 variëren
32 'LINE(255-I,96)-(I,96+95*SIN(I/33)):REM 33 variëren; I MOD
15 toevoegen
33 'LINE(255-I,96-95*COS(I/33))-(I,96+95*SIN(I/33)):REM 33
bij COS of SIN wijzigen
34 'LINE(255-I,96+95*SIN(I/17))-(I,96+95*SIN(I/34)):REM 34
of 17 bij COS of SIN wijzigen
35 'LINE(255-I,191)-(I,96+95*SIN(I/33)):REM 33 wijzigen
36 'LINE(255,96-95*SIN(I/33))-(I,96+95*SIN(I/33)):REM 33 wijzigen
37 'LINE(255,96+95*SIN(I/33))-(I,96+95*SIN(I/33)),I MOD 16:
REM 33 wijzigen
38 'LINE(128-127*SIN(I/33),96)-(I,96+95*SIN(I/33))
39 'LINE(128-127*SIN(I/33),96-95*SIN(I/33))-(I,96+95*SIN(I/
33))
40 NEXT
50 WAIT &HAA,&H40:REM wacht op CAP uit
```

Dit programma bevat diverse variaties in tekeningen. Alle regels zijn door de ' als REM-regel beveiligd. Wilt U nu een van deze mogelijkheden testen, dan hoeft U slechts het REM teken in de betreffende regel te verwijderen. De ingedrukte CAP toets laat de tekening zo lang op het beeldscherm, totdat U weer op deze toets drukt. Daarna keert het programma terug naar de normale mode.

Een specialiteit van het MSX-BASIC zijn de bij grafiek en geluid voorkomende macro- of subinstrukties, die als strings weergegeven kunnen worden.

```
10 SCREEN 2
20 S=2
30 A$="c=c;s=s;nr10u10nr10e5r10nd10g5d10e5u10"
40 FOR S=1 TO 39 STEP 2
```

```
50 C=SMOD13+3
60 PSET(100-S*1.4,120+S*1.4)70 DRAW A$
80 NEXT
90 WAIT &HAA;&H40
```

2.2 De tekstmode

De tekstmode geeft mogelijkheid tot maximaal 40 tekens per regel en 24 regels per beeldscherm. 256 Verschillende tekens, waaronder 'niet-Engelse-letters' staan ter beschikking. Daardoor is het mogelijk, zonder verandering van de tekenset, op een buitenlandse tekenset over te schakelen. In de tekstmode zijn slechts twee kleuren beschikbaar: de achtergrondkleur, die tevens voor de rand dient, en de tekenkleur.

Hoe wordt nu de beeldschermtekst opgeslagen?

Omdat het over een tekstmode gaat wordt eerst elk der 256 mogelijke tekens opgeslagen. Een teken van deze soort bestaat uit 8*8 punten, waarbij in de tekstmode in horizontale richting slechts 6 punten getoond worden.

Kijk eens naar het voorbeeld

```
1  * * *
2*       *
3*       *
4*       *
5*   *   *
6*     *
7  * *   *
8
  1 2 3 4 5 6 7 8
```

Uitgaande van de matrix-weergave, wordt voor een geplaatste punt een 1, en voor de niet geplaatste punt een 0 gebruikt. De aldus verkregen reeks wordt als een binair getal geïnterpreteerd. (zie hoofdstuk 5)

```
&B01110000
&B10001000
&B10001000
&B10001000
&B10101000
&B10010000
&B01101000
&B00000000
```

Op deze manier verkrijgen we 8 bytes, een byte is een 8-bits binair getal. Hun waarde ligt tussen 0 en 255. Daarmee kan exact het onderhavige teken gedefinieerd worden. Elk van de 256 mogelijke tekens wordt op deze manier door 8 bytes weergegeven. Voor de opslag van de totale tekenset zijn dan 8*256 = 2024 bytes (2KB) nodig. Dit geheugengebied, dat bij de MSX systemen in het video RAM ligt, wordt een tekengenerator genoemd.

Het navolgende programma leest de complete tekenset van het MSX basic uit:

```
10 SCREEN 0
20 ST=BASE(2)
30 FOR I = ST TO ST+256*8-1
40 PRINT RIGHT$("00000000"+BIN$(VPEEK(I)),8)
50 NEXT
```

Regel 40 is de kern van het programma.

>VPEEK(I)< leest een byte uit de VRAM. Door >BIN$< wordt de gelezen byte in een binair getal omgezet, en door >RIGHT$< worden de eventueel ontbrekende nullen bij het begin aangevuld, zodat de totale lengte van de uit te voeren strings precies 8 tekens groot is.
>BASE(2)< in regel 20 levert bovendien het startadres van de tekenset.

Bij het runnen van het programma ziet U de gehele tekenset quasi in groot formaat over het beeldscherm lopen. Daarbij staat een 1 voor een punt en een 0 voor een open plaats.

U ziet dat de afzonderlijke tekens verticaal niet gescheiden zijn, d.w.z. dat de laatste byte van een teken onmiddellijk gevolgd wordt door de eerste

regel van het nieuwe teken. Om de afzonderlijke tekens van elkaar te onderscheiden krijgt elk van hen een getal, de zg. tekenkode. De tekenkode geeft aan welk teken in de tekengenerator bedoeld wordt.

De bytes 0 t/m 7 komen overeen met het teken 0, bytes 8 t/m 15 het teken 1 en bytes 16 t/m 23 het teken 2 enz.

In zijn algemeenheid betekent dit:

Het teken met de kode n is van het (n*8)ste byte tot aan het (n*8+7)e byte van de tekengenerator opgeslagen. Als nummering geldt de welbekende ASCII kode (American Standard Code for Information Interchange).
Deze kode legt de betekenis van de tekens met de kode 0 tot 127 vast. De MSX standaard gebruikt de ASCII-kode met kleine afwijkingen. De tekens met de kode 128 t/m 255 zijn voor MSX's eigen grafische tekens en buitenlandse letters vastgelegd. Maar laat ons het besprokene eens uitproberen. Het volgende programma voert, bij het ingeven van de betreffende kode, het daarbij behorende teken uit.

```
 5 CLS
10 SCREEN 1
20 ST=BASE(7)
30 LOCATE 3,3:INPUT"Tekencode ";ZC
40 LOCATE 0,6
50 FOR I=ST+ZC*8 TO ST+ZC*8+7
60 PRINT RIGHT$("00000000"+BIN$(VPEEK(I)),8)
70 NEXT
80 LOCATE 0,16:PRINT CHR$(ZC)
90 GOTO 30
```

Zoals U ziet is het direkt met >VPEEK< gelezen teken gelijk aan het met >CHR$(ZC)< verkregen teken. >PRINT CHR$< voert het bij die kode behorend teken uit. Het tegengestelde van de >CHR$< instruktie is de >ASC< funktie. >ASC(A$)< geeft de kode van het eerste teken van A$ weer.

Hiermee is de opbouw van de tekengenerator voldoende duidelijk behandeld. Zoals het met de >VPEEK< funktie mogelijk is de tekengenerator

uit te lezen, zo is het ook mogelijk deze met de >VPOKE< funktie te herschrijven. Dat betekent dat de tekenset eenvoudig naar eigen inzichten en wensen te veranderen is.

Wijzigen van de tekenset

Het teken, waarmee de eenvoudigste weergave is te bereiken, is de spatie. Omdat bij de spatie geen punt in de 8*8 matrix is geplaatst, wordt het door acht 0-bytes weergegeven. Dit teken willen we nu veranderen.
De ASCII kode van de spatie is 32, die ook verkregen wordt door >PRINT ASC(" ")< in te geven. Er moet nu een punt in de linker bovenhoek van de spatie geplaatst worden. De bovenste rij wordt door de eerste 8 bits voorgesteld, dus

```
SCREEN 0
VPOKE BASE (2)+32*8, &B10000000
```

In plaats van de &B10000000 kunnen we ook de decimale waarde 128 plaatsen, die met >VAL("&B10000000")< berekend wordt.
We krijgen dan direkt een gewijzigde spatie, waarbij de linker bovenpunt geplaatst is. Deze verandering vindt onmiddellijk, voor alle op het beeldscherm aanwezige spaties, plaats. De oorspronkelijke toestand kan met

```
VPOKE BASE (2)+32*8,0
```

hersteld worden.

Programma voor een eigen tekenset

```
10 REM karaktersetgenerator
20 SCREEN 0:COLOR 15,4,4:DEFINT A-Z
30 BA=BASE(PEEK(&HFCAF)*5+2):REM Basisadres
tekentabel
40 XA=5:YA=7: REM positie van letter
50 EI=0: REM invoerapparaat
60 ON STRIG GOSUB 460: REM Interruptsprong
definiëren
70 STRIG(EI) ON: REM Interrupt inschakelen
80 AU$=" ":EI$=CHR$(1)+CHR$(64+10):
90 ON KEY GOSUB 700,800: REM Interruptsprongen
definiëren
100 KEY 1,"next":KEY (1) ON
```

```
110 KEY 2,"Einde":KEY (2) ON:
120 REM
130 REM teken weergeven
140 REM
200 CLS:LOCATE 4,4:PRINT"teken:";
210 A$=INKEY$:IF A$="" THEN 210:REM invoer
220 PRINT A$: REM letter weergeven
230 BZ=BA+ASC(A$)*8:REM Basisadres teken
240 FOR I=BZ TO BZ+7
250 BY=VPEEK(I)
260 P=128
270 LOCATE XA,YA+I-BZ
280 FOR J=0 TO 7
290 IF (BY AND P)=0 THEN C$=AU$ ELSE C$=EI$
300 PRINT C$:REM tekenmatrix weergeven
310 P=P/2
320 NEXT
330 NEXT
340 X=0:Y=0
350 LOCATE XA+X,YA+Y,1
360 R=STICK(EI)
370 IF R=0 THEN 360
380 IF R=8 OR R=1 OR R=2 THEN Y=Y-1
390 IF R>3 AND R<7 THEN Y=Y+1
400 IF R>1 AND R<5 THEN X=X+1
410 IF R>5 AND R<=8 THEN X=X-1
420 IF X>7 THEN X=X-1:BEEP
430 IF X<0 THEN X=X+1:BEEP
440 IF Y>7 THEN Y=Y-1:BEEP
450 IF Y<0 THEN Y=Y+1:BEEP
460 GOTO 360
470 REM STRIG Interrupt Routine
480 BY=VPEEK(BZ+Y) XOR 2^(7-X)
490 IF (BY AND 2^(7-X))=0 THEN PRINT AU$ ELSE PRINT
EI$:REM matrix wijzigen
500 LOCATE XA+X,YA+Y
```

```
510 VPOKE BZ+Y,BY
520 RETURN
700 REM Key 1 Interrupt Routine
710 KEY (1) ON:LOCATE,,0:GOTO 200
800 REM Key 2 Interrupt Routine
810 LOCATE ,,0:REM Cursor uit
820 DEFUSR1=&H139D:REM
830 X=USR1(1)
840 CLS
850 END
```

Programmabeschrijving

Dit programma is in zoverre interessant, dat het zowel in >SCREEN 0< als in >SCREEN 1< mode funktioneert. Opmerkelijk is bovendien dat het interruptbestuurd is. Zoals U al wel zult weten is interruptbesturing in de MSX computers voorzien.

Regel 30

Hier wordt het basisadres van de tekentabel, afhankelijk van de onderhavige mode berekend.

Regel 40

De X-, resp. Y-positie van de letter wordt bepaald.

Regel 50

In EI worden de invoerapparaten vastgelegd
EI=0: Toetsenbord, d.w.z. cursortoets en spatie(vuurknop)
EI=1: Joystick 1
EI=2: Joystick 2

Regel 60

Deze regel definieert het sprongadres voor een interrupt, die door de vuurknop opgeroepen wordt. Als de vuurknop ingedrukt wordt, gaat het programma met regel 470 verder.

Regel 70

Schakelt de interrupt van de vuurknop in.

Regel 80

In AU$ (uitschakel$) wordt een spatie opgeslagen.
In EI$ (inschakel$) wordt een speciaal teken in de vorm van een reversed cirkel opgeslagen. AU$ wordt in plaats van een niet geplaatste, EI$ in plaats van een geplaatste punt gebruikt.

Regel 90

Definieert 2 sprongen na de interrupt door funktietoetsen Key1 en Key2

Regels 100, 110

De funktietoets 1 wordt met "next" vastgelegd en de interrupt wordt ingeschakeld.

De funktietoets 2 wordt met "einde" vastgelegd. Ook hier wordt de interrupt ingeschakeld.

Regel 200

Hier begint het programmadeel voor het invoeren van de tekens. Verder is deze regel vrij eenvoudig te bevatten.

Regel 210

In deze regel wordt het teken, dat via de toetsen ingegeven werd, in A$ opgeslagen.

Regel 220

Geeft het ingegeven teken op het beeldscherm weer.

Regel 230

Op deze regel wordt het basisadres van het teken (BZ) berekend. Het teken van het basisadres wordt bepaald uit het basisadres (BA) en de ASCII-kode van de in A$ opgeslagen letter *8. De 8 plaatst de teller op het begin van het betreffende teken in de tekengenerator.

Regel 240

Deze lus maakt het mogelijk byte na byte te lezen van een teken uit de tekentabel.

Regel 250

Leest een reeks van 8 punten van het teken in BY (bytematrix)

Regel 260

Hier begint de routine die het teken vergroot op het beeldscherm weergeeft.

In P (Potentie) wordt de waarde 128 voor later gebruik opgeslagen.

Regel 270

Hier wordt de aktuele positie voor de uitvoer op het beeldscherm berekend.

Regel 280

Lus voor het bit na bit lezen van een byte.

Regel 290

Als het bit geplaatst is, wordt het bijzondere teken in EI$ uitgevoerd, anders een spatie (AU$).

Regel 300

Voert het onderhavige teken uit.

Regel 310

P, waarvan de waarde altijd met het aktuele bit overeenkomt, wordt gehalveerd. Zo ontstaat een getallenreeks die er als volgt uitziet: 128, 64, 32, 16, 8, 4, 2, 1

Deze reeks komt overeen met de waarde van de bits, van links naar rechts gelezen.

Regel 350

Deze regel plaatst de cursor op de aktuele positie binnen het grote, op het beeldscherm weergegeven, teken en schakelt die in.

Regel 360 tot 460

Hier wordt de joystickbeweging en de vraag gerealiseerd. Door deze routine kan men, binnen de "grote tekens", met de joystick vrij bewegen. Hier begint het eigenlijke hoofdprogramma.

Regel 470, 520

Naar deze regel wordt, bij het veroorzaken van een interrupt door vuurknop of spatie, gesprongen. In BY wordt nu in de 'grote matrix' door knopdruk een teken geplaatst of gewist, afhankelijk van de voorafgaande situatie.

Regel 500

Plaatst de cursor weer op de hierboven veranderde positie, binnen het 'grote teken'.

Regel 510

Schrijft het veranderde bit (=1 punt) in de tekentabel.

Regel 700, 710

Ingang voor de door KEY1 opgeroepen interrupt. Bewerkt de keus en de beeldschermuitvoer van het volgende teken.

Regels 800 tot 850

Routine die, door het indrukken van KEY2, via een interrupt wordt bereikt. Schakelt de cursor uit en herstelt de originele toetsenbezetting.

In de volgende regels wordt het programma beëindigd.

Regel 830

Roept de routine voor de standaard toetsenreservering op.

Na het starten van het programma legt U het te veranderen teken, door het indrukken van die toets, vast. Daarna wordt het teken in groot formaat weergegeven en de cursor verschijnt op het beeldscherm. U kunt nu met behulp van de cursortoetsen, resp. joystick, de cursor binnen deze 8*8 matrix bewegen. Door het indrukken van de spatietoets, resp. vuurknop van de joystick, wordt de op dat tijdstip onder de cursor liggende punt

geïnverteerd, d.w.z. geplaatst als hij nog niet geplaatst was, of omgekeerd. De verandering gaat onmiddellijk in, zowel in de weergave als in het originele teken. Met de funktietoets "F1" kunt U volgende tekens wijzigen, met "F2" het programma beëindigen. Let er op dat, bij het veranderen van letters, in de tekstmode slechts tekens van 6 punten bestaan. Dit in tegenstelling tot de grafische mode 1.
Hiermee is de tekenset uitvoerig besproken.

Besturingstekens

De hierna volgende bijzonderheden dienen nog enigszins toegelicht te worden. De tekens met de kode van 0 t/m 31 kunnen niet direkt door een simpele >CHR$< opgeroepen worden. Dit is noodzakelijk omdat alle kodes, kleiner dan 32 normaal als besturingstekens worden geinterpreteerd. Ook deze besturingstekens zijn door de ASCII-kode genormaliseerd. Alle besturingstekens kunnen via de CTRL-toets, in kombinatie met bepaalde andere toetsen, opgeroepen worden. Bovendien zijn voor belangrijke stuurtekens aparte toetsen aangewezen. Met het volgende programma kunt U de kode's van de CTRL+toetskombinatie ontdekken:

```
10 A$=INKEY$:IF A$="" THEN 10
20 PRINT ASC(A$):GOTO 10
```

De navolgende toewijzing geldt:

Kode	CTRL+ toets	Funktie
0	@	–
1	A	Uitvoer grafische tekens
2	B	Sprong per woord terug
3	C	Einde invoer (AUTO uitschakelen)
4	D	–
5	E	Wissen tot einde regel
6	F	Sprong per woord vooruit
7	G	Pieptoon
8	H	Backspace (BS toets)
9	I	Tabulator (TAB toets)
10	J	Regel opvoer (Line feed)
11	K	Cursor home (HOME toets)
12	L	Beeldscherm wissen (SHIFT/HOME toets)
13	M	Carriage return (RETURN toets)
14	N	Sprong aan einde regel
15	O	–
16	P	–
17	Q	–
18	R	Insert mode in/uit
19	S	–
20	T	–
21	U	Regel compleet verwijderen
22	V	–
23	W	–
24	X	Select-toets
25	Y	–
26	Z	–
27	[	Escape toets (haakje open)
28	U+(SHIFT)	Cursor links
29	]	Cursor rechts
30	^ (golf)	Cursor omhoog
31	onderlijnen	Cursor omlaag
127		DELETE toets

De besturingstekens kunnen met CTRL en gedeeltelijk met speciale toetsen, maar ook door >PRINT CHR$(..)< opgeroepen worden, bijv.

>PRINT CHR$(7)<. Voor kodes, kleiner dan 32, worden door >CHR$< geen tekens op het beeldscherm gebracht. Omdat deze tekens wel aanwezig zijn is er een andere methode om die op het beeldscherm te krijgen.
Hier een voorbeeld:

Het teken met kode n (n is kleiner dan 32!) wordt door
PRINT CHR$(1);CHR$(64+N)
getoond.

Hier is >CHR$(1)< een besturingsteken dat de computer meedeelt dat de volgende kode kleiner is dan 32. Interessant in deze samenhang is dat het teken met kode 0 in de tekenset niet gevuld is, dus gelijk aan een spatie.

Behalve kode 0 is er ook nog een tweede bijzondere en wel:

De kode 255

```
10 SCREEN 0: WIDTH 36
20 LOCATE 0,5
30 FOR I=5 TO 20
40 PRINT STRING$(36,255)
50 NEXT
60 LOCATE 0,0:PRINT"Het cursorteken heeft code 255"
70 FOR I=0 TO 34:LOCATE I,0,1
80 FOR W=0 TO 150: NEXT W
90 NEXT
```

De verklaring van hetgeen U zoëven zag is als volgt:
De cursor wordt door het teken met kode 255 weergegeven. Het operationele systeem verandert voortdurend naargelang het teken dat zich op de cursorpositie bevindt, dus de definitie van het 255e teken. Zij komt overeen met de inverse weergave van het teken. Door het inverteren worden geplaatste punten gewist en omgekeerd. Het weergeven van geïnverteerde tekens zullen we in het hoofdstuk over de grafische mode 1 behandelen.

Wat is een naamtabel?
Om het complete beeld te ontwikkelen is het opslaan van de tekenset niet

voldoende. Er moet nog vastgelegd worden welk teken op welke plaats van het beeldscherm moet verschijnen.
Deze opgave bestaat uit het aanleggen van een naam- of modelnaamtabel. Alle mogelijke tekenposities op het beeldscherm worden doorgenummerd. Men begint in de linkerbovenhoek en teken voor teken wordt doorgenummerd, totdat de rechter benedenhoek wordt bereikt. Door deze handelwijze is elke beeldschermpositie ondubbelzinnig bepaald.
Omdat het beeld in model 1 uit 40 posities en 24 regels opgebouwd is, moet de naamtabel 40*24= 960 bytes lang zijn. Byte 0 van de naamtabel bevat de kode van het teken dat linksboven in de hoek staat, byte 1 de kode van het volgende teken enz.
De kode van het eerste teken in de 2e rij is dan de 40e byte van de tabel toegewezen. Bij >SCREEN 0< krijgen we het startadres van de naamtabel met >BASE(0)<. In standaardgevallen is dit adres 0

Daardoor kan de instruktievolgorde

```
LOCATE X,Y:PRINT CHR$(64);
```

vervangen worden door:

```
VPOKE X+40*Y,64
```

Dit is alleen geldig bij >WIDTH 40<

Met behulp van de >VPOKE< instruktie kan beeldschermuitvoer vaak met grotere snelheden uitgevoerd worden. Intern worden diverse >PRINT< instrukties, uiteindelijk omgevormd tot >VPOKE< instrukties (die op hun beurt weer uit I/0 instrukties bestaan), uitgevoerd. (zie hoofdstuk 3)

Nogmaals samengevat:

SCREEN 0

Startadres van de naamtabel	BASE(0)=0
Lengte van de tabel	40*24=960 bytes
Startadres tekengenerator	BASE(2)=&H0800
Lengte van de tabel	256*8=2048 bytes

Naamtabel	&H0000–&H03BF nauwelijks 1K
Vrij	&H03C0–&H07FF meer dan 1k
Tekengenerator	&H0800–&H0FFF 2K
Vrij	&H1000–&H3FFF 12K!

Zoals U ziet is minder dan een kwart van de 16K VRAM gebruikt. Om deze braakliggende capaciteit te kunnen gebruiken gaan we ons nu bezighouden met de registers van de VDP.

Registers van de VDP

De TMS9918A bezit 8 registers, 7 schrijfregisters (niet leesbaar) en 1 leesregister (niet beschrijfbaar).
Een register moeten we ons voorstellen als een geheugenplaats voor een, resp. meerdere bytes. Een geheugenplaats die een speciale funktie bezit, noemt men vaak een register. Een register 'registreert' voor een bepaald element belangrijke informatie.

De VDP heeft tot taak een beeldscherm te vervaardigen. Daarvoor heeft hij bijv. startadressen van de tekengenerator en de naamtabel nodig. Bevinden we ons in de tekstmode dan zijn deze gegevens, alsmede de informatie over de aktuele beeldschermkleuren, voldoende om een scherm aan te maken.
Om een korrekt beeld door de VDP te laten maken moet o.a. deze informatie doorgegeven worden. Diverse registers van de VDP zijn 8-bits registers, waarbij vaak niet alle 8 bits gebruikt worden.

De naamtabel wordt in register 2 opgeslagen. Daartoe wordt ze eerst gekodeerd. De echte opslagplaats van de naamtabel in de tekstmode verkrijgen we door >BASE(0)<. De standaard hiervoor is de 0. (Indien U nog niet op de hoogte bent van hexadecimale en binaire getallenstelsels lees dan eerst in hoofdstuk 5 de paragraaf over getalstelsels)

Principieel zijn adressen van de video RAM opgeslagen onder de geheugenplaatsen 0 tot &H3FFF. Om &H3FFF, dus het hoogste VRAM adres, op te slaan zijn 14 bits (=2+3*4, waarom?) nodig. Van deze 14 bits worden slechts de bovenste, dus die met de hoogste plaatswaarde, in de registers opgeslagen.

In het geval van de naamtabel worden slechts de bovenste 4 opgeslagen, dus de bits 10 t/m 13. Het bit nr. 10 heeft de waarde 2^10= 1024. Dat betekent dat het adres van de naamtabel in stappen van 1 KB verplaatst kan worden.

In het hierna volgende leert U een mogelijkheid om de vrije ruimte in de VRAM, door een verandering van de VDP registers, te gebruiken.
Daardoor kunt U 14 verschillende beeldschermen gelijktijdig opslaan en naar keuze hierop overschakelen.
Omdat het oorspronkelijke adres van de naamtabel 0 is, geven >BASE(0)< en ook >VDP(2)< de waarde 0. De tekengenerator bezet de adressen tot &H0FFF. Het daarboven liggende gebied is vrij. We zullen nu de naamtabel verplaatsen naar adres &H2000.
U dient daartoe in te geven

```
BASE(0)=&H2000
```

of, indien U direkt het VDP register 2 wilt wijzigen:

```
VDP(2)= &H2000/2^9
```

De deling door 2^9 houdt nu alleen maar rekening met de bovenste 4 bits (10 t/m 13).
Jammer genoeg bereiken we door deze instrukties wel een interessant, maar niet het gewenste effect. Op het nu uitgevoerde beeld dat er, onder bepaalde voorwaarden, zelfs chaotisch uitziet, kunnen we noch de cursor bewegen, noch enige invoer verzorgen. Schakel daarom weer in de uitgangspositie:

```
VDP(2)=0    resp.    BASE(0)=0
```

Laat U zich niet beïnvloeden door het feit dat U niet kunt zien wat U ingeeft. De invoer is op het oorspronkelijk beeldscherm terechtgekomen en niet op het scherm dat toevallig getoond wordt.

Hiermee wordt aangetoond dat de VDP onafhankelijk van het BASIC werkt. Als zijn registers veranderd worden, dan geeft hij het daarmee overeenstemmende beeld. In ons geval 'wist' het systeem niet dat we de naamtabel verlegd hadden. Als gevolg hiervan kwamen diverse invoeren in de oude naamtabel terecht, die niet meer door de VDP gebruikt werd. Het is gebruikelijk dat de >SCREEN< instruktie de noodzakelijke informaties aan het operationele systeem doorgeeft. Daarbij wordt dan het, met >BASE< ingegeven, adres gebruikt.

Geeft u dus in:

```
BASE(0)=&H2000
SCREEN 0
```

De naamtabel bevindt zich nu op het aangegeven adres, wat U met >PRINT HEX&(BASE(0))< kunt uittesten. Deze methode heeft als nadeel dat de gegeven beeldscherminstruktie door de >SCREEN< instruktie werd gewist. Moeten de diverse beeldschermen naast elkaar gebruikt worden, dan moet het mogelijk zijn over te schakelen zonder dat de inhoud van het beeldscherm verloren gaat.

Het adres waar het operationele systeem het adres van de naamtabel opslaat is &HF922.
Plaatsen we nu dit adres op de nieuwe waarde, dan krijgt U ook een cursor op het nieuwe beeld.

Het hierna volgende programma moet U aantonen dat U meerdere beeldschermen in de direkt mode, dus bijv. bij het programmeren, kunt gebruiken. U kunt dan op de diverse beeldschermen verschillende delen van Uw programma onderbrengen, en naar behoefte heen en weer schakelen.

```
9999 REM vooraf KEY1, "RUN 10000"+CHR$(13) invoeren
10000 REM pagina-omschakelaar
10010 A$=INKEY$:IF A$="" THEN 10010
10020 A=VAL(A$)
10030 IF A=0 AND A$<>"0" THEN A=(ASC(A$) AND
&B11011111)-55
10040 IF A>13 OR A<0 THEN 10010
10050 POKE &HF923,(A+2+2*(A<2))*4
10060 VDP(2)=A+2+2*(A<2)
10070 LOCATE 0,20
```

Na het invoeren van het programma en:

```
KEY1, "RUN 10000"+CHR$(13)
```

kunt U door het indrukken van F1 het gewenste beeldscherm door het indrukken van de toetsen 0 tot 9 en A,B,C of D kiezen, waarbij 0 het standaard beeldscherm is. Bij het eerste gebruik van een beeldscherm moet U dit meestal nog met SHIFT+HOME wissen.

Programmaverklaring:

Regel 10010 haalt de ingedrukte toets op
Regel 10020 onderzoek waarde indien getal
Regel 10030 onderzoek waarde indien letter
Regel 10040 onderzoek op geldigheid
Regel 10050 schrijft het adres van naamtabel voor het O.S.
&HF922 bevat low byte (=0), en
&HF923 bevat high byte van adres.

Toewijzing van A aan het adres

A	Na tab adr	A	Na tab adr
1	&H0400	7	&H2400
2	&H1000	8	&H2800
3	&H1400	9	&H2C00
4	&H1800	10 (A)	&H3000
5	&H1C00	11 (B)	&H3400
6	&H2000	12 (C)	&H3800
		13 (D)	&H3C00

Regel 10060 Video register 2 met de gegeven waarden laden.
Daarbij veroorzaakt +2*(A>2) een –2 als A<2 is, waardoor de toewijzing een feit is.
Regel 10070 Plaatst de cursor aan het einde van het beeldscherm.
Vanzelfsprekend kunt U deze routine ook in Uw programma's opnemen. Zou de regelnummering gelijk zijn aan de in het geheugen opgenomen BASIC regels vanaf 10000, dan kan dit programma ook anders genummerd worden.

Let U bij het veranderen van basisadressen vooral op het verschil van de >VDP< naar de >BASE< instruktie.
De VDP instruktie beschrijft uitsluitend het onderhavige register met de aangegeven waarde. De verandering wordt door de >SCREEN< instruktie teruggeplaatst in de uitgangssituatie. Daarentegen worden de door >BASE< veroorzaakte wijzigingen niet alleen in het register van de video chip aangebracht maar ook in de RAM opgeslagen en zo bij de nieuwe

>SCREEN< instruktie als uitgangssituatie aangenomen. Een verandering via de >BASE< instruktie is zolang geldig totdat zij met een andere >BASE< instruktie herroepen wordt. De wijzigingen met >VDP< worden door >SCREEN<, maar ook door >BASE< overschreven.

Zoals U weet kan door >VDP< een register niet alleen beschreven, maar ook gelezen worden. Bij de TMS9918A is dit echter niet mogelijk, met uitzondering van register 8. Daarom worden de aktuele registerwaarden altijd in RAM mee opgeslagen. Vanaf adres &HF3DF staan de registerwaarden van de video chip, te beginnen met 0, d.w.z. voor:

```
PRINT VDP (N)
```

kan ook geschreven worden

```
PRINT PEEK(&HF3DF+N)
```

In de tekstmode zijn nog meerdere VDP registers van belang. In register 4 worden de 3 MSB bits, (Most Significant Bits) van het startadres van de tekensgenerator opgeslagen. Er bestaat dus een mogelijkheid om tussen de diverse tekensets en zelfs tussen tekenset en beeldscherm te switchen. Dit zal nog nader in het hoofdstuk over de grafiekmode I worden besproken.

Kleuren in de tekstmode

We zullen nu wat nader op de kleuren ingaan. De kleuren worden in register 7 opgeslagen. Omdat er 16 verschillende kleuren zijn kunnen slechts 2 kleuren in 1 byte opgeslagen worden. Daarbij bepalen de 4 MSB's (bit 4 t/m 7) van het register 7 de achtergrondkleur, en de 4 LSB's (Lower Significant Bytes (de bytes 0 t/m 3) de schrijfkleur.

In de tekstmode is de vensterkleur altijd gelijk aan de achtergrondkleur. In de tekstmode zal het U dus niet lukken meerdere kleuren op het beeldscherm te brengen. Ook een reverse weergave van letters of cijfers is niet mogelijk. Omdat het zeer nuttig is voor een overzichtelijke beeldvorming, bijv. een menu, zullen we U een programma geven dat de reverse weergave implementeert.

Daarvoor zullen we eerst de tekenset veranderen. De tekens met de kode's 0 tot 127 blijven onveranderd. De kode's van 128 tot 255 vormen de onderste 128 tekens, alleen worden ze geïnverteerd. Als een teken geïnverteerd moet worden weergegeven, dan wordt eenvoudig 128 aan zijn kode toegevoegd. Maar eerst het programma dat de tekenset op de besproken manier manipuleert.

```
10 BA=BASE(2)+128*8
20 FOR I=BA TO BA+128*8-1
30 VPOKE I,VPEEK(I-128*8) XOR 255
40 NEXT
```

Nu kunnen geïnverteerde teksten door >VPOKE< uitgevoerd worden. De normale uitvoer met

LOCATE X,Y: PRINT CHR$(Z)

moet vervangen worden door

VPOKE BASE(0)+Y+X*40,Z+128

om het geïnverteerde teken te verkrijgen. Omdat deze handelwijze principieel te omvangrijk zou zijn om gehele woorden en zinnen te schrijven hebben we de volgende machinetaalroutine geschreven. Met behulp van deze routine wordt een uitvoer met de PRINT-instruktie mogelijk, alleen is nu alles invers weergegeven.

```
10 REM BASIC Lader Patch Invers
20 CLEAR 200, &HF370
30 FOR I=&HF370 TO &HF37A
40 READ A$
50 POKE I,VAL("&H"+A$)
60 NEXT
70 DATA E1,F1,FE,20,38,02,F6,80,F5,E5,C9
80 POKE &HFDA5,&HF70
90 POKE &HFDA6,&HF3
100 REM aktiveren met POKE &HFDA4,&HC3
110 REM uitschakelen met POKE &HFDA4,&HC9
120 REM niet in directmode!
```

Dit programma laadt een systeemuitbreiding, die in het patch bereik van het MSX systeem geschreven wordt. Door >POKE&HFEE4,&HE1< wordt de geïnverteerde tekenuitvoer ingeschakeld en door >POKE &HFEE4,&HC9< uitgeschakeld. De tekenset moet voor het gebruik van dit programma dienovereenkomstig behandeld worden, d.w.z. de kode's 128 tot 255 moeten de inverse matrixen van de tekens 0 tot 127 bevatten.

Voor de 'freaks' hier de assemblerlisting van de patch:

```
FDA4  C370F3    JP    &HF370 ; patch uitvoer op beeldscherm
F370  E1        POP   HL ; terugkeeradres
F371  F1        POP   AF ; tekenkode
F372  FE20      CP    &H20 ; besturingsteken?
F374  3802      JR    CN &HF378 ; ja, dan niet wijzigen.
F376  F680      OR    &H80 ; kode = kode + 128
F378  F5        PUSH  AF ; tekenkode staat in stack
F379  F5        PUSH  AL ; terugspringadres in stack
F379  CA        RET       ; verder in het ROM programma.
```

Hier een voorbeeld:

```
10 POKE &HFDA4,&HC3:PRINT"Invers"
20 PRINT "nog altijd"
30 POKE &HFDA4,&HC9:PRINT"nu normaal"
40 PRINT "nog normaal, tot POKE &HFEE4,&HC3"
```

Omdat we ons juist met een veeleisende beeldschermvormgeving bezighouden zullen we nog even bij dit onderwerp blijven staan.

'Windows' met MSX

Ofschoon MSX geen windows heeft voorzien, kunnen deze toch op een acceptabele manier aangemaakt worden. Een window (raam of kader) is een vast, gedefinieerd gedeelte in het totale beeldscherm. In dit window kan, onafhankelijk van het overige beeldscherm, uitvoer weergegeven worden. Het kan elke gewenste rechthoekige vorm aannemen.

Hoe kan een soort window voor de MSX gedefinieerd worden?

Als eerste zou hiervoor de instruktie >WIDTH< gebruikt kunnen worden. Zij legt de breedte van het uitvoergebied vast. Deze instruktie voert echter automatisch een CLS (Clear Screen) uit, waardoor de inhoud van andere kaders wordt gewist.
Het automatisch wissen van het beeldscherm bij de instruktie >WIDTH< kan met een bepaalde >POKE< instruktie worden voorkomen. De plaats waar de aktuele beeldschermbreedte voor het syteem opgeslagen ligt is adres &HF3B0.

Door een verandering van deze geheugenplaats met >POKE< kan de beeldschermregelbreedte ingesteld worden, zonder dat een CLS volgt. Hiermee is de eerste stap in de richting van de windowtechniek gezet.

Om het vastgelegde beeldschermbereik verder te begrenzen houden we ons nu bezig met het aangeven van de invoerruimte op het beeldscherm. De invoerruimte kan vanuit het BASIC in- en uitgeschakeld worden.
De regel, waarin de opgave volgt, is opgeslagen in adres &HF3B1. De hier opgeslagen waarde komt overeen met de laatste, door de beeldschermuitvoer gebruikte, regel als KEY off (uit) is. Is KEY on dan is de waarde het nummer van de laatst gebruikte regel. Probeert U maar eens:

```
POKE &HF3B1,15
```

Ogenschijnlijk verandert er niets. Als u echter met de cursor in de richting van de onderste beeldschermrand beweegt, zult U vaststellen dat die op de 15e regel (indien KEY off, anders op de 14e) blijft staan. Hiermee is het onderste deel van het beeldscherm beschermd. Ook bij het scrollen beweegt alleen het bovenste beeldschermdeel. Op deze manier legt U de onderste regel in een window vast.
Maar dat is nog niet alles. Er is nog een andere geheugenplaats die het beeldschermformaat belangrijk beïnvloedt. Deze geheugenplaats is &HF3DE. Voorlopig wordt deze byte als een 'flag' gebruikt, om te bepalen of de key in- of uitgeschakeld is.

```
KEY OFF: PRINT PEEK(&HF3DE)
```

geeft als waarde 0 en

```
KEY ON: PRINT PEEK (&HF3DE)
```

geeft als waarde 255.
Wordt deze plaats direkt door >POKE< op 0 resp. 255 gezet, dan treedt deze situatie pas na invoer van CLS in.
Met de >POKE< is men ook in staat andere waarden als 0 en 255 in te geven, probeert U maar eens

```
CLS: POKE&HF3B1,20: KEY ON      en daarna
POKE &HF3DE,250
```

Als U nu met de cursor naar beneden gaat, komt U niet verder dan de vijfde regel boven de invoer. Geeft U daarentegen in

```
POKE &HF3DE,2
```

dan kunt U tot 2 regels onder de regel van de uitvoer komen.

Samengevat:

&HF3B1-Regel van de KEYuitvoer

&HF3DE-KEY on/off flag in samenwerking met de regels van de KEYuitvoer.

De laatste beeldschermregel verkrijgen we met

```
PRINT
PEEK(&HF3B1)+PEEK(&HF3DE)+256*(PEEK(&HFDE)>127)
```

Door handig met deze geheugenplaatsen te manipuleren is een beeldschermvorm met windows mogelijk.

2.3 De mode 'grafiek 1'

In deze mode worden, in tegenstelling tot de tekstmode, alle punten van de 8*8 matrix van de tekengenerator aangegeven. Hierdoor wordt het maximaal aantal tekens per regel beperkt tot 32. Het aantal regels blijft 24. Deze mode laat het gebruik van alle 16 kleuren gelijktijdig toe. De achtergronden de kaderkleur kunnen verschillend zijn. Een ander belangrijk aspect van deze mode is dat het gebruik van sprites voorzien is.

Het basisadres van de tekengenerator verkrijgen we door grafiek 1 met >BASE(7)<; het adres van de naamtabel met >BASE(5)<. Deze tabellen zijn hetzelfde opgebouwd als die van de tekstmode. De naamtabel dient natuurlijk wel aan het nieuwe beeldformaat aangepast te worden. Hun lengte is dus 32*24=768= &H300 bytes

De VDP registers 2 en 4 worden hetzelfde als in de tekstmode gebruikt.

We gaan nu eerst in op de behandeling van de kleuren.
Met de instruktie >COLOR< kunnen schrift-, achtergrond- en kaderkleur onafhankelijk van elkaar bepaald worden. Het VDP register 7 slaat de kaderkleur in de 4 lower bytes op. De achtergrondkleur en de schriftkleur worden in de grafiekmode 1 door een kleurtabel bepaald. De bovenste 4 bits van register 7 worden dus niet gebruikt.
Het startadres van de kleurentabel is met >BASE(6)< te vinden. De bovenste 8 bits van het kleurentabeladres zijn in het VDP register 3 opgeslagen. Daardoor is de kleurentabel in stappen van &H40=2^7=2^(14-7) te verplaatsen.

Hoe is nu de kleurentabel opgebouwd?

Bij het inschakelen van de computer kunt U hoogstens een schrift- en achtergrondkleur verkrijgen. Het is echter ook mogelijk alle 16 kleuren gelijktijdig te benutten. Daartoe bevat elke byte van de kleurtabel informatie over schrift- en achtergrondkleur. Zoals in de tekstmode bij het inbrengen in VDP register 7, zijn de onderste 4 bits voor de achtergrond en de bovenste 4 voor de schriftkleur verantwoordelijk. Alle mogelijke tekens van de tekengenerator zijn in groepen van 8 opeenvolgende tekens ingedeeld. Een groep van acht kan, door het inbrengen van een overeenkomstige waarde in de kleurtabel een eigen, onafhankelijke kleur krijgen. De kleur van het teken wordt met de kodes 0 t/m 7, dus de eerste byte, van de kleurentabel bepaald. De kode 8 t/m 15 door de tweede byte enz. Algemeen geldt:
De kleur van het teken met de kode n, wordt door de 1+INT(n/8)e byte van de kleurtabel bepaald. De kleurtabel is derhalve 32 bytes lang.

Wordt de kleur door BASIC met de >COLOR< instruktie vastgelegd, dan worden alle 32 bytes van de kleurtabel met dezelfde waarde beschreven, bijv. >COLOR 15,4,4<. Met >VPOKE< bestaat de mogelijkheid afzonderlijke 8groepen verschillend te kleuren. Als voorbeeld bekijken we de uitvoer van alle hoofdletters met reverse weergave.
De hoofdletters bezetten de kodes van &H41 (=A) tot &H5A (=Z). De bytes van INT (&H41/8)+1 tot INT (&H5A/8)+1, dus bytes 9 t/m 12, moeten met de reverse kleuren beschreven worden.

```
FOR I=BASE(6)+8 TO BASE(6)+11: VPOKE I,15+4*16:NEXT
```

Natuurlijk kan ook hier weer tussen meerdere kleurtabellen heen en weer geschakeld worden.

De indeling van de VRAM, na het inschakelen is:

Tekengenerator	:H0000 - &H07FF
vrij	:H0800 - &H17FF
Naamtabel	:H1800 - &H1AFF
Sprite-attribuuttabel	:H1B00 - &H1B7F
vrij	:H1B80 - &H1FFF
Kleurtabel	:H2000 - &H201F
vrij	:H2020 - &H37FF
Sprite-model	:H3800 - &H3FFF

(De sprite tabellen worden in hoofdstuk 2.6 behandeld)

Een omschakeling met >BASE< instruktie in de grafiekmode is jammer genoeg niet zonder meer mogelijk:
De routine die de >BASE< instruktie uitvoert heeft bij de Sony computer een kleine afwijking. Schrijft U voor de test een tweede kleurentabel vanaf &H2040:

```
FOR I =&H105F:VPOKE I, &H4F:NEXT
```

Schakel nu met de >VDP< instruktie naar de nieuwe kleurentabel
&H2040/&H40=129 dus
VDP(3)=129
en U krijgt nu nieuwe kleuren. Met
VDP (3)=128
kan weer teruggeschakeld worden. Hetzelfde resultaat dient U met
BASE(6)=&H2040
te bereiken. Maar dat is niet het geval!! De volgende truc moet hulp bieden:
DEF USR 9= &H7B
Na deze invoer kunnen alle >BASE< instrukties, met betrekking tot de grafiek mode, door het bijstellen met >X=USR9(1)< gelijk gericht worden. Het korrekte resultaat bereiken we door

```
BASE(6)=&H2040:X=USR9(1)
```

Hetzelfde geldt ook voor de instrukties >BASE(5)=...< tot >BASE(9)=...<.
De uitvoering van de >BASE< funktie, dus in de vorm >...=BASE(...)< is voor alle waarden korrekt en mag niet achter >X=USR9(1) geplaatst worden.

Het overschakelen met >BASE< of >VDP< kan nu gebruikt worden om verschillende beelden, evenals bij de tekstmode, naast elkaar op te slaan en naar keuze uit te voeren.

Er moeten twee verschillende beelden, met verschillende kleurtabellen, en als bijzonderheid, ook verschillende tekengeneratoren aangemaakt worden. We kunnen dan naar keuze verschillende schrifttypes gebruiken. Ontwerp daarvoor een geheugenreserveringsplan, waarin deze informatie ondergebracht kan worden. Let daarbij op dat

- de naamtabel in &H400 stappen
- de tekengenerator in &H800 stappen
- de kleurtabellen in &H040 stappen
- de spriteattr.tabel in &H080 stappen
- de spritemodeltabel in &H800 stappen

verplaatst moeten worden.

Voorstel:

&H0000 - &H0800	tekengenerator 1
&H0800 - &H1000	tekengenerator 2
&H1000 - &H1300	naamtabel 1
&H1300 - &H1320	kleurtabel 1
&H1320 - &H1400	
&H1400 - &H1700	naamtabel 2
&H1700 - &H1720	kleurtabel 2
&H1720 - &H1B00	vrij
&HB00 - &H1B80	sprite attribuuttabel
&H1B80 - &H3800	vrij
&H3800 - &H4000	sprite model

```
10000 A$=INKEY$: IF A&="" THEN 10000
10010 A=VAL(A$): IF A>1 THEN 10000
10020 DEFUSR 9=&H7B
10030 BASE(7)=&H800*A: X=USR9(1): REM tekengenerator
10040 POKE &HF925,&H8*A: REM High Byte
10050 BASE(5)=&H1000+&H400*A: X=USR9(1): REM naamtabel
10060 POKE &HF923,&H10+&H4*A: REM High Byte
10070 BASE(6)=&H1300+&H400*A: X=USR9(1): REM kleurtabel
```

Dit programma funktioneert analoog aan het programma uit het voorgaande hoofdstuk. De BASIC adressen van de tekengenerator zijn:

&HF924(low) en &HF925(high).

Naar dit adres moet het startadres van de tekengenerator geschreven worden.
Natuurlijk moet U vooraf de tweede tekenset definiëren.

Probeert u het volgende kleine programma:

```
1 VDP(2)=VDP(2) XOR 1:GOTO1
```

XOR 1 zorgt ervoor dat het bit 0 van VDP register 2 geinverteerd wordt. Hierdoor wordt voortdurend tussen de aktuele naamtabel (afstand &H400) en de daarboven liggende naamtabel overgeschakeld. Dit omschakelen funktionneert echter niet snel genoeg om een overlapping van de beide beelden te bewerkstelligen.

Om dit te bereiken moeten we in de interne interruptroutine zien te komen. Deze routine wordt 50 keer per seconde opgeroepen. Zij maakt o.a. de programmering van interrupts in BASIC mogelijk.
Om onze eigen routine te kunnen inbouwen, gebruiken we de tweede patch van de interruptroutine. Dit adres is &HFD9F.

```
FD9F C360F3  JP    F360       ; patch interrupt
F360 F5      PUSH  AF         ; accu redden
F361 21F3E1  LD    HL,&HF3E1  ; adres VDP register 2 inhoud
F364 7E      LD    A,(HL)     ; waarde VDP register 2 lezen
F365 EE01    XOR   01         ; bit inverteren (omschakelen)
F367 47      LD    B,A        ; nieuwe waarde
F368 DE02    LD    C,2        ; moet in register 2 geladen worden
F36A CD6047  CALL  &H9947     ; leidt tot VDP(C)=B
F36D F1      POP   AF         ; accu ophalen
F36E C9      RET              ; verder in ROM
```

De BASIC lader van dit programma is:

```
10 REM Interrupt omschakelaar
20 CLEAR 200,&HF300
30 FOR I=&HF360 TO &HF36E
40 READ A$:POKE I,VAL("&H"+A$):NEXT
50 DATA F5,21,E1,F3,7E,EE,01,47
60 DATA 0E,02,CD,47,00,F1,C9
70 POKE &HFDA0,&H60
80 POKE &HFDA1,&HF3
90 REM aktief met POKE &HFD9F,&HC3
100 REM uit met POKE &HFD9F,&HC9
```

Dit programma veroorzaakt een omschakeling van de naamtabel. Vanzelfsprekend kunnen ook alle andere tabellen omgeschakeld worden. Daarvoor zijn de volgende wijzigingen nodig:

Het onderhavige registernummer moet als tweede kode in regel 60 ingegeven worden (nu 02).
De tweede tot vierde kode van regel 50 stelt de machine-instruktie LD HL, &HF3E1 voor. Dat is het RAM adres, waarin de aktuele waarde van het VDP register 2 is opgeslagen. Het is noodzakelijk alle inhouden van de schrijfregisters van de VDP additioneel in RAM op te slaan, omdat de registers van de VDP niet rechtstreeks gelezen kunnen worden. Geef daartoe dan mee

```
PRINT VDP(2)
```

Als U met

```
PRINT VDP(2)
```

de waarde van register 2 wilt uitlezen, dan wordt de inhoud intern door het lezen van de geheugenpositie &HF3E1 verstrekt. In overeenstemming hiermee wordt de waarde van register 0 op adres &HF3DF en algemeen de waarde van register N op adres (&HF3DF+N) opgeslagen.

Hieruit volgt dat het derde en vierde hexgetal van regel 50 dienovereenkomstig veranderd moeten worden. Daarbij is eerst de low byte (in ons geval dus &HE1) en dan pas de high byte &HF3 opgeslagen.

Door handig te programmeren kunt U bereiken dat verschillende schrijfvormen gelijktijdig op het beeldscherm verschijnen, o.a. cursief of vetschrift. Ook het onderlijnen van woorden is op deze manier mogelijk.
Dezelfde methode is ook te gebruiken om meer dan 32 (namelijk 64!) verschillende sprites gelijktijdig weer te geven.

Nu echter voorlopig genoeg met de heen en weer schakelarij. Tot slot van dit hoofdstuk zouden we U graag een programma geven, waarmee U een hardcopy van de actuele beeldscherminhoud kunt vervaardigen.

```
20000 REM SUB Texthardcopy
20010 DEFINT A-Z
20020 SN=PEEK(&HFCAF)
20030 IF SN>1 THEN RETURN
20040 BA=BASE(SN*5)
20050 AS=40-8*SN
```

```
20060 FOR Z=0 TO 23
20070 FOR S=0 TO AS-1
20080 BY=VPEEK(BA+Z*AS+S)
20090 IF BY>31 THEN LPRINT CHR$(BY);
20100 NEXT
20110 LPRINT
20120 NEXT
20130 RETURN
```

Het bijzondere van dit programma is dat het zowel in >SCREEN 0< als in >SCREEN 1< loopt. Het programma stelt automatisch vast welke mode ingeschakeld is. Hiervoor wordt het adres &HFCAF uitgelezen. Dit adres bevat het nummer van de aktueel ingeschakelde mode.
We hebben dit programma met opzet als een subroutine ingegeven, zodat U het direkt in Uw eigen programma's kunt inbouwen.
In regel 20 wordt het aktuele >SCREEN< nummer met >PEEK (&HFCAF)< gevonden. Dit is niet de enige mogelijkheid de mode vast te stellen. Omdat de VDP het beeld moet samenstellen moet ook in de VDP registers het kengetal van de mode opgeslagen zijn. Onder andere worden hiervoor de VDP registers 0 en 1 gebruikt. Ze worden ook wel als commandoregister betiteld. Voor de keus van de mode zijn bit 3 en bit 4 van register 1 en bit 6 van register 0 aangewezen. De bits 3 en 4 worden ook als M1 en M2 en bit 6 van register 0 als M3 aangeduid.

Zo komen we aan de volgende tabel:

	M1	M2	M3
Tekstmode	1	0	0
Grafiek 1	0	0	0
Grafiek 2	0	0	1
Multicolor	0	1	0

M1: Bit 3 van register 1

M2: Bit 4 van register 1

M3: Bit 6 van register 0

2.4 De grafiek mode 2

De grafiek mode 2 biedt de hoogst mogelijke oplossing van 256*192 punten. Daarbij kunnen alle kleuren gelijktijdig en bovendien nog sprites gebruikt worden. In principe is de grafiek mode 2 gelijk aan de grafiek mode 1. Bij de grafiek mode 2 is echter een grotere tekengenerator voorzien, zodat voor elk van de 768 (32*24) tekenposities op het beeldscherm een eigen, van alle andere tekens te onderscheiden, teken gedefinieerd kan worden. Dat betekent dat elk van de 49152 beeldpunten geplaatst, resp. teruggeplaatst kan worden. Bovendien bestaat de mogelijkheid voor een 8*8 teken verschillende kleuren te bepalen. Aan elk byte (8 punten) van dit uit 8 bytes bestaande teken, kunnen twee gekozen kleuren toegewezen worden. Hierdoor zijn de tekengenerator en de kleurtabel &H1800 bytes lang.

VRAM opdeling

Evenals in de grafiek mode 1, bestaat de naamtabel uit de grafiek mode 2 uit 768 elementen, die exact uit de 768 tekenposities van het beeldscherm bestaan. Omdat in de grafiek mode 1 de modelnamen exact 8 bits lang zijn, kunnen maximaal 256 verschillende tekendefinities, op de in het laatste hoofdstuk besproken manier, geadresseerd worden.

In de grafiek mode 2 moeten nu toch 768 verschillende tekendefinities geadresseerd worden. Daarvoor wordt het beeldscherm in 3 gelijke delen van 256 tekenposities verdeeld. Op dezelfde wijze worden ook de naam-, kleur- en tekentabel verdeeld. Het bovenste deel van de tabellen komt dan overeen met het bovendeel, de middelste tabel met het middendeel, de onderste tabel op het onderste deel van het beeldscherm. Met behulp van deze truc is het nu mogelijk 768 verschillende tekens weer te geven. Het nadeel van deze handelwijze is dat de drie delen onafhankelijk van elkaar zijn. Een tekendefenitie van een deel kan niet direkt in een ander deel weergegeven worden. Zoals we verder zullen zien speelt dit in de meeste gevallen geen rol van betekenis.

Bekijken we eens een voorbeeld:

In een naamtabel staat op de 300e plaats, dus in het middendeel, de waarde 10. Als gevolg hiervan wordt op de 300e positie van het beeldscherm (regel 9, kolom 12) het teken met kode 10, d.w.z. het tiende teken van het

middendeel van de tekengenerator aangegeven. De kleur van het teken wordt door de bytes 10*8 tot 10*8+7 van het middendeel via de kleurtabel bepaald.

	Naamtabel	tekengenerator	kleurtabel	beeldscherm
1	0 : 256	0 : 2047	0 : 2047	0e regel : : 7e regel
2	256 : 300=(10) : : 511	2048 : ->2047+10*8 tot 2047+10*8+7 : 4095	2048 : ->2047+10*8 tot 2047+10*8+7 : 4095	8e regel : op regel 9 kolom 12 wordt het over- eenkomend teken in de overeen-
3	512 : 767	4096 : 6143	4096 : 6143	komstige kleur weergegeven.

Tot elke 8 punten, dus een byte, van de tekengenerator behoren 2 kleuren (2*4bit= 1byte) van de kleurentabel. Daarbij bepaalt de waarde van de onderste 4 bits de kleur van een geplaatste punt, de waarde van de 4 bovenste bits de kleur van een teruggeplaatste punt. Laat ons het navolgende programma eens wat nader bekijken, dan wordt de samenhang iets duidelijker verklaard.

```
10 SCREEN 2
20 READ X,Y:REM Position
30 NT=BASE(10):
40 FT=BASE(11):
50 ZG=BASE(12):
60 BN=(32*Y+X)*8: REM Bytenummer
70 FOR I=0 TO 7
80 READ B,VF,HF: REM Byte, voorgrond, achtergrond
90 VPOKE ZG+BN+I,B: REM tekendefenitie
100 VPOKE FT+BN+I,HF*16+VF: REM kleuren
```

```
110 NEXT I
120 N=(32*Y+X) MOD 256: REM tekencode
130 FOR I=NT TO NT+767
140 VPOKE I,N
150 NEXT I
160 IF INKEY$="" THEN 160
170 DATA 9,12: REM positie
180 DATA &B00010000,3,15
190 DATA &B00101000,3,15
200 DATA &B01000100,3,15
210 DATA &B10010010,3,13
220 DATA &B01000100,3,15
230 DATA &B00101000,3,15
240 DATA &B00010000,3,15
250 DATA &B00010000,11,3
```

De waarden van de regels 170 tot 250 kunt U naar believen wijzigen, maar let op wat er dan gebeurt. Vervolgens wordt door de lus van de regels 70 tot 110 de definiëring van de tekens overgebracht in de >VRAM<. Let erop dat hier niet de naamtabel gebruikt wordt. Zodra de mode >SCREEN 2< ingeschakeld wordt nummert de systeemroutine elk derde deel van de naamtabel van 0 tot 255 door.
Vanaf dit tijdstip blijft de naamtabel ongewijzigd. Dat betekent dat alle wijzigingen op het beeldscherm direkt in de tekengenerator worden opgenomen. Dit is het principiële verschil van de grafiek mode 2 ten opzichte van de tekstmode of de grafiek mode 1. Bij deze laatste wordt in normale gevallen met een vaste tekenset en een vaste kleurtabel gewerkt. Uiteraard wordt de naamtabel voortdurend veranderd.

In de grafiek mode 2 wordt meestal de naamtabel ongewijzigd gelaten en alle veranderingen in de tekengenerator en de kleurentabel doorgevoerd. Dat een verandering van de naamtabel mogelijk is wordt aangetoond door de regels 120 tot 150 van het bovenstaande programma. De totale naamtabel wordt met de namen van de eerder gedefinieerde tekens gevuld. Het bovenste en onderste derde deel blijft dan leeg, omdat de definiëring uitsluitend voor het middendeel geldt. Dit wordt dan ook volledig met de gedefinieerde tekens gevuld.

Probeert u nu het volgende programma.

```
5 REM
10 SCREEN 2
20 NT=BASE(10)
30 S=7:GOSUB 210
40 GOSUB 170
50 S=20:GOSUB 210
60 GOSUB 120:
70 S=11:GOSUB 210
80 GOSUB 170
90 GOSUB 120
100 IF INKEY$="" THEN 100
110 END
120 FOR J=0 TO 2
130 FOR I=0 TO 255
140 VPOKE NT+J*256+I,I
150 NEXT I,J
160 RETURN
170 FOR I=NT TO NT+767
180 VPOKE I,RND(1)*256
190 NEXT I
200 RETURN
210 FOR I=10 TO 80 STEP S
220 CIRCLE (128,96),I
230 NEXT
240 RETURN
```

Ga eens na wat er aan de hand van het getoonde op het beeldscherm intern gebeurt. Zoals U ziet gaan de BASIC grafiek instruktieroutines steeds uit van dezelfde voorgeschreven, genummerde naamtabel.
De volgorde in de naamtabel wordt nooit gewijzigd. Doen we bijv. hetzelfde met de routine vanaf regel 170, dan wordt het bestaande beeld overhoop gegooid. Wordt er dan zonder nummering verder getekend, dan ontstaan in geen geval cirkels (regel 50). Eerst na het opnieuw nummeren (regel 60) wordt het beeld echt zichtbaar.
Het is dus niet aan te raden iets in de naamtabel te wijzigen, als BASIC routines gebruikt moeten worden. Alle veranderingen moeten direkt in de tekengenerator en de kleurtabel doorgevoerd worden.

De standaard indeling van de VRAM in de hoogoplossende mode is:

&H0 - &H1800	Tekengenerator
&H1800 - &H1B00	Naamtabel
&H1B00 - &H1B80	Sprite attribuut
&H1B80 - &H2000	vrij
&H2000 - &H3800	Kleurtabel
&H3800 - &H4000	Sprite model

Het basisadres van de naamtabel wordt, zoals gebruikelijk, door het VDP-register 2 bepaald. Voor de tekengenerator en de kleurtabel geldt de volgende afwijkende regeling:
Omdat beide tabellen 6K groot zijn, kunnen ze in 8K stappen verschoven worden. d.w.z. ze beginnen ofwel bij adres 0 of bij adres &H2000. Om beide mogelijkheden te kunnen onderscheiden wordt telkens het hoogst gebruikte bit genomen. 1 betekent daarbij vanaf adres &H2000, een 0 vanaf adres 0. Bij de kleurtabel is dit bit nr. 7 van register 3, bij de tekengenerator bit nr. 7 van register 4. Alle laagwaardige bits moeten bovendien op 1 geplaatst zijn, anders treden zeer merkwaardige effecten op, die ons maar zelden van nut kunnen zijn.
Met deze standaardindeling is de VRAM nagenoeg geheel gebruikt. Het is echter mogelijk bijv. twee naamtabellen of zelfs meerdere attributentabellen onder te brengen. Wordt afgezien van het gebruik van sprites, dan wordt dit aantal aanzienlijk verhoogd. Een verplaatsing is, zoals reeds gezien, door verandering van de VDP met de >VDP< instruktie, maar ook met een >BASE< instruktie mogelijk.
Jammer genoeg is hier weer een fout te melden. Opdat >BASE< bij een ingeschakelde >SCREEN 2< korrekt uitgevoerd wordt, moet direkt daar aan volgend de systeemroutine vanaf adres &H007E opgeroepen worden. Zij plaatst de VDP-registers op de juiste waarden. Dus:

```
DEF USR8=&H7E
BASE( . . .)= . . .: X= USR8(1)
```

Het gebruik van deze omweg is noodzakelijk als >SCREEN 2< ingeschakeld is en de aktuele beeldscherminhoud niet veranderd mag worden. Wordt hieraan geen waarde gehecht dan is ook

```
BASE( . . .)= . . .: SCREEN 2
```

mogelijk.

De adressen van de BASIC interpreter voor naamtabel en tekengenerator zijn &HF922/3 en &HF924/5.
Om de tekenprestaties van uw computer beter te begrijpen en te benutten, geven we hierna nog enige gebruikersprogramma's.

Grafiek editor

Eerst volgt een programma, waarbij U zonder programmeerkennis direkt op het beeldscherm tekeningen kunt maken. Het gebruik van het programma is volgens het concept van de 'muis' opgebouwd. Dat betekent dat U voortdurend een symbool, o.a. een pijl, met behulp van de joystick of de cursortoetsen over het beeldscherm beweegt. Alle funkties van het programma worden opgeroepen als U dat symbool op een gemarkeerde positie plaatst en de vuurknop, resp. de spatietoets indrukt. Aan de benedenrand van het scherm ziet u alle 16 kleuren. Kies dan met de pijl de aktuele schrijfkleur uit. Met het indrukken van de vuurknop wordt de kleur gekozen die de pijlpunt aanwijst. Naast het kleurenscala bevinden zich de afkortingen Cu en Ra. Zij staan in de aktuele kleur aangegeven. Door het kiezen van Cu op de bekende manier (erheen lopen en vuren) krijgt U, in plaats van de pijl, een C. Het kiezen van een kleur met de C (Cursor) neemt de dan weer verschijnende pijl de gekozen kleur aan. Bij de keus van Ra krijgt U een R als symbool voor de raamkleur en kunt U de raamkleur instellen.

Aan de bovenrand van het beeldscherm krijgt U de volgende uitvoer:

PUN/LIN/REC/PAI/TEX/

PUN- Puntmode

Deze wordt door de pijl gekenmerkt. Bevindt U zich in het tekenveld dan wordt in de puntmode bij het indrukken van de vuurknop een punt in de aktuele kleur geplaatst.

LIN- Lijnmode

Gaat u met het symbool naar LIN en drukt U dan op de vuurknop dan krijgt U als symbool een pijl met een gat in de punt. Hiermee kunnen lijnen getrokken worden. Door het indrukken van de vuurknop markeert U het begin en het einde van een lijn. Bij de tweede knopdruk worden beide punten met elkaar verbonden. De afzonderlijke modi blijven ingeschakeld totdat een nieuwe mode gekozen wordt.

REC- Rechthoek

De REC mode wordt door het rechthoeksymbool gekenmerkt. Analoog aan LIN worden 2 punten gemarkeerd. In deze mode wordt echter de rechthoek, die als denkbeeldige lijn een diagonaal bezit, getekend.

PAI- PAINT

Na het kiezen van de PAI-funktie krijgt U een opgevulde rechthoek. Een door de aktuele kleur begrensd gebied wordt met de aktuele kleur gevuld. Let erop dat de te vullen gebieden werkelijk gesloten zijn, omdat anders Uw gehele beeldscherm opgevuld gaat worden.

TEX- TEKSTMODE

Na de keuze van TEX ontvangt u een T als cursor, waarmee letters of tekens uitgevoerd kunnen worden. Na het indrukken van de vuurknop wordt op het indrukken van een toets gewacht. Het daarbij behorende teken wordt dan op de aktuele positie weergegeven.

```
10 DEFINT A-Z
20 EI=1:REM JOY
30 CC=1:REM SPRITE CURSOR COLOR
40 OPEN "grp:" FOR OUTPUT AS #1
50 SCREEN 2,0:COLOR 15,4,4
60 FOR I=0 TO 2
70 READ CH$:CH=ASC(CH$):BA=&H1BBF+8*CH
80 FOR J=0 TO 7
90 A$=A$+CHR$(PEEK(BA+J)):NEXT J
100 READ N:SPRITE$(N)=A$:A$=""
110 NEXT I
120 FOR I=0 TO 7:READ A:A$=A$+CHR$(A):NEXT
130 SPRITE$(0)=A$
140 A$=CHR$(ASC(A$))+CHR$(&H48)+CHR$(&H48)+RIGHT$(A$,5)
150 SPRITE$(3)=A$
160 SPRITE$(4)=CHR$(&H7F)+STRING$(6,&H41)+CHR$(&H7F)
170 SPRITE$(5)=STRING$(8,&H7F)
180 X=120:Y=80:XA=X:YA=Y
190 READ XL,XR,YO,YU
200 READ JO,JU
210 LINE (XL,YO+JO)-(XR,YU-JU),2,B
```

```
220 FOR I=0 TO 15:LINE (16+I*8,YU-1)-(22+I*8,YU+7),I,BF:
NEXT
230 SF=15:HF=4:GOSUB 780
240 ON STRIG GOSUB 370,370,370,370,370:STRIG(EI) ON
250 PSET (16,YO):PRINT#1,"Pun/Lin/Rec/Pai/Tex/"
260 PUT SPRITE 0,(X,Y),CC,M
270 RI=STICK(EI):IF RI=0 THEN 270
280 IF RI>1 AND RI<5 THEN X=X+XP:XP=XP+2 ELSE XP=1
290 IF RI>3 AND RI<7 THEN Y=Y+YP:YP=YP+2 ELSE YP=1
300 IF RI>5 AND RI<9 THEN X=X-XM:XM=XM+2 ELSE XM=1
310 IF RI=1 OR RI=2 OR RI=8  THEN Y=Y-YM:YM=YM+2 ELSE YM
=1
320 IF X>XR THEN X=XR
330 IF X<XL THEN X=XL
340 IF Y>YU THEN Y=YU
350 IF Y<YO THEN Y=YO
360 GOTO 260
370 REM strig
380 YM=0:YP=0:XM=0:XP=0
390 IF Y>YO+JO AND Y<YU-JU THEN 610
400 IF Y<=YO+JO THEN 520
410 PO=INT((X-16)/8)
420 IF PO<0 THEN RETURN
430 IF PO>15 THEN 490
440 IF M<>1 AND M<>2 THEN SF=PO:GOSUB 780:RETURN
450 IF M=1 THEN COLOR ,,PO
460 IF M=2 THEN CC=PO
470 GOSUB 780
480 M=MA:RETURN
490 IF PO=16 THEN RETURN
500 IF PO<23 THEN MA=M:M=INT((PO-14)/3):RETURN
510 RETURN
520 REM
530 PO=INT((X-16)/32)
540 NF=0:RF=0:LF=0:PF=0:TF=0
550 IF PO<1 THEN NF=-1:M=0:RETURN
560 IF PO<2 THEN LF=-1:M=3:RETURN
570 IF PO<3 THEN RF=-1:M=4:RETURN
580 IF PO<4 THEN PF=-1:M=5:RETURN
590 IF PO<5 THEN TF=-1:M=6:RETURN
600 RETURN
610 REM
```

```
620 IF M=1 OR M=2 THEN RETURN
630 IF NF THEN 700
640 IF TF THEN GOSUB 720:RETURN
650 IF LF=1 THEN LINE (XA,YA)-(X,Y),SF:LF=-1:RETURN
660 IF LF=-1 THEN XA=X:YA=Y:LF=1
670 IF RF=1 THEN LINE (XA,YA)-(X,Y),SF,B:RF=-1:RETURN
680 IF RF=-1 THEN XA=X:YA=Y:RF=1
690 IF PF THEN PAINT (X,Y),SF:RETURN
700 PSET (X,Y),SF
710 RETURN
720 REM
730 POKE &HF3F8,PEEK(&HF3FA):POKE &HF3F9,PEEK(&HF3FB)
740 PSET(X,Y),0
750 A$=INKEY$:IF A$="" THEN 750
760 PRINT #1,A$;
770 RETURN
780 REM
790 LINE (150,184)-(198,192),HF,BF
800 COLOR SF,HF
810 PSET (152,184),SF:PRINT#1,"Ra.Cu."
820 RETURN
830 DATA R,1,C,2,T,6
840 DATA &B01111100
850 DATA &B01111000
860 DATA &B01111000
870 DATA &B01111100
880 DATA &B01001110
890 DATA &B00000111
900 DATA &B00000011
910 DATA &B00000000
920 DATA 0,249,0,185
930 DATA 10,4
```

Programmabeschrijving:

Regel 40
: Grafiekuitvoer. Opent gegevens voor de uitvoer van tekst naar het beeldscherm.

Regel 60
: Lus voor het definiëren van drie sprites die letters voorstellen.

Regel 70

Het teken wordt in CH$ gelezen en de daarbij behorende ASCIIkode wordt in CH opgeslagen. BA bevat het basisadres van het aktuele teken in de ROM tekengenerator (startadres &H1BBF).

Regel 80

Lus voor het uitlezen van 8 bytes per teken.

Regel 90

Veranderen van de tekendefinitie in een string (A$) voor de spritedefinitie.

Regel 100

Spritenummer lezen, sprite definiëren en A$ wissen.

Regel 120, 130

Een pijl als sprite definiëren.

Regel 140, 150

Maakt een pijl met gat in de punt, die LIN aangeeft.

Regel 160

Vervaardigt de open rechthoek die REC aangeeft.

Regel 170

Vervaardigt de opgevulde rechthoek voor 'PAI'.

Regel 180

X, Z: startpositie van de sprite
XA, YA: oude positie voor het tekenen van lijn/rechthoek.

Regel 190

Leest beeldschermbegrenzing: XL=links, XR=rechts, YO=boven,YU=onder.

Regel 200

Verschillen van totaal beeldscherm tot tekenvlak.
JO=boven, JU=onder.

Regel 210

Tekent het kader

Regel 220

Tekent de kleurrechthoekjes aan de onderste beeldschermrand.

Regel 230

SF=Schriftkleur en HF=achtergrondkleur worden op hun standaardwaarden gezet.

Regel 240

Interruptdefinitie van vuurknop of spatie en toestaan van interrupts.

Regel 260

M = aktueel spritenummer.

CC = cursorkleur
Aangeven van de gegeven cursorsprites.

Regel 270 tot 360
Besturing van de cursorsprites, waarbij XP, YP, XM en YM de snelheid van de gegeven bewegingsrichting veranderen. De snelheid stijgt bij het langer vasthouden van een stuurrichting.

Regel 370
Interruptroutine ter behandeling van de 'strig'.

Regel 380
Snelheid van de cursorsprite op 0 terugplaatsen.

Regel 390
Nagaan of de cursorsprite in het tekenveld is.

Regel 400
Nagaan of de cursorsprite boven het tekendeel van het tekenveld is, daarna naar een interpretatie van de instruktievolgorde springen op regel 520.

Regel 410
Positie op de onderste regel vaststellen.

Regel 420
Indien links van de kleurvelden, dan ongeldig.

Regel 430
Indien rechts van de kleurvelden dan doorgaan met 490

Regel 440
Plaatsen van de schrijfkleur door cursorsprite. (SF=PO, schriftkleur-=veldkleur). Sprong naar de uitvoer shbc

Regel 450
Kaderkleur plaatsen

Regel 460
Cursorspritekleur plaatsen

Regel 470
Naar kleuruitvoer springen

Regel 480
Oude sprite oproepen

Regel 490
Bij ongeldige positie in onderste instruktieveld gebeurt er niets.

Regel 500
Herkenning instruktie voor 'CU' en 'RA'

Regel 530
Positie binnen de bovenste instructierij vaststellen

Regel 540

NF=Normaalmode; LF=lijnmode; RF= rechthoekmode; PF=paint-mode; TF=tekstmode.

Regel 550

Als de spritecursor op 'PUN' staat dan punt plaatsen.

Regel 560 tot 590

Indien op LIN dan spritenummer 3 oproepen en mode-flag plaatsen.

Regel 620

Niet toegestane sprites uitsluiten (C en R)

Regel 630

Bij HF=–1 naar 700 gaan. In de normaalmode wordt direkt een punt geplaatst.

Regel 640

Indien in tekstmode, naar 720 springen.

Regel 650

Tweede punt van de lijn X,Y met de eerste punt van XA, XY verbinden.

Regel 660

Registreert de eerste punt en plaatst LF op 1, zodat de volgende keer de verbinding gelegd wordt.

Regel 670-680

Dezelfde techniek als in regels 650 en 660

Regel 690

Veld kleuren.

Regel 700

Punt plaatsen.

Regel 720

Tekstuitvoer.

Regel 730

Tekenbuffer wissen.

Regel 740

Aktuele positie vastleggen.

Regel 750

Toetsenvraag.

Regel 760

Uitvoer per bestand op het beeldscherm.

Regel 780

Uitvoer van schrift- achtergrond- en kaderkleur.

Regel 800

Kleur plaatsen.

Regel 810
De onderste opdrachtwoorden "Ra.Cu" uitvoeren.
Regels 830 tot 930
Spritedefinitie voor de pijl.

Driedimensionaal tekenen

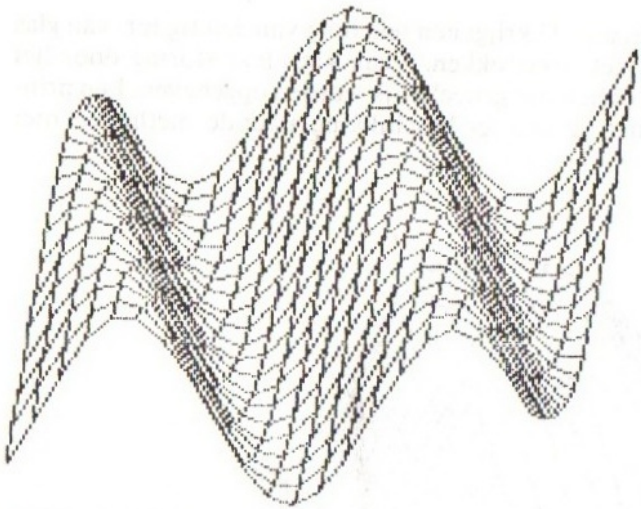
Een van de mooiste en spannendste toepassingen van de grafiek is het weergeven van driedimensionale vormen of funkties. Jammer genoeg duurt het aanmaken van deze tekeningen bij veel programma's vaak uren, ja zelfs dagen.

Het volgende programma vraagt in zijn basisversie slechts enige minuten om een compleet beeld aan te maken.
De grondslag van de weergave van 3D funkties is het idee om doorsneden aan een twee dimensionele tekening toe te voegen. De 3D funktie wordt op regelmatige afstanden langs de XY-as doorsneden. De bij het snijden ontstane lijn wordt eerst normaal getoond, de volgende snede wordt iets naar achteren verlegd. De daaruit resulterende lijn wordt, om het 3D-effect te laten ontstaan, enigszins verplaatst ten opzichte van de voorgaande. Deze handeling wordt herhaald totdat het te tekenen deel de Z-as afdekt. Deze methode levert een 3D lijnengrafiek.
Vaak is dit niet voldoende om een realistische weergave te verkrijgen. Dit wordt pas bereikt door het toepassen van een netgrafiek. Een voorbeeld van een netgrafiek is een getekende globe waarbij de lengte- en breedtegraden geaccentueerd worden. Het volgende programma tekent een netgrafiek van een willekeurige funktie voor een bepaalde interval. De omvang van het net bepaalt de tekensnelheid. Een zeer grof net is snel gemaakt, maar geeft alleen een benadering van het beeld van de funktie. Dit ligt opgesloten in de methode van het netgrafiek.

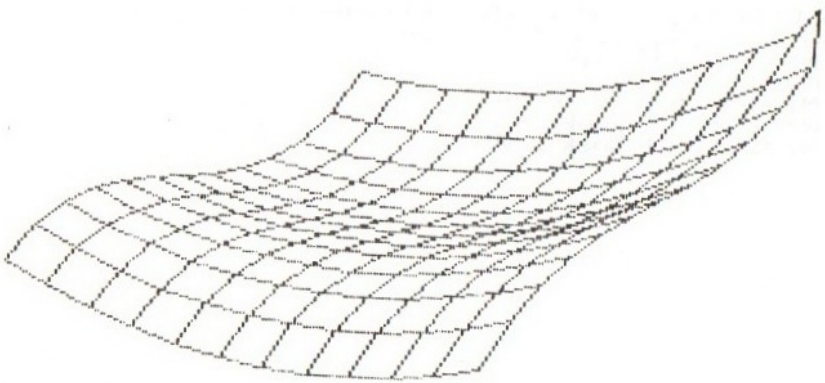
Bij een netgrafiek worden niet alle punten van de kromme berekend, vandaar het grote snelheidsvoordeel ten opzichte van gebruikelijke tekenmethoden. Slechts de knooppunten van het net worden berekend en daarna door lijnen verbonden. Principieel kunnen willekeurige funkties getekend worden. Men dient echter het waardebereik, d.w.z. de begrenzing van de assen, waarbinnen getekend moet worden, zodanig te kiezen dat er ook sprekende tekeningen ontstaan.
Bij veel funkties is het vaak storend dat de totale funktie, dus inclusief het eigenlijk verdekte gebied, wordt getekend. In werkelijkheid zien we ook alleen maar de naar ons toegekeerde zijde van een globe, niet de rugzijde,

tenzij hij van glas is. m.a.w. U krijgt een tekening van een figuur, van glas gebouwd en met een net overtrokken. Vaak kan deze storing door het veranderen van het gezichtspunt gedeeltelijk worden opgeheven. Een principiële oplossing is met de ons ter beschikking staande methoden niet mogelijk.

```
40 DEFFNF(X,Z)=SIN(SQR(X*X+Z*Z))
90 DATA -5,5,-1.9,1.9,-2,4
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM hoek X-AS
140 WZ=52:REM hoek Z-AS
```



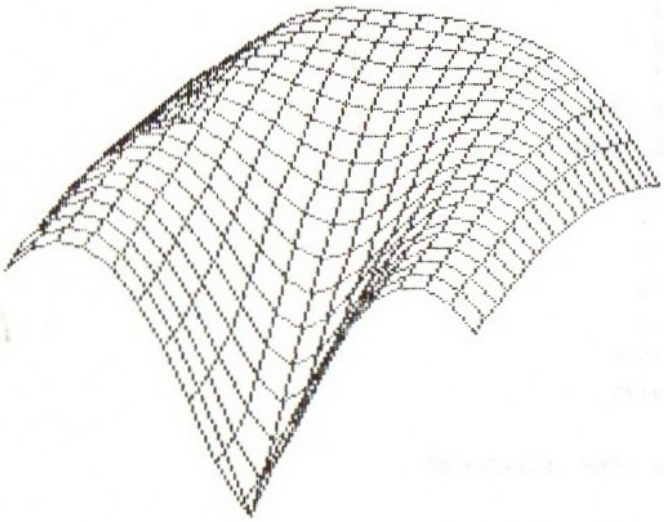
```
40 DEFFNF(X,Z)=SIN(X-Z)
90 DATA -5,5,-1.9,1.9,-2,2
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=12:REM hoek X-AS
140 WZ=62:REM hoek Z-AS
```



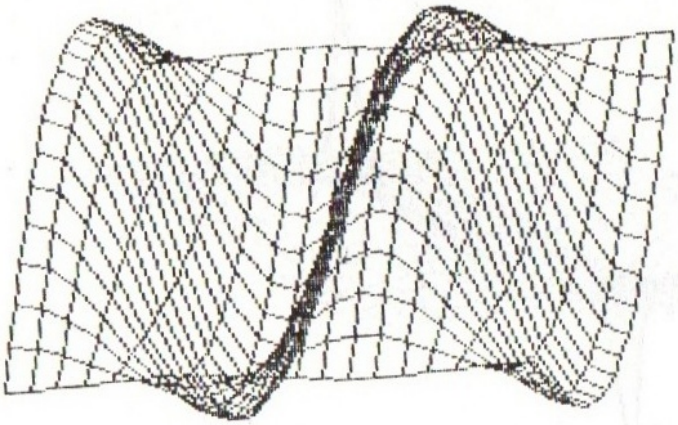
```
40 DEFFNF(X,Z)=(X-Z)^2+2*X-.05*Z*Z*Z+3
90 DATA -400,400,-500000,500000,-170,170
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=12:REM hoek X-AS
140 WZ=32:REM hoek Z-AS
```

```
10 '3-D
20 CLEAR 500,&HEFFF
30 DEFUSR1=&HF000
40 DEFFNF(X,Z)=SIN(SQR(X*X+Z*Z))
50 SCREEN 2
60 AP=18:REM aantal punten
70 DIM X(AP+1),Y(AP+1)
80 READ XA,XE,YA,YE,ZA,ZE:REM assenbegrenzing
90 DATA -2.1,2.1,0,1.3,-1.6,.1
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM hoek, X-as
140 WZ=52:REM hoek, Z-as
150 DE=3.141529*/180
160 ZX=COS(WZ*DE)^2
170 XX=COS(WX*DE)^2
180 ZY=SIN(WZ*DE)^2
190 XY=SIN(WX*DE)^2
200 XO=XA/(XA-XE)*256
210 YO=YE/(YE-YA)*192
220 J=0
230 FOR Z=ZE TO ZA STEP -(ZE-ZA)/AP
240 I=0
250 FOR X=XE TO XA STEP -(XE-XA)/AP
260 Y=FNF(X,Z)
270 XK=XX*MX*X-ZX*MZ*Z
280 YK=MY*Y-ZY*MZ*Z-XY*MX*X
290 XK=XK+XO:YK=YO-YK
300 IF I=0 THEN 320
310 LINE (XK,YK)-(X(I),Y(I))
320 I=I+1
330 IF J=0 THEN 350
340 LINE (XK,YK)-(X(I),Y(I))
350 X(I)=XK
```

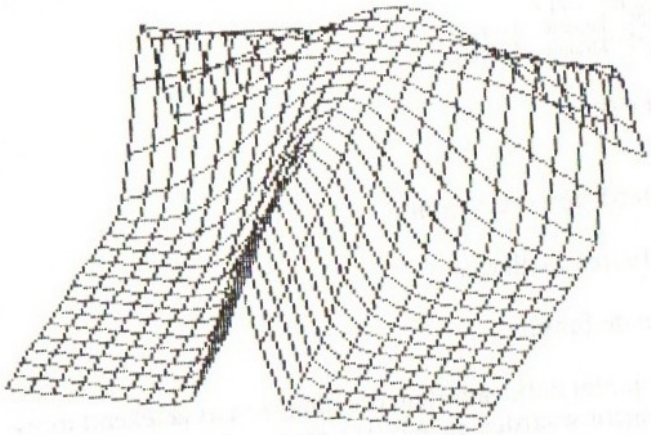
```
360 Y(I)=YK
370 NEXT X
380 J=J+1
390 NEXT Z
400 A$=INKEY$:IF A$="" THEN 400
410 IF A$<>"j" AND A$<>"J" THEN END
420 X=USR1(1)
430 END
```



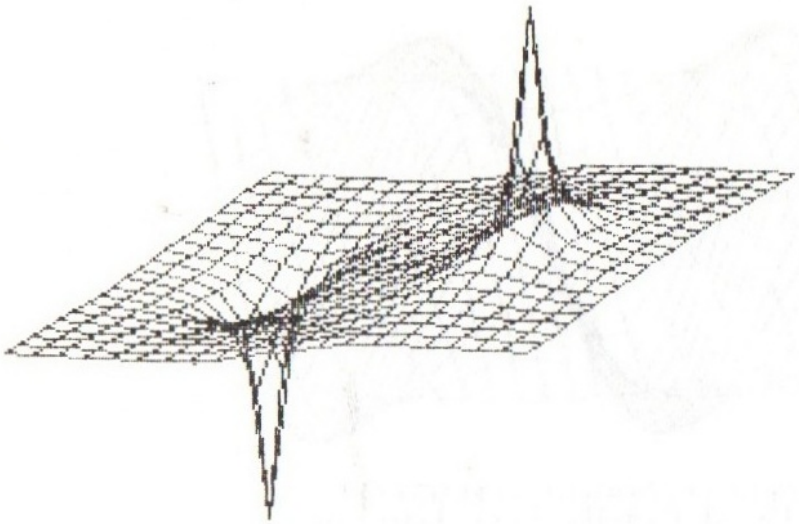
```
40 DEFFNF(X,Z)=SIN(SQR(X*X+Z*Z))
90 DATA -2.1,2.1,0,1.3,-1.6,.1
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM hoek X-AS
140 WZ=52:REM hoek Z-AS
```



```
40 DEFFNF(X,Z)=SIN(X)*COS(Z)
90 DATA -4.8,4.8,-1,1,-1.6,1.6
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=170/(ZE-ZA)
130 WX=10:REM hoek X-AS
140 WZ=62-WX/2:REM hoek Z-AS
```



```
40 DEFFNF(X,Z)=EXP(-X*X*Z*Z)
90 DATA -3,3,-1.2,1.2,-.5,3
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM hoek  X-AS
140 WZ=52:REM hoek  Z-AS
```



```
40 DEFFNF(X,Z)=X/SQR((1-X*X)^2+
   (2*X*Z/300)^2+1E-03)
90 DATA -2,2,-7,7,-300,300
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=12:REM hoek X-AS
140 WZ=42:REM hoek Z-AS
```

Programma: 3D netgrafiek

Regel 20
: Ruimte voor hardcopy reserveren.

Regel 30
: Hardcopy startadres vastleggen.

Regel 40
: Definiëring van de funktie

Regel 60
: AP bepaalt het aantal netkwadraten, die langs een as getekend moeten worden. Hogere waarden in AP (tot max. 30) geven een beter beeld maar het tekenen duurt langer. Kleinere waarden (tot max. 10) geven een snel, maar grof inzicht in de funktie.

Regel 70
: Reservering van de veldvariabelen die telkens de coördinaten van het laatstelijk getekend netpunt opslaan.

Regel 80,90

Hier worden de assenbegrenzingen uit de daaronder gelegen DATA-regels ingelezen. Zij bepalen direct de vorm van de funktie die weergegeven wordt.

Komt geen bevredigend beeld tot stand, dan dienen de waarden vaak veranderd te worden. Indien U bij een onbekende funktie geen voorstelling over de keus van deze waarde heeft, gebruik dan redelijk grote segmenten.

Regel 100 tot 120

Maatstaffaktor voor het omrekenen in beeldschermcoördinaten voor X-, Y- en Z-as. De aangegeven getallen mogen (en moeten) slechts in uitzonderingsgevallen veranderd worden.

Regel 130

WX is de hoek die de X-as in de weergave horizontaal insluit. Deze hoek is medebepalend voor het aanzicht van de tekening.

Regel 140

WZ gebruikt de hoek, die de Z-as horizontaal insluit. Ook deze hoek is zeer bepalend voor het aanzien van het beeld.

Regel 150

Omrekeningsfaktor van graden naar radialen.

Regel 160 tot 190

Afhankelijk van de hoek WZ en WX van de assen worden hier de stuik- en strekfaktoren voor de afzonderlijke assenrichtingen bepaald. Ze bewerken dat afstanden, die in werkelijkheid onder een scherpe hoek bekeken worden, in de tekening dienovereenkomstig ingekort worden.

Regel 200, 210

XO en YO zijn samenvattingen die gebruikt worden om de reële coördinaten in beeldschermcoördinaten om te rekenen. Daarbij wordt rekening gehouden met de gekozen grenzen.

Regel 220

J=0 betekent dat de eerste lijn in de X-richting getekend wordt.

Regel 230

Door de Z-lus wordt het reële beeld zogezegd in schijven gesneden. De lijn die dan op de snede ontstaat wordt getekend.

Regel 240

I is de teller die het nummer van het aktuele netkwadraat bevat.

Regel 250

Door de X-lus wordt telkens een lijn van de Z-snede getekend.

Regel 260-290
Berekening van de funktiewaarde
Herrekening van 3D naar 2D
Herrekening van de beeldschermkwadraten.
Regel 300
Indien begin van de lijn (I=0) geen verbindingslijn, dan niet tekenen (Horizontaal)
Regel 310
Horizontale verbindingslijn tekenen.
Regel 330
Indien eerste lijn, dus geen verticale verbindingslijn tekenen.
Regel 340
Verticale verbindingslijn tekenen.
Regels 350, 360
Netpuntkoördinaten tekenen.
Regels 390 tot 430
Indien andere toets gedrukt dan j of J, hardcopy aanmaken, (hardcopy moet vooraf ingelezen zijn) anders END.

Programma: grafiek hardcopy in machinetaal

Om de met de grafiekeditor of de 3D funktieplotter gemaakte tekeningen op papier te kunnen brengen, volgt nu een hardcopyprogramma. Het programma loopt zonder verandering op de EPSON FX-80 printer en andere kompatibelen. Ter aanpassing aan andere printertypes, moet alleen de stuursequentie, die de printer op grafiekmode schakelt, aangepast worden, vooropgesteld dat een 8bits mode bestaat.
Voor de hardcopy wordt direkt de tekengenerator in hoogoplossend vermogen ingelezen. Zoals U weet, zijn daar 8 opeenvolgende bytes een teken en de volgende 8 het volgende teken enz. Het probleem bestaat uit het onder elkaar printen van 8 punten, en niet (zoals bij het beeldscherm) naast elkaar liggende punten.

Voorbeeld:

Kodes Beeldschermgeheugen.

00010000 = 16
00101000 = 40
01000100 = 68

```
10111010 = 186
01000100 =  68
00101000 =  40
00010000 =  16
11111110 = 254
!!!!! !!
!! ! 17!
17!147! !
 !! !! !
41!85!0   printerkode
  !   !
 85   41
```

Het teken wordt door:

```
FOR I=32 tot 39: READ A
VPOKE I,A: NEXT
DATA 16,40,68,16,68,40,16,254
```

op het beeldscherm weergegeven. Voor de printeruitvoer moeten kolomsgewijs gevormde waarden gezonden worden.

```
LPRINT CHR$(27);"K";CHR$(8);CHR$(0);
FOR I=0 to 7: READ A
LPRINT CHR$(A);:NEXT
DATA 17,41,85,147,85,41,17,0
```

De eerste >LPRINT< instruktie is de stuurkodereeks voor de EPSON FX-80, die de uitvoer van 8 grafiekkodes aankondigt.

Het hardcopyprogramma moet nu de gehele beeldschermtekensset stap voor stap volgens bovenvermeld voorschrift omzetten om kolomsgewijs te kunnen zenden. In BASIC kan dit wel enige uren duren, vandaar de oplossing in machinetaal.

```
F000              10                    ; hardcopy
F000              20    print   EQU     &H00A5
F000              30            ORG     &HF000
F000  210000      40            LD      HL TABLE1 ; stuurkode-
                                          tabel
```

```
F003 7E          50          LD    A, (HL)
F004 47          60          LD    B,A ; aantal stuurkodes
F005 23          70  NESTE1  INC   HL
F006 7E          80          LD    A, (HL)
F007 CDA500      90          CALL  PRINT
F00A 10F9        100         DJNZ  NESTE1
F00C 210000      110         LD    HL,&H0000 ; naamtabel
F00F 0618        120         LD    B,24 ; regelteller
F011 C5          130 NEZEIL  PUSH  BC
F012 E5          140         PUSH  HL
F013 210000      150         LD    HL,TABLE2 ; stuurkode-
                                   tabel
F016 7E          160         LD    A, (HL)
F017 47          170         LD    B,A ; aantal stuurkodes
F018 23          180 NESTE2  INC   HL
F019 7E          190         LD    A, (HL)
F01A CDA500      200         CALL  PRINT
F01D 10F9        210         DJNZ  NESTE2
F01F E1          220         POP   HL
F020 0620        230         LD    B,32 ; kolommenteller
F022 C5          240 NESPAL  PUSH  BC
F023 E5          250         PUSH  HL ; naampointer
F024 CDA500      260         CALL  &H0BA5 ; tekendefinitie
                                   in RAM kopiëren
F027 0608        270         LD    B,8 ; byteteller
F029 C5          280 NEBYTE  PUSH  BC
F02A 2118FC      290         LD    HL,&HFC18
F02  D-0608      300         LD    B,8 ; byteteller
F02F 97          310         SUB   A ; accu wissen
F030 CB06        320 NEBIT   RLC   (HL); bit in carry
F032 17          330         RLA    ; carry in accu
F033 23          340         INC   HL
F034 10FA        350 DJNZ    NEBIT
F036 CDA500      360         CALL  PRINT
F039 C1          370         POP   BC ; byteteller
F03A 10ED        380         DJNZ  NEBYTE
F03C E1          390         POP   HL; naamtabelpointer
F03D 110800      400         LD    DE,8
F040 19          410         ADD   HL,DE
F041 C1          420         POP   BC ; kolommenteller
```

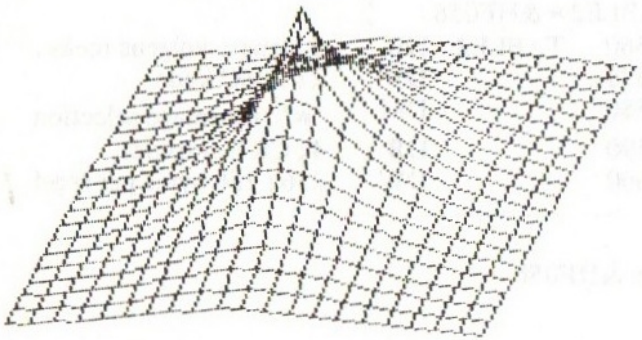
```
Fo42  10DE      430              DJNZ  NESPAL
F044  3E0A      440              LD    A,10 ; line feed
F046  CDA500    450              CALL  PRINT
F049  3E0D      460              LD    A,13
F04B  CDA500    470              CALL  PRINT
F04E  C1        480              POP   BC ; regelteller
F04F  10C0      490              DJNZ  NEZEIL
F051  C9        500              RET
F052            510                    ; stuurkodetabellen
*****regel 40 TABLE1 = &HF052
F052  03        520   TABLE1     DB    3 ; lengte van de tabel
F053  1B        530               DB    27 ; ESC kode
F054  41        540               DM    "A" ; regelopvoer
F055  08        550               DB    8 ; 8/72 inch
***** regel 150 TABLE2 = &HF056
F056  05        560   TABLE2     DB    5 ; lengte volgens reeks
F057  1B        570               DB    27 ; ESC kode
F058  2A        580               DM    "*" ; bit-image selection
F059  04        590               DB    4 ; CRT graphics
F05A  0001      600               DW    256 ; 256 bytes per regel
```

Programma harcop
Start: &HF000 einde: &HF05b
lengte: &H5C bytes
fouten: 0
Variabelentabel:

PRINT	00A5	NESTE1	F005
NEZEIL	F011	NESTE2	F018
NESPAL	F022	NEBYTE	F029
NEBIT	F030	TABLE1	F052
TABLE2	F056		

```
10 CLEAR 200,&HEFFF:MAXFILES=1
20 FOR I=&HF000 TO &HF05B
30 READ A$:W=VAL("&H"+A$)
40 S=S+W:POKE I,W:NEXT
50 IF S<>8670 THEN BEEP:PRINT"Fout in Data's":END
60 PRINT"Alles OK !"
```

```
70 DEFUSR1=&HF000
80 DATA 21,52,F0,7E,47,23,7E,CD
90 DATA A5,00,10,F9,21,00,00,06
100 DATA 18,C5,E5,21,56,F0,7E,47
110 DATA 23,7E,CD,A5,00,10,F9,E1
120 DATA 06,20,C5,E5,CD,A5,0B,06
130 DATA 08,C5,21,18,FC,06,08,97
140 DATA CB,06,17,23,10,FA,CD,A5
150 DATA 00,C1,10,ED,E1,11,08,00
160 DATA 19,C1,10,DE,3E,0A,CD,A5
170 DATA 00,3E,0D,CD,A5,00,C1,10
180 DATA C0,C9,03,1B,41,08,05,1B
190 DATA 2A,04,00,01
```



```
40 DEFFNF(X,Z)=EXP(-SQR(X*X+Z*Z))
90 DATA -3,3,-0.9,0.9,-2,2
100 MX=170/(XE-XA)
110 MY=130/(YE-YA)
120 MZ=150/(ZE-ZA)
130 WX=22:REM hoek   X-AS
140 WZ=52:REM hoek   Z-AS
```

2.5 De multicolor mode

Deze mode biedt een uitvoer van 64*48 gekleurde kwadraten. Eik kwadraat bestaat daarbij uit 4*4 punten. De kleur van elk van deze kwadraten kan een der mogelijke 16 kleuren zijn. Hierdoor kunnen alle 16 kleuren gelijktijdig gebruikt worden, de sprites staan volledig ter beschikking.
De naamtabel is dezelfde als bij de beide grafiekmodi. Zij bestaat uit 768 invoergegevens, waarbij de naam niet meer in een kleurentabel, maar

alleen nog in de tekengenerator staat. De kleur wordt aldus door de tekengenerator bepaald. Omdat een tekendefinitie gebruikelijk 8 bytes lang is, staat een naam op een 8 bytes gedeelte van de tekengenerator.
Alleen 2 van deze 8 bytes bepalen telkens de beeldschermuitvoer. Deze beide bytes bepalen 4 kleuren, waarbij elk kwadraat de omvang van 4*4 punten bedekt. De 4 MSB's (hoogwaardige 4-7) bepalen de kleur van het kwart links boven van de tekenpositie. De 4 laagwaardige bits bepalen de kleur van het kwartdeel rechts boven. Het tweede byte definieert de kleuren van het linker en rechter onderstuk.
De posities van de betrokken 2e bytes, binnen het 8 bytes deelstuk dat de naam aanwijst, hangt af van de positie van de naam in de naamtabel, dus van de betreffende beeldschermpositie. Voor namen van de eerste regel, (naam 0-31) worden de eerste twee bytes van het 8byte deelstuk gebruikt.

De volgende regel van namen (32-63) komt overeen met beeldschermregel 2 en gebruikt byte 3 en 4. De volgende regel gebruikt derhalve byte 5 en 6 en uiteraard de laatste regel byte 7 en 8. Dit schema wordt voor het gehele beeldscherm voortgezet.
De bepaling van het VDP register en het gebruik van de >BASE< instruktie is analoog aan hetgeen tot nu toe is geleerd. Overigens is er van deze mode niets te vertellen. De oplossing is hier veel te grof om er interessante grafieken mee te maken. In elk geval kan de multicolormode, in verbinding met sprites als achtergrond dienst doen. Laten we ons dan ook maar met de sprites bezighouden.

2.6 De sprites

De mogelijkheden sprites weer te geven op de MSX systemen kunnen bijzonder goed genoemd worden. Tot 32 sprites kunnen geprogrammeerd worden. Met behulp van sprites kunnen bewegingen op een uitstekende wijze op het beeldscherm gebracht worden.
In principe is een sprite slechts een overgedimensioneerd teken, dat via speciale instrukties ook op het grafisch beeldscherm getoond kan worden. Een grote sprite bestaat uit 16*16 punten. Analoog aan het teken wordt een sprite, door het inbrengen in een sprite-model tabel gedefinieerd. De positie van de sprite komt overeen met de coördinaten van de linker bovenhoek. Door het veranderen van de positiecoördinaten kan het spritemodel

over het gehele beeldscherm bewogen worden. Een weergave van rustig glijdende bewegingen, die overigens in hoog oplossende grafiek zeer tijdrovend werkt, kan nu eenvoudig geprogrammeerd worden.

Bij de MSX computers zijn hiermee de spritemogelijkheden nog niet uitgeput. Het aangeven van de sprites gebeurt op de zogenaamde spritevlakken. Op elk van de 32 achter elkaar liggende vlakken kan een sprite aangegeven worden. Het bijzondere aan deze handelwijze ligt in het 'achter elkaar liggen'. Een sprite, die op een verdergelegen vlak aangegeven wordt, wordt door zijn voorgaand vlak bedekt, als de modellen elkaar overlappen. Hierdoor kunnen de voorstellingen in pseudo-3D (zonder enig perspectief) aangemaakt worden. De voorgrond, zoals bijv. bewegende figuren, voertuigen e.d., wordt op de voorste vlakken (= lage getallen) getoond. Hiermee overlappen deze sprites alle andere, wat overeenkomt met de realiteit. Beelden, die in werkelijkheid verder van de kijker verwijderd zijn, worden op verder gelegen spritevlakken getoond. Op het achterste vlak (=31) worden bijv. vaak wolken e.d. weergegeven. Na het laatste spritevlak bevindt zich het eigenlijke beeldschermvlak, d.w.z. sprites bedekken in de letterlijke zin het standaardbeeldschermvlak.
Het gebruik van sprites is in de mode 1, 2 of 3 identiek. In mode 0 (tekstmode) kunnen sprites niet gebruikt worden. Het vlak, waarop de sprite getoond moet worden, wordt door de eerste parameter van de >PUT SPRITE< instruktie bepaald. Sprites kunnen in 4 verschillende modificaties gebruikt worden. Zoals reeds vermeld bestaat de mogelijkheid 16*16 punten grote sprites, maar ook 8*8 punten grote sprites te gebruiken. Onafhankelijk van het aantal punten kunnen sprites normaal (originele grootte) en vergroot (dubbele omvang) weergegeven worden. Bij vergrote uitvoer wordt niet het aantal punten verhoogd maar elke punt wordt dan vier keer zo groot getoond. De keuze van de weergave mode volgt uit de tweede parameter van de >SCREEN< instruktie., daarbij betekent:

parameter	punten	vergroting	reële grootte
0	8*8	normaal	8*8
1	8*8	vergroot	16*16
2	16*16	normaal	16*16
3	16*16	vergroot	32*32

De definitie van een 8*8 sprite hebben we in principe reeds in het hoofdstuk van de tekstmode leren kennen. Een sprite wordt als bitmatrix geschreven, waarbij elke horizontale lijn van elk 8 punten een byte (8 bits) voorstelt. De waarde van de bytes is gelijk aan de waarde van het binair getal, dat verkregen wordt als men voor alle punten een 1 schrijft en voor alle niet geplaatste punten een 0. Hierdoor wordt een 8*8 sprite (evenals bij een teken) door een reeks van 8 cijfers gekarakteriseerd. Bij tekendefiniëring moesten we deze getallen direkt in de VRAM poken. Bij de sprites bestaat hiervoor een BASIC instruktie >SPRITE $(. . .)= . . .<
Misschien verwondert het U dat het hier plotseling over een stringinstruktie gaat. Voor de interne verwerking zijn strings en tekensketens identiek, d.w.z. een string wordt als een tekensketen opgeslagen. De letter "A" = CHR$(65) wordt intern als een getal met de waarde 65 opgeslagen.
Het voordeel, de spritedefiniëring via strings door te geven, bestaat uit een aantal instrukties die ter beschikking staan en daarna met >SPRITE< de spritemodellen te manipuleren. Bekijk maar eens de volgende sprite

```
1 ********      &HFF
2 *** ***       &HEE
3 ******        &HFC
4 * ***         &HB8
5 ******        &HFC
6 *** ****      &HEF
7 **   **       &HC6
8 *    * *      &H85
  12345678
```

Deze sprite moet als een model gedefinieerd worden:

1e mogelijkheid:

```
10 SPRITE$(1)=CHR$(&HFF)+CHR$(&HEE)+CHR$(&HFC)+CHR$(&HB8)+CHR$(&
HFC)+CHR$(&HEF)+CHR$(&HC6)+CHR$(&H85)
```

2e mogelijkheid:

```
10 FORI=0TO7:READB$:A$=A$+CHR$(VAL("&H"+B$)):NEXT
20 SPRITE$(1)=A$
30 DATA FF,EE,FC,B8,FC,EF,C6,85
```

De tweede mogelijkheid, die in eerste aanleg omvangrijker lijkt, heeft als voordeel dat de sprite definitie in A$ opgeslagen is, dus steeds ter beschikking staat en evt. later gewijzigd kan worden.
Bij 8*8 sprites kunnen tot 256 (!!) (de nummers 0-255) verschillende spritemodellen gedefinieerd worden. Van deze 256 kunnen telkens 32 willekeurige sprites gelijktijdig getoond worden.
Bekijk ook eens de mogelijkheid die, analoog aan de eerste mogelijkheid, als

SPRITE$(1)="1AXz1,Bz" ook funktioneert.

De 16*16 sprites worden in principe op dezelfde manier gedefinieerd, daartoe wordt de 16*16 grafiek in vier 8*8 matrices verdeeld, die dan na elkaar uitgegeven worden. De volgorde is dan

I	III
II	IV

In totaal kunnen 64 verschillende 16*16 modellen gedefinieerd worden. Om het ontwerpen van sprites niet al te moeizaam via het papier te laten verlopen, stellen we U een sprite editor voor.

Programma: sprite-editor

De funktie van de sprite editor is gelijk aan die van de tekeneditor. Het wezenlijke verschil bestaat in de onderscheiding van 8*8 en 16*16 punten, daarna tussen normaal of vergroot. Daarna kunt U vrij, zoals gebruikelijk, met behulp van de joystick (cursortoetsen) binnen de matrix bewegen en door het indrukken van de vuurknop (spatiebalk) punten plaatsen, resp. herstellen.
Naast de definitiematrix ziet U de bij dat model behorende waarden aangegeven. Die dient U dan in volgorde, kolomsgewijs van boven naar beneden voor de definitie van sprites in Uw programma's te gebruiken. Daarbij stelt de linker kolom de waarde voor het eerste en tweede kwadrant voor, de rechter kolom de waarde voor het derde en vierde kwadrant in hexadecimale vorm. (volgens het hierboven gegeven spritemodel)
Deze waarden dient U ofwel in >CHR$< vorm (volgens de reeds eerder gegeven mogelijkheid 1) of in >DATA< regels (mogelijkheid 2: op dezelfde plaats) aan te geven. Indien U niet met hexadecimalen vertrouwd bent,

bekijk dan eerst het hoofdstuk van de machinetaal. Hier vindt U onder het opschrift getalsystemen de vereiste informatie. De sprite ziet U boven de editor-matrix in originele grootte weergegeven.

Hier volgt nu de funktietoetsenreservering:

KEY 1: Norm (normaalmode)
Als U deze funktietoets indrukt, kunt U met de joystick afzonderlijke punten in de gekozen matrix door het indrukken van de vuurknop wissen of plaatsen.

KEY 2: Lion (Line on mode)
Door het indrukken van de tweede funktietoets schakelt U de line on mode in. In deze mode kunt U door het indrukken van de vuurknop binnen de spritematrix lijnen tekenen. U bent dan niet meer gedwongen punten te plaatsen, maar gesloten lijnen te tekenen.

KEY 3: Liof (line off mode)
Deze toets veroorzaakt het inschakelen van de line off mode. Deze funktie is tegengesteld aan de lion mode, d.w.z. zij wist de gesloten lijnen.

KEY 4: PRNT (printer mode)
Als U deze toets indrukt, wordt de door U getekende sprite, met zijn waarden (&H) uitgevoerd naar de printer.

KEY 5 Einde
Deze toets beëindigt het programma.

```
10 DEFINT A-Z
20 SCREEN 2
30 A$="J"
40 INPUT"8x8 Sprite (j/n) ";A$
50 IF (ASC(A$)AND&B11011111)=ASC("J") THEN M=0 ELSE M=2
70 A$="N":INPUT"vergroten (j/n)";A$
80 IF (ASC(A$)AND&B11011111)=ASC("j") THEN M=M+1
90 SCREEN 1,M:COLOR 15,4,4
100 MK=INT(M/2)
```

```
110 BA=BASE(9)
120 XA=1:YA=6:QZ=1
130 EI=1
140 ON STRIG GOSUB 530,530,530,530,530
150 STRIG(EI) ON
160 AU$=" ":EI$=CHR$(42)
170 ON KEY GOSUB 700,620,660,850,740
180 KEY 1,"Norm":KEY (1) ON
190 KEY 2,"Lion":KEY (2) ON
200 KEY 3,"Liof":KEY (3) ON
210 KEY 4,"Prnt":KEY (4) ON
220 KEY 5,"Einde":KEY (5) ON
230 CLS:PUT SPRITE 0,(24,0)
240 FOR QZ=1 TO 1+3*MK
250 FOR Y=0 TO 7
260 GOSUB 580
270 NEXT Y,QZ
280 GOSUB 700
290 X=0:Y=0:QZ=1
300 LOCATE XA-8*(QZ>2)+X,YA-8*(QZ=2 OR QZ=4)+Y,1
310 R=STICK(EI)
320 IF R=0 THEN 310
330 STRIG(EI) OFF
340 IF R=8 OR R=1 OR R=2 THEN Y=Y-1
350 IF R>3 AND R<7 THEN Y=Y+1
360 IF R>1 AND R<5 THEN X=X+1
370 IF R>5 AND R<=8 THEN X=X-1
380 IF X<8 THEN 410
390 IF M<2 THEN X=7:BEEP:GOTO 410
400 IF QZ=1 OR QZ=2 THEN X=0:QZ=QZ+2 ELSE X=7:BEEP
410 IF X>=0 THEN 440
420 IF M<2 THEN X=0:BEEP:GOTO 440
430 IF QZ=3 OR QZ=4 THEN X=7:QZ=QZ-2 ELSE X=0:BEEP
440 IF Y<8 THEN 470
450 IF M<2 THEN Y=7:BEEP:GOTO 470
460 IF QZ=1 OR QZ=3 THEN Y=0:QZ=QZ+1 ELSE Y=7:BEEP
```

```
470 IF Y>=0 THEN 500
480 IF M<2 THEN Y=0:BEEP:GOTO 500
490 IF QZ=2 OR QZ=4 THEN Y=7:QZ=QZ1 ELSE Y=0:BEEP
500 IF LF=0 THEN STRIG(EI) ON:GOTO 300
510 IF LF=-1 THEN BY=VPEEK(BA+(QZ-1)*8+Y) OR 2^(7-X):GOS
UB 560:GOTO 300
520 BY=VPEEK(BA+(QZ-1)*8+Y) AND (NOT 2^(7-X)):GOSUB 560:GOTO 300
530 REM Strig
540 BY=VPEEK(BA+(QZ-1)*8+Y) XOR2^(7-X)
550 LOCATE ,,0
560 IF (BY AND 2^(7-X))=0 THEN PRINT AU$; ELSE PRINT EI$;
570 VPOKE BA+(QZ-1)*8+Y,BY
580 LOCATE XA+8*(1+MK)+3-3*(QZ>2),YA+Y-8*(QZ=2 OR QZ=4)
590 PRINTRIGHT$("00"+HEX$(BY),2);
600 LOCATE XA-8*(QZ>2)+X,YA-8*(QZ=2 OR QZ=4)+Y,1
610 RETURN
620 REM Lion
630 LF=-1
640 A$="lijn tekenen":GOSUB 800
650 RETURN
660 REM liof
670 LF=1
680 A$="lijn wissen":GOSUB 800
670 RETURN
700 REM normaal
710 LF=0
720 A$="normaal ":GOSUB 800
730 RETURN
740 REM End
750 LOCATE ,,0
760 DEFUSR1=&H139D
770 X=USR1 (1)
780 CLS
790 END
800 XP=POS(0):YP=CSRLIN
```

```
810 LOCATE 10,1,0
820 PRINTA$;
830 LOCATE XP,YP,1
840 RETURN
850 REM Print
860 LOCATE ,,0:TB=BASE(5)
870 FOR Z=YA TO YA-1+8*(1+MK)
880 FOR S=0 TO 31
890 LPRINT CHR$ (VPEEK(TB+32*Z+S));
900 NEXT S
910 LPRINT:NEXT Z
920 LOCATE ,,1:RETURN
```

Programmabeschrijving:

De principiële opbouw van de sprite-editor is gelijk aan de tekengenerator. Het wezenlijke verschil bestaat in de onderscheiding 8*8 en 16*16 spritegrootte en in de toegevoegde funkties die het gebruik veraangenamen.

De hierna volgende variabelen maken de verschillen duidelijk. M bevat de sprite-aanwijsmode (0 tot 3) en MK=0 bij 8*8 en MK=1 bij 16*16 grote spritematrix.
Belangrijk is de variabele QZ. Zij bevat het nummer van het 8*8 gedeelte bij 16*16 spritegrootte.
Met behulp van QZ, x en y kan altijd de reële beeldschermpositie verkregen worden.

Gedeeltelijk ontstaan relatief gecompliceerde structuren, zoals $>\ldots+8*(qz>2)\ldots<$. Indien het gebruik van dergelijke arithmetische uitdrukkingen onbekend is, probeert U dan eens het volgende uit:

PRINT 1=3 of PRINT 4>2

Een ware stelling levert dan de waarde –1 op, een foutieve stelling een 0.

Voortbouwend op deze truc funktioneert het verschil in weergave van 8*8 en 16*16 sprites. Maakt U zich vertrouwd met deze programmeertechniek. Zij kan U zeer nuttig zijn en programma's aanzienlijk bekorten.
Voor het onderscheid tussen de verschillende verwerkingswijzen wordt de

variabele LF gebruikt. 0 betekent normaal, 1 regel wissen en –1 regel plaatsen.
In alle verwerkingswijzen, behalve 'normaal', is de STRIG interrupt uitgeschakeld, omdat die daar niet nodig is.

Hoe bewegen we nu een sprite?

De positie van de sprite wordt bepaald door de coördinaten van de linker bovenhoek. Deze coördinaten worden met >PUT SPRITE< vastgelegd. Het complete format van de instruktie is:

PUT SPRITE vlak (X-coördinaten, '-coördinaten) kleur, modelnummer.

Het modelnummer is het toegevoegde getal in

>SPRITE$(modelnummer)= . . .<

Het aanmaken van gehele beelden met sprites is nu mogelijk. Een demoprogramma hiervoor zou, op grond van de vele spritedefinities, te omvangrijk worden. Let daarom bij Uw programma's op het navolgende.

TIPS:

- Beeldelementen die groter dan 16*16 punten moeten zijn, kunnen uit meerdere, naast elkaar getoonde, sprites opgebouwd worden. Bij het bewegen van een reuzesprite dienen dan ook meerdere >PUT SPRITE< instrukties na elkaar uitgevoerd te worden.

- Meerkleurige sprites zijn principieel door overlapping van verschillende kleuren te maken. Ook daarvoor moeten dan, bijv. voor het bewegen van een driekleurige sprite (dus voor drie op elkaar liggende sprites), ook drie gelijksoortige >PUT SPRITE< instrukties uitgevoerd worden. De coördinaten kunnen, indien de instrukties achter elkaar staan, direkt bij de eerste instruktie aangegeven worden. Komen bovendien modelnummer en vlaknummer overeen, dan kàn eenvoudigheidshalve volstaan worden met >PUT SPRITE n< in te geven, waarbij n het model- resp. het vlaknummer is.

- Op één horizontale lijn kunnen maximaal 4 verschillende sprites getoond worden. Bij 5 of meer worden telkens de 4 laagste vlaknummers uitgevoerd.

- Een interessante mogelijkheid van sprites is de 'filmsimulatie'. Bij 64 verschillende 16*16 sprites kunnen bij handige programmering een groot aantal bewegingen uitstekend weergegeven worden.

Hiervoor hebben we een programma samengesteld dat een lopende 'olifantenkudde' voorstelt. Er wordt niet alleen een model bewogen, maar door de opeenvolging van modelschakelingen de indruk gewekt van het lopen. (principe van de tekenfilm)

Programma: kudde lopende olifanten

Spritedefinitie (16*16)

Sprite 1

```
                    1234567812345678      = 8 +8=16

        &H00  1 0000000000000000          &H00
        &H00  2 0000110000000100          &H04
        &H12  3 0001001000000010          &H02
A$(1)   &H13  4 0001001111110001          &HF1   A$(3)
        &H1A  5 0001101000001101          &H0D
        &H3E  6 0011111000000011          &H03
        &H40  7 0100000010000001          &H81
        &H89  8 1000100110000001          &H81
        &H56  1 0101011010000001          &H81
        &H90  2 1001000010000001          &H81
        &H57  3 0101011100001001          &H09
A$(2)   &H94  4 1001010001001011          &H4B   A$(4)
        &H52  5 0101001010111010          &HBA
        &HF2  6 1111001010001010          &H8A
        &H02  7 0000001010001010          &H8A
        &H07  8 0000011110011110          &H9E
```

Sprite 2

			1234567812345678	= 8 +8=16	
	&H00	1	0000000000000000	&H00	
	&H0C	2	0000110000000100	&H04	
	&H12	3	0001001000000010	&H02	
B$(1)	&H13	4	0001001111110001	&HF1	B$(3)
	&H1A	5	0001101000001101	&H0D	
	&H3E	6	0011111000000011	&H03	
	&H40	7	0100000010000001	&H81	
	&H89	8	1000100110000001	&H81	
	&H56	1	0101011010000001	&H81	
	&H90	2	1001000010000001	&H81	
	&H57	3	0101011100001001	&H09	
B$(2)	&H94	4	1001010001001010	&H4A	B$(4)
	&H52	5	0101001010111010	&HBA	
	&HF2	6	1111001010001010	&H8A	
	&H02	7	0000010100010100	&H14	
	&H0F	8	0000111100111100	&H3C	

Sprite 3

			1234567812345678	= 8 +8=16	
	&H00	1	0000000000000100	&H04	
	&H0C	2	0000110000000010	&H02	
	&H12	3	0001001011100010	&HE2	
C$(1)	&H17	4	0001011100010001	&H11	C$(3)
	&H1A	5	0001101000001101	&H0D	
	&H1E	6	0001111000000011	&H03	
	&H20	7	0010000000000001	&H01	
	&H49	8	0100100110000001	&H81	
	&H56	1	0101011010000001	&H81	
	&H91	2	1001000100000001	&H01	
	&H96	3	1001011000001010	&H0A	
C$(2)	&HA4	4	1010010001001010	&H4A	C$(4)
	&HA2	5	1010001010111010	&HBA	
	&HF2	6	1111001010001001	&H89	
	&H01	7	0000000101000101	&H45	
	&H03	8	0000001111001111	&HCF	

Programma:

```
10 DEFINT A-Z
20 SCREEN 2,2
30 FOR I=1 TO 4:FOR J=0 TO 7:READ A$:A=VAL("&H+A$)
40 A$(I)=A$(I)+CHR$(A):NEXT J,I
50 DATA 00,0C,12,13,1A,3E,40,89
60 DATA 56,90,57,94,52,F2,05,0F
70 DATA 00,04,02,F1,0D,03,81,81
80 DATA 81,81,09,4A,BA,8A,14,3C
90 SPRITE$(1)=A$(1)+A$(2)+A$(3)+A$(4)
100 B$(1)=A$(1)
110 B$(2)=LEFT$(A$(2),6)+CHR$(&H2)+CHR$(&H7)
120 B$(3)=A$(3)
130 B$(4)=A$(4):MID$(B$(4),4,1)=CHR$(&H4B):B$(4)=LEFT$(B$(4),
6)+CHR$(&H8A)+CHR$(&H9E)
140 SPRITE$(2)=B$(1)+B$(2)+B$(3)+B$(4)
150 SPRITE$(0)=B$(1)+B$(2)+B$(3)+B$(4)
160 FOR J=0 TO 31:READ A$:A=VAL("&H"+A$)
170 C$=C$+CHR$(A):NEXT J
180 DATA 00,0C,12,17,1A,1E,20,49
190 DATA 56,91,96,A4,A2,F2,01,03
200 DATA 04,02,E2,11,0D,03,01,81
210 DATA 81,01,0A,4A,BA,89,45,CF
220 SPRITE$(3)=C$
230 '
240 'Beweging Sprite
250 '
260 X=256:Y=20
270 FOR N=0 TO 3:X=X1
280 FOR Z=1 TO 15 STEP 2
290 PUT SPRITE Z,(((X+20*Z)MOD 287)-31,Y+(Z MOD
4)*30-2*(N=2)),Z,N
300 NEXT Z,N
310 IF X>-31 THEN 270
320 GOTO 260
```

Programmabeschrijving:

regel 30
Lus voor het uitlezen van de DATA regels in de volgorde A$(1) tot A$(4). Lezen van de DATA regels met automatische toevoeging"&H"

Regel 40
Samenvoegen van in de DATA regels aangegeven waarden met A$(1) tot A$(4).

Regel 50 tot 80
DATA regels sprite 1

Regel 90
Sprite$(1) opbouwen.

Regel 100 tot 130
sprite 2 en 0 uit delen van sprite 1 opbouwen.

Regel 140
Sprite 2 uit de gelezen string definiëren.

Regel 150
Sprite 0 definiëren

Regel 160
Lus voor het uitlezen van sprite 3.
Lezen van DATA regels met automatische toevoeging "&H"

Regel 170
Samenvoegen van in DATA regels aangegeven waarden met C$(1) tot C$(4)

Regels 180 tot 210
DATA regels voor sprite 3

Regel 220
Met sprite$(3)=C$ sprite 3 definiëren

Regel 260
X- en Y-coördinaten bepalen voor positie op beeldscherm

Regel 270
Volgorde bepalen van de na elkaar volgende sprites in een lus. X bepaalt de bewegingsrichting op het beeldscherm.

Regel 280
8 Sprites op het beeldscherm in verschillende kleuren na elkaar weergeven

Regel 290
Roept de sprites op. Bepaalt met behulp van de in regel 270 en 280 aangegeven lussen de bewegingsrichting, aantal en volgorde van

optreden van de sprites.

Regel 310

Als de bewegingsrichtingteller X groter is dan –31, doorgaan met regel 270, d.w.z. indien de linker beeldschermrand wordt overschreden.

Regel 320

Zo niet dan doorgaan met regel 260

Sprites intern

Ter afsluiting behandelen we nog in het kort de interne verwerking van de sprites in de VRAM en de daarmee samenhangende VDP registers.

Evenals de teken- en modeltabel, bestaat er ook een spritemodeltabel. Zij is 2048 (=2KB) bytes lang en kan in stappen van 2 KB verschoven worden. De mogelijkheid om tussen meerdere spritetabellen met interrupts om te schakelen is aanwezig. Zie hiervoor onderdeel grafiek mode I
De spritemodeltabel is in 256 blokken van elk 8 bytes (een 8*8 model) onderverdeeld. Bij 8*8 sprites is het modelnummer gelijk aan het in de tabel staande 8 blok dat de definities bevat. Bij 16*16 sprites moet het modelnummer met 4 worden vermenigvuldigd om het bloknummer van de definitie te verkrijgen.

In principe kunt U sprites dus ook met >VPOKE< definiëren. Het startadres wordt door het VDP register 6 bepaald. Dit register bevat de 3 MSB's van het adres van de spritemodeltabel.

Het startadres kan ook met >BASE(PEEKC(&HFCAF)*5+4)< voor alle modi verkregen worden. Bij het inschakelen van de computer bezet de spritemodeltabel in alle mogelijke modi de adressen &H3800 tot &H3FFF. Hieraan is de waarde van VDP register 6 gelijk. De tweede tabel is de zg. sprite attributen tabel (SAT). Zij bevat de informatie die door >PUT SPRITE< vastgelegd werd. Omdat er 32 vlakken bestaan is de SAT in 32 blokken onderverdeeld. Elk blok is 4 bytes lang. De bloknummers (0-31) komen overeen met het onderhavige vlaknummer, d.w.z. invoer in blok 5 geldt voor vlak 5.
De SAT is aldus in 128 stappen te verplaatsen. VDP register 5 bevat de 7 bovenste bits van het 14-bit adres van de tabel. Met >BASE< kan het SAT startadres in alle modi met >BASE(PEEK(&HFCAF)*5+3) verkregen worden.

Bij het inschakelen vindt men de waarden &H1B00 tot &H1BFF, VDP register 5 bevat &H1B00/2^(14–7)=&H36.
De invoer in een blok van 4 bytes heeft de volgende betekenis:

Byte 0 - Y coördinaten
Byte 1 - X coördinaten
Byte 2 - modelnummer
Byte 3 - o.a. kleur

De coördinaten hebben principieel betrekking op de linker bovenhoek van de sprite.
Ook negatieve coördinaten zijn mogelijk, (weergave in twee-komplement: 256-getal) hetgeen betekent dat het bovendeel van de sprite niet getoond wordt. Het is zogezegd achter het raam verdwenen. Daardoor is het ook mogelijk sprites langzaam te laten opkomen. Bij >PUT SPRITE< kunnen achtereenvolgens # coördinaten tussen –31 en +192 aangegeven worden. Met de X coördinaten is dit niet zo eenvoudig mogelijk, omdat alle 256 mogelijke waarden van een byte als normale coördinaat gebruikt worden. Om dit toch mogelijk te maken wordt bit > van de 4e byte van een blok gebruikt. Het heet het EC (early clock) bit. Indien 0 dan bestaat de normale toestand. Wordt de waarde 1 toegekend, dan zijn alle X posities van de sprite 32 punten naar links verschoven. Dan kan met X coördinaten tussen 0 en 32 bereikt worden dat de sprite van links het beeldscherm binnenkomt. Bij de >PUT SPRITE< instruktie kunnen direkt waarden tussen –32 en +255 ingegeven worden.
Het 3e bit bevat een wijzer in de spritemodeltabel. Bij 8*8 sprites is die gelijk aan het spritenummer van het BASIC. De bits 0-3 van het 4e byte bevatten tenslotte de kleur van de sprite. Om de sprite aanwijsmode te bepalen zijn 2 registers van het VDP voorzien:

Bit 0 MAG - magnification - vergroting
Bit 1 SIZE - size - grootte

	0	1
MAG	normaal	vergroot
SIZE	8*8	16*16

Komen meer dan 4 sprites in een horizontale lijn voor, dan wordt in VDP register 8, dat gewoonlijk als statusregister wordt betiteld (read only), het

bit 6 geplaatst. In de bits 0 tot 4 wordt dan het nummer van de 5e sprite opgeslagen, die op het eerste, niet meer getoonde, vlak ligt.

Uiteindelijk is er ook nog de zogenaamde coïncidentie flag, die aangeeft of tenminste 2 sprites elkaar overlappen. De C vlag (coïncidentievlag) is het 5e bit van het statusregister. Via dit bit is de BASIC interrupt funktie "ON SPRITE" mogelijk. Hiermee kunnen spritebotsingen direkt verwerkt worden.

3. I/O

3.1 Algemene beschouwing bij I/O

Wat zou een computer zijn zonder periferie, dus zonder beeldscherm, toetsenbord, diskdrive, printer enz.?

De mogelijkheden van een computer zijn pas bruikbaar als bijpassende periferie, speciaal toetsenbord en beeldscherm, aanwezig en aansluitbaar is. Een computer heeft daartoe dan ook verschillende externe aansluitpunten. Bij het MSX systeem zijn dat o.a.

Monitorbus
RGB monitor
Printer-interface
Joystickpoorten
Cartridge slots

Eerst zullen we de MSX periferie in het kort weergeven.

De cassetterecorder

Over de principiële bediening van de cassetterecorder is niet veel te vertellen.
De standaard instrukties >CSAVE< en >CLOAD< dienen voor het opslaan/laden van BASIC programma's.
De instrukties >BLOAD< en >BSAVE< zijn ontwikkeld voor het opslaan van geheugeninhouden. Dit komt meestal bij het gebruik van programma's in machinetaal voor.
Interessant zijn de instrukties >SAVE< en >LOAD<.
Het zijn algemene instrukties, die voor het overdragen van gegevens, met behulp van bestanden, dienen. Het aantal gelijktijdig te gebruiken bestanden wordt vastgelegd door >MAXFILES<. Door deze instruktie worden gebieden in de RAM voor het beheer van deze bestanden vastgelegd.
Het bijzondere aan >LOAD< en >SAVE< is, dat zij niet alleen betrekking hebben op de cassetterecorder. In theorie kan elk periferie-apparaat als doel- resp. bronadres voor over te dragen of te ontvangen gegevens dienen. Daardoor is het mogelijk in het argument van de instruktie het uitvoerapparaat aan te geven. Diverse instrukties voor het behandelen van bestanden (>OPEN<, >PRINT<, >CLOSE<) kunnen op de hierbo-

ven beschreven manier gebruikt worden.
Door het definiëren van een ander doel wordt dan de gehele gegevensstroom na ar het aangegeven apparaat geleid. Aan de volgende periferie kan in deze vorm uitgevoerd worden:

CAS - Cassetterekorder
CRT - Beeldscherm
GRP - Grafiek-beeldscherm
LPT - Printer

De techniek van gegevensuitvoer via >PRINT#< hebben we reeds vaker in de grafiekprogramma's van vorige hoofdstukken gebruikt.

Ofschoon periferie een belangrijk thema is, wordt de behandeling ervan vaak verwaarloosd. De I/O programmering is gebonden aan de hardware, en daardoor moeilijk toegankelijk. Veel ideeën zijn slechts met behulp van machinetaal te verwezenlijken. Hierna behandelen we de basisprincipes van de I/O programmering met betrekking tot de MSX computers.

Beeldschermconfiguraties
Het signaal, dat via de normale antennestekker opgeroepen wordt, wordt samengesteld uit de informatie, verkregen via de ingebouwde modulator van de VDP. Het soundsignaal van PSG (Programmable Sound Generator) is hierin opgenomen.

De monitoruitgang gebruikt dezelfde basisinformatie, alleen wordt het signaal niet gemoduleerd, d.w.z. het kan alleen door een monitor en niet door een TV ontvanger verwerkt worden. De RGB uitgang levert een signaal voor hoogoplossende kleurenmonitors en wordt weer uit dezelfde basisinformatie opgebouwd.
Vermeldenswaardig is dat de RGB uitgang tevens een stereo-signaal bevat. Het rechter en linker kanaal zijn afzonderlijk op te roepen. Omdat hier met een voudige middelen niets te programmeren of te veranderen is, gaan we maar over naar het tweede belangrijke periferieapparaat:

Toetsenbord
Elektronisch gezien is het toetsenbord een rechthoekig net van gekruiste leidingen. Op elk knooppunt bevindt zich een toets. In deze samenhang wordt gesproken van een toetsenbordmatrix. Het toetsenbord wordt 50 keer per seconde door de interne interrupt afgecheckt. Daartoe worden na

elkaar door een horizontale leiding stroomstoten gevoerd (bit=1). Op de verticaal lopende leidingen wordt dan getest op spanning. Heeft een van deze leidingen spanning (dus bit=1) dan is de op dit knooppunt liggende toets ingedrukt. Dat betekent dat de stroomkring hier gesloten is, waardoor bepaald kan worden welke toets werd ingedrukt.

Hoe moet deze handeling geprogrammeerd worden?
Eerst is een uitvoerinstruktie nodig (OUT) die een horizontale leiding van stroom voorziet, direkt daarna een invoerinstruktie (INP) die signaleert welke toets ingedrukt werd. De IC die het toetsenbord onderzoekt is de PPI (Programmable Peripheral Interface). De MSX standaard schrijft de PPI 8255 van de firma INTEL voor.
De uitvoerpoort, die de stroom in het toetsenbord stuurt, is poort C (van de PPI). Om de verschillende uitgangen te onderscheiden is aan elke I/O poort een nummer toegewezen.

Poort C van de PPI heeft het nummer &HAA.
De onderste 4 bits van poort C van de PPI bepalen de horizontale leiding, waardoor de stroom gezonden wordt. De bovenste 4 moeten onaangeroerd blijven. Het begin van het programma ziet er dan ook als volgt uit:

```
10 CLS
20 Z=INP (&HAA): REM poort C lezen
30 Z=Z1 AND &HF0: REM bovenste 4 bits bewaren, onderste 4
wissen.
40 LOCATE 0,0
```

Nu volgt het onderzoek naar de 9 mogelijke leidingen:

```
50 FOR I = 0 TO 8
60 OUT &HAA, Z+I: REM leiding I onderzoeken.
```

Nu wordt de waarde van de verticale leiding met >INP< gelezen. De van het toetsenbord komende leidingen vormen poort B van de PPI. Ze heeft als nummer &HA9.

```
70 PRINT RIGHT $ ("00000000"+BIN$ (INP(&HA9)),8)
80 NEXT I
90 GOTO 20
```

Als U dit programma runt ontvangt U het beeld van de toetsenbordmatrix op het beeldscherm. Een 0 toont telkens een ingedrukte toets aan. Nu de regels 45 en 46 nog tussenvoegen

```
45 a$=INKEY$: IF A$="" THEN PRINT " " ELSE PRINT A$
46 LOCATE 0,1
```

en U kunt de toetsenbordmatrix vullen. U krijgt dan het volgende beeld:

7	6	5	4	3	2	1	0
8	9	-	=	U			;
'		,	.	/		A	B
C	D	E	I	G	H	I	J
K	L	M	N	O	P	Q	R
S	T	U	V	W	X	Y	Z
shift	ctrl	graph	cap	code	F1	F2	F3
F4	F5	esc	tab	stop	bs	select	return
space	home	ins	del				

Houdt U er rekening mee dat de met >INKEY$< onderzochte toets alleen maar met een enkelvoudige indruk van de toets in de matrix voorkomt. Dat betekent dat speciale tekens met SHIFT, CTRL, GRAPH e.d. door de onderzoekroutine apart herkend en geïnterpreteerd worden. Als we ons eigen toetsenbordonderzoek programmeren, dan moeten we op de hierboven uitgelezen gegevens terugvallen. Omdat, zoals reeds vermeld, dit onderzoek ook intern door de interrupt uitgevoerd wordt, kunnen we op de gegevens terugvallen, die in de zgn. NEWKEY-buffer opgeslagen zijn. Ons programma wordt daardoor vereenvoudigd tot:

```
10 CLS
20 LOCATE 0,0
30 FOR I=&HFBE5 TO &HFBED
40 PRINT RIGHT$("0000000"+BIN$(PEEK(I)),8)
50 NEXT I:GOTO 20
```

Hoe kunnen we nu deze bitmatrix gebruiken om alle ingedrukte toetsen te verkrijgen.
Het programma dat het toetsenbord op deze wijze onderzoekt, zullen we in het volgende hoofdstuk gebruiken om een polyfone (d.w.z. meerstemmige) klank te programmeren.

Binnen de ROM zijn alle tabellen, benodigd voor de toetsen-decodering, opgenomen. Bekijken we de tabel van het toetsenbord zonder >SHIFT< of andere speciale toetsen, dan vinden we die vanaf adres &H0DA5. Van dit adres af vinden we de ASCII-kodes van de tekens terug, die door het simpel indrukken van een toets getoond worden:

```
FORI=0 TO 5: FOR J =7 TO 0 STEP-1:
PRINTCHR$(PEEK(&HDA5+I*8+J));:NEXT:PRINT:NEXT
```

Deze tabellen worden intern gebruikt, om de, bij de ingedrukte toets behorende, ASCII-kode te vinden. Deze ASCII-kodes worden dan in de zg. KEYBUFfer afgelegd en staan daar ter beschikking.

Het volgende kleine programma leest de keybuffer en maakt hem in orde voor de volgende nieuwe invoer.

```
10 KB=VAL("&HFBF0"):REM Startadres KEYBUF
20 GOTO 50: KEYBUF Initialisatie
30  A=PEEK(&HF3F8)-&HF0:IF A=0 THEN 30:REM aantal ingedrukte
toetsen
40  FOR I=KB TO KB+A-1:PRINT CHR$(PEEK(I)):NEXT:REM
Bufferinhoud op scherm zetten
50 POKE &HF3F8,&HF0:POKE &HF3F9,&HFB:REM Bufferinitialisatie
60 POKE &HF3FA,&HF0:POKE &HF3FB,&HFB
70 POKE &HF3F7,1:REM Wachttijd uitschakelen bij repeteerfunctie
80 PRINT:GOTO 10
```

Interessant is de >POKE< instruktie van regel 70:

Wordt deze regel weggelaten, dan zal het ingedrukt houden van de toets niet geregistreerd worden. Normaal wordt een wachttijd in acht genomen totdat de repeteerfunktie in werking treedt. De wachttijdteller (REPCNT-repeat counter) is de waarde op adres &HF3F7.
Wordt deze waarde op 1 gezet, dan is herhaling met minimale wachttijd mogelijk.
Tot zover dan het I/O-toetsenbord.

Verder naar boven hebben we de 4 bovenste bits van poort C nog niet bekeken. Nemen we eerst bit 6

Bit 6 van poort C van de PPI geeft de toestand van de capdiode weer. De betekenis hiervan is:

BIT 6=1 De capdiode geeft geen licht
BIT 6=0 De capdiode licht op

Probeert U eens:
De capdiode uitschakelen en dan:

```
PRINT HEX$(INP(&HAA))
```

De capdiode inschakelen en dan

```
PRINT HEX$(INP(&HAA))
```

Het uitschakelen van de capdiode wordt met

```
OUT &HAA, INP(&HAA) OR 216
```

of eenvoudig met

```
OUT &HAB,13
```

bereikt.

De tweede versie is een stuurinstruktie via de stuurpoort &HAB. De instruktie luidt:

'Zet bit 6 van de CAP diode op 1'

Het inschakelen van de cap-diode wordt bereikt met:

```
OUT &HAA,INP(&HAA) AND NOT "^6
```

of door de stuurkode

```
OUT &HAD,12
```

wat zoveel betekent als 'zet bit 6 van poort C op 0'.

Let er daarbij op dat dit alleen betrekking heeft op de lichtdiode. De overschakeling van kleine- naar hoofdletters verandert hierdoor niet. Zou deze omschakeling vanuit een programma moeten gebeuren dan dient U als volgt te handelen:

1) Simulatie 'CAP-toets indrukken' via programma.

```
DEF USR1=&HF36:X=USR1(0)
```

Deze machineprogramma-oproep verandert bij elke oproep de status van CAP, d.w.z. het komt overeen met het indrukken van de cap-toets.

2) Test op kleine- of hoofdletters.

```
F=PEEK(&HFCAB):IF F=0 THEN PRINT "kleine letter" ELSE
PRINT "hoofdletter"
```

3) Systematisch inschakelen van hoofdletters en cap-diode:

```
POKE &HFCAB, 0:OUT &HAB,13
```

In samenhang met de captoets is bijv. ook een wachtlus via de >WAIT< instruktie mogelijk. De >WAIT< instruktie voert, praktisch gesproken, een >INP< instruktie uit en vergelijkt de ontvangen waarde met de aangegeven waarde. Er wordt een >XOR< met de tweede waarde en dan een >AND< met de eerstaangegeven waarde uitgevoerd. Is het resultaat 0 dan wordt verder gewacht, anders wordt het programma voortgezet.
Spreken we nu af dat een brandende capdiode de wachttoestand aangeeft met

```
WAIT &HAA,2^6
```

dan wordt het programma zolang gestopt totdat de cap-toets wordt ingedrukt, wat de lichtende diode uitschakelt.

Moet daarentegen gewacht worden, tot de diode ingeschakeld wordt, dan dient het betreffende bit met XOR geïnverteerd worden.

```
WAIT &HAA,2^6,2^6
```

Tot zover de captoets.

Bezien we nu de funktie van bit 7 van poort C.

U kent zeker het zogenaamde toetsenbordgeruis (klikken). Door de >SCREEN< of

```
POKE&HFF3DB,0
```

kan dit geruis uitgeschakeld worden.
Dit klikken wordt, ofschoon het hier over een toon gaat, niet door de PSG (sound chip) voortgebracht. De mogelijkheid bestaat om via de PPI een klikken via de audio-uitgang te veroorzaken. Dit gebeurt eenvoudig via het plaatsen en terugplaatsen van een leiding die met deze uitgang verbonden

is. De overgang van de ene naar de andere toestand zorgt voor een bewegen van het luidsprekermembraam, waardoor een klikken hoorbaar wordt. Interessant is de mogelijkheid om niet alleen dit klikken te veroorzaken, maar door een snelle opeenvolging van plaatsen en terugplaatsen van de leiding de luidspreker zodanig te doen trillen dat een toon wordt voortgebracht.

Schakelt U daarvoor het toetsenbordgeruis met >SCREEN0< of met >POKE&HF3DB,0< uit en geef het navolgende in:

```
OUT &HAB,15
```

Ofschoon het klikgeluid uitgeschakeld is, hoort U bij het indrukken van de returntoets een klikken. Dit geluid toont aan dat de luidsprekermembraam gespannen staat. Bij het nogmaals uitvoeren van deze instruktie hoort U niets, omdat de toestand van de membraam niet wijzigt. Met

```
OUT &HAB,14
```

wordt de spanning van de luidspreker weer op 0 geplaatst. Omdat de membraam door de >OUT &HAB,15< instruktie gespannen stond hoort U nu weer een klik.
De aktuele stand van deze zogenaamde software sound leiding verkrijgen we door een >INP< instruktie. De toestand komt overeen met de inhoud van bit 7 van poort C van de PPI. Deze poort heeft als adres &HAA. Om uitsluitend bit 7 te bekijken, gebruiken we de logische instruktie

```
PRINT INP (&HAA) AND 2^7
```

Krijgt U na de bovenstaande invoer een 0 dan was de leiding 0, wordt daarentegen 128 weergegeven dan was de leiding in toestand 1.

Omdat een toon door trillingen klinkt, zoals een vioolsnaar of luidsprekermembraam, veroorzaakt een snelle opeenvolging van plaatsen en terugplaatsen van de software sound leiding een toon.

```
10 FOR I=0 TO 100
20 OUT &HAB,15
30 OUT &HAB,14
40 NEXT
```

Door het invoegen van een regel 25, waarin bijv. een wachtlus, een >REM< of een >PRINT< instruktie staat, kan de snelheid waarmee het klikken uitgevoerd wordt, vertraagd worden. De frequentie van de trillin-

gen in de membraam wordt kleiner. Minder trillingen veroorzaken een lagere toon. Veel trillingen, dus een hoge frequentie, geeft een hoge toon.

De beide laatste bits van poort C zijn, in samenhang met de kassetterecorder belangrijk.

Bit 4 bestuurt de cassettemotor:

OUT &HAB,8 : Motor aan
OUT &HAB,9 : Motor uit

Bit 5 wordt uiteindelijk gebruikt om informatie op kassette op te slaan. >OUT &HAB,11< schakelt de magnetiseringsspanning in en met >OUT &HSAB,10< weer uit.

3.2. Geheugenindeling

Een der belangrijkste opdrachten van het PPI in het MSX systeem is wel de organisatie van de geheugenindeling.
Zoals U weet zijn er verschillende computerversies binnen de MSX standaard. Verschillende RAM-capaciteiten en/of toegevoegde ROM's met gebruiksklare gebruikersprogramma's (of BASIC uitbreidingen) voor verschillende speciale doelen zijn in de handel. Bovendien bestaan in principe de mogelijkheden de ROM of RAM capaciteit van de computer met modules uit te breiden.
Om al deze mogelijkheden te standaardiseren en compatible te maken werd een variabele geheugentoewijzing ontwikkeld.
De programmering van de diverse mogelijkheden volgt via poort A van de PPI.
Het adresgebied van de CPU dekt de adressen van 0 tot &HFFFF af. Deze 64K zijn in 4 pagina's van elk 16K onderverdeeld. Elk van deze pagina's kan een eigen geheugenbereik toegewezen krijgen, bijv. ROM de adressen 0 tot &H4000, Diskette cartridge &H4000-&8000 enz. Bij de toewijzing van geheugen aan een pagina zijn er telkens 4 mogelijkheden. Deze worden als de 4 slots van de computer aangemerkt. (wegens het cartridge-slot (insteekmodule)) Dat betekent:

slot 0 omvat in elk geval de ingebouwde BASIC-ROM (minstens

32K). Bestaan er meerdere, vast ingebouwde, ROM's dan zijn die meestal aan slot 0 toegekend. Verder bevindt zich in het bovenste slot 0 nog het RAM bereik van de gebruiker.

slot 1 is het geheugen beschikbaar komt uit een ingestoken module.

slot 2 is dat een RAM gebied, die minstens in de onderste gebieden tot &H8000 door alle 64K versies gebruikt wordt. Vaak zijn ook de bovenste gebieden van slot 2 met RAM geheugen vastgelegd.

slot3 is evenals slot 1 een uitbreidingsmodule van het geheugengebied. Slot 3 wordt meestal voor disketten gebruikt.

Elke 16K pagina van het CPU adressengebied kan, onafhankelijk van de andere pagina's, aan een slot toegewezen worden

adres	slot 0	slot 1	slot 2	slot 3
0-&H3FFF	operationeel systeem ROM	insteek module	(RAM)	insteek module
&H4000-&H7FFF	BASIC ROM	1	(RAM)	2
&H8000-&HBFFF	personal-programm ROM (SONY)	RAM		
&HC000-&HFFFF	RAM		RAM	

Wat kunnen we nu met deze informatie doen?

Op de eerste plaats is het mogelijk de geheugenkonfiguratie van Uw eigen computer te ontdekken.
Geeft U daarvoor in:

```
PRINT BIN$(INP(&HA8))
```

Verdeel dan het verkregen binair getal van rechts naar links in groepen van 2 bits. Onderzoek nu de waarde van elke 2bits groep. (de waarde ligt tussen 0 en 3).
De Sony Hit Bit heeft bijv.

10 10 00 00 hetgeen omgerekend 2 2 0 0 wordt.

Dit betekent dat in het bereik van &HFFFF-&H8000 (dus de twee bovenste pagina's RAM van slot 2) en van &H8000-&H0, dus de twee onderste pagina's ROM uit slot 2 geselecteerd zijn. Poort A geeft dus voortdurend inlichtingen over de geheugenconfiguratie.

Interessant is de mogelijkheid via >OUT< instrukties in poort A met BASIC de geheugenconfiguratie te manipuleren. Om zinvol te kunnen werken is hiervoor een 32K versie, beter nog een 64K versie, nodig. Omdat het uitschakelen van de ROM vanuit BASIC een onmiddellijke ineenstorting van de computer tot gevolg heeft (bijv. >OUT &HA8,&HCC<) moeten we eerst een experimenteergebied vastleggen. We verplaatsen het begin van de BASIC naar &HC000 (quasi 16K versie) en hebben daarmee pagina 3 (&H8000-&HC000) ter beschikking voor ons experiment.
Het verplaatsen van het BASIC gebied, vindt plaats door wijziging van adres &HF677, die de BASIC start bevat.

```
POKE &HC000,0: POKE&HF677,&HC0: NEW
```

De wijziging is niet direkt te bespeuren. Alleen zeer lange programma's kunnen nu niet meer geladen worden. Om de geheugeninhoud te zien volgt hier een minimonitor. Natuurlijk kan ook de monitor uit hoofdstuk 5 gebruikt worden.

```
10 FOR I=&H8000 TO &HB100 STEP 8
20 PRINT RIGHT$("000"+HEX$(I),4);" ";
30 FOR J=0 TO 7
40 BY=PEEK(I+J)
50 PRINT RIGHT$("0")+HEX$(BY),2);" ";
60 BY=BY AND 127
70 IF BY<32 OR BY>126 THEN BY=46
80 A$=A$+CHR$(BY)
90 NEXT J
100 PRINT A$;
110 A$="":NEXT I
```

We kunnen nu het gebied van &H8000 tot &HC000 (pagina 3) aan een ander slot toewijzen.

```
OUT &HAS, (INP(&HA8) AND &HCF) OR (3*16)
```

Hierbij staat de 3 voor het gewenste slotnummer.
Zoals reeds aangegeven, ligt bij de Sony Hit Bit het ingebouwde programma voor de personnal data bank in systeem slot no. 1.
Geheugenreservering van andere computers moet U ontlenen aan de meegeleverde handboeken.
Schakel nu van pagina 3 over naar pagina 1. Als geheugeninhoud verkrijgt U de mededelingen van de personnal data bank. Ook het uitlezen van de diskettenuitbreiding is op gelijke wijze mogelijk.

Nu nog iets wat alleen maar met de 64K versies mogelijk is.
Misschien heeft U zich al afgevraagd waarom U toch eigenlijk een 64K computer kocht, terwijl er slechts 28K (melding bij inschakelen) voor BASIC programma's vrij zijn.
Waar blijven dan de ontbrekende kilobytes?

Zij worden voortdurend door systeem slot 0, dus van de ROM in beslag genomen, d.w.z. met BASIC zijn de overige 32K RAM, dus de adressen &H0-&H8000, niet bereikbaar. Desondanks was Uw beslissing voor 64K juist. De werking van een diskdrive is slechts met 64K mogelijk. Bovendien kan met machinetaal het gehele gebied gebruikt worden.
Uiteindelijk bestaat nog de mogelijkheid 32K ROM, die niet te veranderen is, in het daaronder liggende RAM te kopiëren. Dan wordt naar slot 2, dus RAM, overgeschakeld, en de computerverwerking kan normaal doorgaan. Het beslissende voordeel daarbij is dan dat het gehele BASIC en het operationele systeem met >POKE< instrukties naar wens veranderd kan worden, omdat het in de RAM staat. We kunnen dan een eigen BASIC aanmaken, resp. het aanwezige naar eigen wensen modificeren.

Eerst dan het programma, dat de ROM inhoud van slot 0 in de RAM van slot 2 kopieert.

```
F000  210080   10          LD    HL,&H8000
F003  16A0     20          LD    D, &HA0
F005  1EAA     30          LD    E,&HAA
F007  0EA8     40          LD    C,&HA8
F009  F3       50          DI
F00A  2B       60 NEXT     DEC   HL
F00B  ED51     70          OUT   (C),D
F00D  7E       80          LD    A,(HL)
F00E  ED59     90          OUT   (C),E
```

```
F010  77     100        LD     (HL),A
F011  7C     110        LD     A, H
F012  B5     120        OR     L
F013  20F5   130        JR     NZ,NEXT
F015  C9     140        RET
```

Program: ramrom
start: &HF000 Einde: &HF015
lengte: &H16 bytes
Fouten: 0
Variabelentabel:
Next F00A

De BASIC lader:

```
10 CLEAR 200,&HEFFF
20 FOR I=&H7000 TO &HF015
30 READ A$:A=VAL("&H"+A$):POKE I;A:NEXT
40 DEF USR1=&HF000
50 X=USR1(1)
60 END
70 DATA 21,00,80,16,A0,1E,AA,DE
80 DATA A8,F3,2B,ED,51,7E,ED,59
90 DATA 77,7C,B5,20,F5,C9
```

Dit programma gaat uit van de volgende konfiguratie: (Sony Hit Bit)

	slot 0	slot 2
&H0000 tot &H7FFF	ROM	RAM
&H8000 tot &HFFFF	---	RAM

Voor andere opdelingen dienen de waarden dienovereenkomstig veranderd te worden.

pagina	0	1	2	3
A0 komt overeen met	ROM slot 0	ROM slot 0	RAM slot 2	RAM slot 2
AA komt overeen met	RAM slot 2	RAM slot 2	RAM slot 2	RAM slot 2

Nadat U het programma hebt gerund vindt geen merkbare verandering plaats, maar het verschil is opmerkelijk.
Schakel eerst met >OUT &HA8,&HA0< weer de oude configuratie in en probeer dan de toetsenbordtabel te wijzigen met >POKE &HDA5,32<. Adres &HDA5 is de ASCII-kode die aan de '0' toets is toegekend. Door de bovenstaande invoer verandert er niets aan het toetsenbord. Als U nu weer naar RAM overschakelt met >OUT &HAB, &HAA< en probeer nog eens met >POKE &HDA5,32< en dan de '0' toets op te roepen.
U krijgt nu een spatie (ASCII-kode 32). Met >POKE &HDA5,49< wordt de '1' op de '0' gelegd. enz.
Experimenteert U nog wat met de tabellen van de toetsenbordmatrix:

&HDA5- &HDD4	zonder shift
&HDD5- &HED4	met shift
&HE05- &HE34	met graph
&HE35- &HE5D	met graph/shift
&HE65- &HE94	met kode
&HE95- &HEC4	met kode/shift
&H1033- &H104A	stuurmodel; de onderste 3 rijen van de toetsenbordmatrix
&H104B- &H105A	tabel voor cijferblokken (mits aanwezig)
&H1061- &H1066	tabel van toetsen, waarop de speciale tekentoets van toepassing is.
&H1067- &H106C	Aangemaakt teken met speciale toets en toegestane toets.
&H106D- &H1072	als boven met speciale shifttoets
&H1073- &H1078	als boven met de speciale kode toets
&H1079- &H107E	als boven met kode/shifttoets
&H107F- &H109D	tabel van speciale letters, die ook betrekking hebben op de CAP vaststelling
&H109E- &H10BC	resultaat van speciale tekens bij ingeschakelde CAP

Indien U bij het omprogrammeren ergens verstrikt raakt, dan kunt U altijd met >OUT &HA8,&HA0< in ROM terugspringen en de oorspronkelijke toestand is weer hersteld.
Niet alleen de toetsentabel is nu rechtstreeks te veranderen, ook de originele tekendefinities staan natuurlijk in ROM. Tot op heden was het een probleem dat elke veranderde tekenset door een >SCREEN< instruktie direkt gewist werd. Bij >SCREEN< wordt steeds de inhoud van de ROM in de VRAM gekopieerd. De ROM tekentabel staat van adres &H1BBF tot &H23BE.
We zullen nu in de spatie een streep poken:

```
POKE &H1BBF+32*8+7,255
```

Tot nu toe is er nog niets gebeurd. We moeten het systeem nog meedelen dat de kopie in slot 2 staat op adres &HF91F.

```
POKE &HF91F,2
```

Het adres van de kopie staat in de adressen &HF929/21. In ons geval is ze nog altijd korrekt, d.w.z. nog steeds &H1BBF.
Geeft U nu >SCREEN 0< in en U verkrijgt de gewenste streep in de spatie. Met behulp van de adressen &HF91F en &HF920/21 kan ook, als de ROM ingeschakeld is, een alternatieve tekenset in de VRAM gekopieerd worden.

Misschien stoort het vraagteken U bij de >INPUT< instruktie. We kunnen dit nu op eenvoudige wijze vervangen door elk gewenst teken, bijv. met een spatie (ASCII kode=32).

```
POKE &H23D0,32
```

Probeert U nu: INPUT A$
Op de plaats waar het vraagteken moet staan is nu een spatie.
Interessant is ook >POKE &H23D0,29<. Daarmee begint de invoer exact op de aktuele positie.

3.3. De VDP als I/O apparaat

Vanuit BASIC is het ingrijpen in de VDP registers en op de video RAM met >VDP<, >POKE< en >VPEEK< mogelijk. Intern wordt de VDP en VRAM steeds als I/O apparaat behandeld.
De poortadressen &H98 en &H99 hebben betrekking op de VDP. De volgende informatie is speciaal voor machinetaalprogrammeurs belangrijk, omdat zij de ingreep in de VRAM met maximale snelheid mogelijk maken.
Er zijn 4 elementaire operaties:

1 Schrijven van gegevens in de VRAM.
2 Lezen van gegevens uit de VRAM
3 Schrijven van de VDP registerinhoud.
4 Lezen van statusregisters.

1. De overdracht van gegevens van de CPU in de VRAM via de VDP maakt gebruik van een zogenaamd 14-bit, automatisch verhogend (autoincrement) adressenregister. Bij elke ingreep, dus lezen of schrijven van de VRAM, wordt het adres van dit interne VDP register met 1 verhoogd. Om het adres op een bepaalde waarde te plaatsen, zijn twee gegevensoverdrachten noodzakelijk. Eerst worden de 5 laagwaardige bits (low bytes) via poort &H99 uitgevoerd. Direkt daarna volgen de resterende bits (high byte). Bij de tweede overdracht moet, om aan te geven dat een VRAM adres vastgelegd wordt, bit 6=1 en bit 7=0 zijn.
Voorbeeld:

```
    Adres &H2030 uit de VDP overbrengen:
    Low byte :&H30
    High byte:&H20
OUT &H99,&H30
OUT &H20 OR 2^6
```

Nadat het adres overgedragen is, kan via Poort &H98 de waarde in dit adres geschreven (>OUT<) worden.
>OUT &H98,20< schrijft bijv. de waarde in de aktuele adressen. Door de ingreep via poort &H98 is het adres automatisch tot &H2031 verhoogd. Een nieuw >OUT &H98, . . .< zou dan ook de waarde in dit adres schrijven. Indien zeer veel gegevens in opeenvolgende adressen moeten geschreven worden, brengt de autoincrement funktie van de VDP een aanzienlijk

snelheidsvoordeel. In plaats van 3 bytes is nu nog slechts 1 byte over te dragen.

2. De handeling van het adressen schrijven is identiek, alleen moeten nu bit 6 en bit 7 bij de 2e overdracht op 0 geplaatst zijn.
Voor >BY=PEEK(&H1234)< kan het volgende geschreven worden:

```
OUT &H99, &H34
OUT &H99, &H12
BY=INP (&H98)
```

Ook hier werd door >INP (&H98)< het adres automatisch tot &H1235 verhoogd.

3. Eerst worden via poort &H99 de in registers te schrijven gegevens overgedragen, daarna over dezelfde poort het registernummer, waarbij bit 7 geplaatst moet zijn.
Voor VDP(e)=4 kan geschreven worden:

```
OUT &H99,4
OUT &H99,3 OR 2^7
```

4. Ter controle van het statusregister is alleen de instruktie >BY=INP(&H99)< nodig. Deze instruktie komt overeen met >BY=VDP(8)<

Het gebruik van deze operaties is ook via BASIC mogelijk, maar leidt vaak tot complicaties, omdat de interrupt de gegevensoverdracht kan verstoren.

Printer

Voor de communicatie met de printer zijn de poorten &H90 en &H91 verantwoordelijk.
Eerst worden de gegevens met >OUT< naar poort &H91 gezonden. Via >INP &H90< wordt gewacht tot de LSB 0 is. Dit betekent dat de printer bereid is te ontvangen. Dan wordt via poort &H90 het strobe signaal naar de printer gestuurd.

PSG

Ook de >SOUND< instruktie is als een gevolg van de >OUT/INP< instruktie te beschrijven. >OUT &HA0,register< bepaalt het betreffende register.
>OUT &HA1,waarde< schrijft de waarde in het vooraf bepaalde register.

Bovendien beschikt de PSG nog over 2 I/O poorten, waaraan o.a. de joysticks aangesloten zijn.

Deze poorten worden via registernummers 14 (poort A) en 15 (poort B) geselecteerd. Voorts kan via poortadres &HA2 hun inhoud gelezen worden.

4: Geluid met de sound-chip

4.1 Inleiding

Om aan de soundgenerator van de MSX computer muziek te ontlokken, is enige elementaire kennis gewenst. Deze wetenschap zullen we U zeer kort en zeer pregnant doorgeven.

Het geluid

Onder geluid verstaan we alle hoorbare trillingen. Ze zijn voor ons hoorbaar als ze in een bereik van 16 tot 20000 herz liggen. Lager dan het gehoor ligt het infrageluid, hoger het ultra sonoor geluid. Geluid is te verdelen in geruis, klanken, tonen en mengsels hiervan. Een gebied van het geluid, met een naar cultuur en tijdvak te onderscheiden verdeling, met de hierboven genoemde verschijningsvormen noemt men muziek.

De toon

Onder toon verstaat men in de natuurkunde de zuivere sinustoon, die echter, met uitzondering van de elektronische muziek, niet in de muzikale praktijk voorkomt. Wat wij in onze taal als toon aanduiden, geldt in de akoestiek als klank.

De klank

Een klank ontstaat door het samenvallen van een basistrilling met de trilling van 'harmonische' boventonen die, ten opzichte van die basistrilling, in een veelvoudige verhouding staan.
Een instrument zoals piano of gitaar brengt nooit een toon voort, zoals wij die zoëven definieerden, maar altijd klanken. Daarbij ontstaat de typische klankkleur van een instrument door de boventonen. De struktuur van deze boventrilling is vaak zeer gecompliceerd, juist omdat deze tijdens het spelen nog verandert. Zo is de aanslag op een piano rijk aan boventonen, bij het afnemen daarentegen arm aan boventonen.

Zoals reeds vermeld, kan men met elektronische apparaten, waartoe computers gerekend moeten worden, zuivere trillingen, dus een natuurkundig zuivere toon, opwekken. Deze tonen klinken vaak enigszins 'iel'. Om een zo 'breed' mogelijke klank te produceren, stelt de sound chip enige funkties ter beschikking. Die maken het mogelijk de toon zodanig te veranderen,

dat die overeenkomt met de klank volgens ons gehoor. De PSG (Programmable Sound Generator) stelt de navolgende faciliteiten ter beschikking:

Verandering toonhoogte
: Er kan een toon opgewekt worden die van het onderste hoorbereik (ca. 27Hz) tot in het ultra sonoor geluidsbereik (dus onhoorbaar) reikt.

Veranderen van de ruisfrequentie

Veranderen van het volume

Klankeffecten
: Klankeffecten worden door verschillende modellen van volumewijziging verkregen.

Drie geluidskanalen
: Er kunnen drie tonen en ruiskanalen tegelijk gespeeld worden.

Voor het programmeren van de sound chip staan de volgende instrukties ter beschikking:

4.2 De >Play< instruktie

Met deze instruktie kunnen muziekstukken, na opgave van toonhoogte, tempo, tijd, volume, pauze en vorm, gespeeld worden. Het beinvloeden van de afzonderlijke parameters wordt door zg. subinstrukties bereikt.

Voorbeeld: Hey Jude/Beatles

```
10 T=80
20 T$="t=t;18":PLAY T$,T$+"14",T$
30 A1$="o5c4"
40 B1$="o5c"
50 C1$="r4"
60 A2$="o4a2r8ao5cdo4g2r4ga"
70 B2$="o3fffaccccc"
80 C2$="o3fcfcfcfcecececef"
```

```
90 A3$="b-4o5f4r8fecdc16o4b-16a2r8o5c"
100 B3$="eb-b-gfccc8f8"
110 C3$="gcb-cb-cb-cfcfcfcfc"
120 A4$="dd4dg16fe16e16f16dc2o4fgao5d"
130 B4$="o4b-b-b-b-ffcf"
140 C4$="o2b-fb-fb-fb-fo3cfcfcfcf"
150 A5$="dcr8co4b-aeff4c2."
160 B5$="b-b-ecaaa2"
170 C5$="o3ccccccccfcfco2f2"
180 PLAY A1$,B1$,C1$
190 PLAY A2$,B2$,C2$
200 PLAY A3$,B3$,C3$
210 PLAY A4$,B4$,C4$
220 PLAY A5$,B5$,C5$
```

Programma: Hey Jude

Regel 10

In deze regel wordt het tempo, waarmee de muziek gespeeld moet worden, vastgehouden. >T< kan daarbij waarden van 32 tot 255 bevatten. De waarden van 32 tot 255 zijn te vergelijken met het aantal kwart-noten dat in een minuut gespeeld kan worden.

Regel 20

Om de subinstruktie >Tn< variabel te houden, wordt binnen de string T=T geplaatst. Na deze invoer moet een semikolon geplaatst zijn.

>L8< en >L4< zijn sub instrukties die de lengte van de te spelen noten bepalen. De subinstruktie >Ln< kan waarden van 1 tot 64 bevatten. Deze waarden hebben de volgende betekenis:

1 = een hele noot
2 = een halve noot
3 = een derde noot
4 = een kwartsnoot enz.

Omdat in een muziekstuk veel kwart en achtste noten voorkomen, hebben we de subinstrukties >L4< en >L8< ondergebracht in T$. Daarmee wordt een minder omvangrijke invoer in A$, B$ en C$ bereikt.
Omdat de vaak gebruikte lengten in >Ln< opgeslagen zijn, worden deze instrukties niet meer in de stemmenstring gebruikt. Deze truc is in zoverre

nuttig dat hij de stringregels overzichtelijker en de invoer eenvoudiger maakt.

Regels 30 tot 50

In deze regels is de inzet van alle drie de stemmen opgenomen. De variabele >O< staat daarbij voor de octaaf. De subinstruktie >On< kan waarden van 1 tot 8 bevatten. O4 is de oktaaf van de zg. 'sleutelgat-C' op de piano. In de muziek noemt men ze ook vaak de 'eengestreepte octaaf'. De subinstruktie >C4< staat voor de toon C met een toonlengte van een kwart.

In regel 50 staat >R4< voor een kwart pauze.

Regels 60 tot 170

Bevatten in A$ de boven-, in B$ de midden- en in C$ de onderstem. Om de synchronisatie van de verschillende stemmen te verzekeren, is het raadzaam voor elke maat een regel te gebruiken.

Regels 180 - 190

Hier worden de in stringvariabelen opgenomen tonen in de gewenste volgorde opgeroepen.

4.3 De >Sound< instruktie

Met deze instruktie in men in staat direkt in het PSG register te schrijven. Daardoor is een speciaal ruis- en tooneffect te bereiken. Alleen met de >SOUND< instruktie is het mogelijk de volledige klankmogelijkheden van de PSG te benutten. Een voordeel van het >SOUND< statement is daarin gelegen dat de opgeroepen toon, resp. ruis, zolang blijft klinken totdat het betreffende register met een nieuwe >SOUND< aanwijzing weer uitgeschakeld wordt.

Reservering van registers

De PSG stelt in totaal 14 registers ter beschikking:

Bit	7	6	5	4	3	2	1	0
Register 0	Low byte van kanaal A							
Register 1	0	0	0	0	High byte KA			
Register 2	Low byte van kanaal B							
Register 3	0	0	0	0	High byte KB			
Register 4	Low byte van kanaal C							
Register 5	0	0	0	0	High byte KC			
Register 6	0	0	0	ruisfrequentie				
Register 7	0	0	GC	GB	GA	TC	TB	TA
Register 8	0	0	0	LA	Volume A			
Register 9	0	0	0	LA	Volume B			
Register 10	0	0	0	LA	Volume C			
Register 11	LB freq. volumewijziging							
Register 12	HB freq. volumewijziging							
Register 13	0	0	0	0	volumemodel			

Afkortingen:

LB = Low byte (zie hoofdstuk machinetaal)
HB = High byte
GC, GB, GA ruis inschakelen voor kanaal A, B en C.
La = Bij instelling van de waarde 16: wijzigen van het volumemodel met behulp van de ingestelde golfkurve

KA, KB, KC = Kanalen A, B en C
freq. = frequentie

De registers 0 tot 5

In deze registers kan de frequentie van de tonen in volgorde van low byte (laagwaardige byte register 0) naar high byte (hoogwaardig byte register 1) opgeslagen worden.

Om de frequentie van de 3 registers te berekenen, wordt gebruik gemaakt van de maatfrequentie van de computer (3.5 Mhz).

$$\frac{3579545 \text{ (Hz)}}{32 * \text{uitvoerfrequentie (Hz)}} = 256* \text{(gegevens uit registers 1,3,5)} + \text{(gegevens van registers 0,2,4)}$$

De formule is op de volgende manier te vereenvoudigen:

$$\frac{111860.8 \text{ (Hz)}}{\text{Uitvoerfrequentie (Hz)}} = 256 * \text{(gegevens registers 1,3,5)} + \text{(gegevens registers 0,2,4)}$$

Als we bijv. de basistoon 'A' (440Hz) willen opwekken, vindt de volgende berekening plaats:

$$\frac{111860.8 \text{ (Hz)}}{440 \text{ (Hz)}} = 254 = 256*0+254$$

We plaatsen nu de berekende waarden steeds in de beide registers, die voor de toonfrequentie van een kanaal moeten zorgen, bijv. register 0 en 1. De waarden zijn 0 voor register 1 en 254 voor register 0.
Met de >SOUND< kunnen we nu deze twee waarden overdragen.

```
SOUND 0,254
SOUND 1,0
```

Op deze wijze zijn frequenties van het laagste gehoorgebied tot aan het ultra sonoor geluid mogelijk.
In de registers 0, 2 en 4 kunnen waarden van 0 tot 255 opgenomen worden.
De registers 1,3 en 5 kunnen waarden van 0 tot 15 bevatten.

Register 6

Dit register bepaalt de ruisfrequentie en is op de volgende manier te berekenen:

$$\text{gegevenswaarde van register 6} = \frac{3\,579\,545 \text{ (MHz)}}{32 * \text{ruisfrequentie (Hz)}}$$

De gegevenswaarde kan 0 tot 31 zijn.

Vereenvoudigd ziet de berekening er als volgt uit:

$$\text{gegevenswaarde} = \frac{111\,860.8 \text{ (Hz)}}{\text{ruisfrequentie (Hz)}}$$

Als we een ruisfrequentie van 8000 Hz willen opwekken dienen we dit op de volgende manier te doen:

$$\frac{111860.8 \text{ (Hz)}}{8000 \text{ (Hz)}} = \text{ongeveer 14 (gegevenswaarde)}$$

In het register 6 moet een waarde van 14 opgenomen worden om een ruis met een frequentie van 8000 Hz te verkrijgen.
Register 6 mag waarden van 0 tot 31 bevatten, waardoor we in staat zijn frequenties van 3500 tot 111860 te verkrijgen.

Register 7

Dit register kiest een kanaal voor toon- en ruisopwekking. Waarden van 0 tot 63 kunnen doorgegeven worden.

	ruis			toon		
Kanaal	C	B	A	C	B	A
Waarde	32	16	8	4	2	1

Als we bijv. kanaal A van toon, kanaal B van ruis en kanaal C van toon en geruis willen voorzien, moeten we de navolgende berekeningen uitvoeren.

Max. = (kanaal+ kanaal) = gegevenswaarde

Max. = maximale, in dit register toegestane waarde (63)

Voor ons voorbeeld:

63 - (1 + 16 + 32 + 4) = 10

Deze waarde schrijven we in register 7.

>SOUND 7,10<

Voor gevorderden blijft nog op te merken dat dit register low georiënteerd is. Dat betekent dat toestand 1 van een bit UIT en 0 AAN is. Tegengesteld hieraan zijn er ook hoog georiënteerde registers. Hier betekent toestand 1=AAN en 0=UIT. De high georiënteerde registerbesturing wordt als de normale vorm aangeduid.

Registers 8 - 10

Deze registers maken het mogelijk het volume te regelen in waarden van 0 tot 15.

Register 8 = kanaal A
Register 9 = kanaal B
Register 10 = kanaal C

Wordt de waarde 16 in een der registers geschreven, dan wordt met behulp van registers 11, 12 en 13 een bepaald volumeverloop bepaald.

Registers 11 en 12

Hierdoor wordt de frequentie voor het veranderen van het volumepatroon aangegeven. Men kan een toon hier zodanig met patronen beladen dat een trillingstoestand optreedt die de klank een 'bovenaards' karakter geeft. De toon wekt de indruk te zweven.

Ter berekening van de perioden van volumeverandering wordt de volgende formule gebruikt:

$$\frac{3\,579\,545 \text{ (Hz)}}{532 * \text{periode (Hz)}} = 256 * (\text{gegevens R12}) + (\text{gegevens R11})$$

R=register

Vereenvoudigd is de formule

$$\frac{6728.5 \text{ (Hz)}}{\text{Perioden (Hz)}} = 256 * (\text{gegevens R12}) + (\text{gegevens R11})$$

Er is derhalve een frequentiebereik van 0.1 tot 7000 Hz mogelijk. Als we bijv. de opeenvolgende volumepatronen met een frequentie van 20Hz willen realiseren, dient de volgende berekening gemaakt te worden.

$$\frac{6728.5 \text{ (Hz)}}{20 \text{ (Hz)}} = \text{ca. } 336 = 256 * 1 + 80$$

In register 11 wordt aldus 80, in register 12 een 1 genoteerd.

Voor een beter begrip van de parameter voor de toonvorming volgt hier nog een voorbeeldprogramma. Geef dit in en start het met >RUN<

Programma: sinuscurve

```
10 REM Sinuskurve
20 SCREEN 2:COLOR 1,15,7
30 OPEN"grp:" FOR OUTPUT AS #1
40 LINE (15,10)-(245,180),1,B
50 LINE (235,95)-(20,95),8
60 LINE (20,20)-(20,150),1
70 PSET (90,2),0
80 PRINT#1,"Sinuskurve"
90 PSET(23,20),0
100 PRINT#1,"y"
110 PSET (237,95),0
120 PRINT#1,"X"
130 PSET (30,160),0
140 PRINT#1,"X=Zeit"
150 PSET (30,150),0
```

```
160 PRINT#1,"Y=Elongation"
170 PSET (80,30),0
180 PRINT#1,"Wellenlaenge"
190 PSET(210,70),0
200 PRINT#1,"Amp"
210 PSET (30,170),0
220 PRINT#1,"Amp=Amplitude"
230 LINE (60,40)-(60,100),3
240 LINE (185,40)-(185,100),9
250 LINE (60,40)-(185,40)
260 LINE (220,50)-(220,65),10
270 LINE (220,80)-(220,95),10
280 PSET(20,125),0
290 FOR X=20 TO 225 STEP 3
300 Y=90+SIN(X/20)*40
310 LINE -(X,Y),2
320 NEXT X
330 IF INKEY$="" THEN 330
```

Programmabeschrijving:

Nadat u het programma gestart heeft, wordt een sinuscurve op het beeldscherm getekend. Zoals we reeds vermeldden bestaat een toon uit trillingen. Een dergelijke trilling moet het beeld op het beeldscherm voorstellen.

Begripsverklaring:

Elongatie
: De uitslag op de Y-as, vanuit het nulpunt, wordt als elongatie aangeduid.

Toonlengte
: De toonlengte wordt door de waarden op de X-as aangegeven.

Amplitude
: De amplitude stelt de grootst mogelijke trillingsuitslag van een curve voor. Hoe groter de amplitude van een trilling is, hoe luider de toon en omgekeerd.

Golflengte
: Een golflengte is een bepaald trillingspatroon dat periodiek wordt herhaald.

Register 13
Dit register is bestemd voor de keuze van een bepaald patroon voor de volumeverandering. Dit patroon noemt men ook de geluidssterktefilter, omdat hier een grondtoon met een bepaalde trilling en amplitudegrootte omhuld wordt en daardoor de amplitudegrootte beïnvloedt. Het volume wordt dan veranderd volgens een door de filtercurve bepaalde vorm.
Register 13 staat ons nu de keuze uit totaal 8 verschillende filtercurven, die allen het volumeverloop in meerdere of mindere mate beïnvloeden. De verschillende vormen van de filtercurve vindt U terug in Uw handboek.
Geeft U nu het volgende programma in en start dit met >RUN<

Programma: envelope

```
10 MAXFILES=1
20 OPEN "grp:" FOR OUTPUT AS #1
30 DEFINT X,Y:DEFSNG S
40 SCREEN 2: COLOR 1,15,7
50 LINE (15,10)-(245,180),1,B
60 LINE (20,15)-(20,170),8
70 PRESET (30,180)
80 PRINT#1,"zaagtand"
90 PRESET (21,15)
100 PRINT#1,"y=volume"
110 PRESET (150,170)
120 PRINT#1,"x=tijd"
130 PRESET (238,95)
140 PRINT#1,"X"
150 LINE (236,95)-(20,95),8
160 LA=50
170 AN=200/LA
180 FY=60/LA
190 FOR I=1 TO AN STEP 2
200 LINE -(20+LA*I,95-LA*FY),2
210 LINE -(20+LA*(I+1),95),2
220 NEXT I
230 FOR X=20 TO 236 STEP 2
240 R=(X-20) MOD (LA*2):REM Rest naar LA * 2
250 AM=R*SGN(LA-.1-R)-LA*2*(R>=LA)
```

```
260 SY=SIN((X-20)/3)
270 Y=95-SY*AM*FY
280 LINE -(X,Y),1
290 NEXT X
300 CLOSE #1
310 IF INKEY$="" THEN 310
```

Programmabeschrijving:

Dit programma toont U hoe de filtercurve funktioneert. We hebben een driehoekstrilling gekozen (de 14e mogelijkheid met subinstruktie uit het handboek)
Als U nu de volgende regels verandert, krijgt U een model als de 12e mogelijkheid in het handboek.

```
120 LINE -(20+LA*(I-1),95-LA),2:REM Flanke
130 LINE -(20+LA*II,95),2:REM diagonaal
160 AM=LA-(X-20)MOD LA:REM Aktuele amplitude van de envelope
```

Door het veranderen van de volgende regels verkrijgt U de 8e mogelijkheid.

```
130 LINE -(20+LA*I,95-LA*FY),2:REM Flanke
140 LINE -(20+LA*(I+1),95),2:REM Diagonaal
170 R=(X-20) MOD (LA*2):REM Rest naar LA*2
180 AM=R*SGN(LA-.1-R)-LA*2*(R>=LA)
```

Probeert U ook eens de verschillende waarden voor de volgende variabelen:

LA - golflengte van de filtercurve
AN - Aantal tanden op de X-as
FY - strekkingsfactor Y-as
SY - sinusamplitude
Y - totale amplitude

Met de filtercurven zijn enige manipulaties met de tonen mogelijk. Probeert U maar eens, al experimenterend, enige interessante tonen of ruis voort te brengen.

Programma: Orgel

Het volgende programma realiseert het toetsenbord van een orgel

Het indrukken van de toetsen	A	W		S	E		D	F	G	Y	H	U	J	K
levert de volgende tonen:	C	Cis		D	Dis		E	F	G	Gis	A	B	Bes	C

Het bijzondere aan dit programma is dat accoorden (3 tonen tegelijk) gespeeld kunnen worden.

```
10 DEFUSR1=&HD12
20 SCREEN 0,,0
30 DEFINT A-Z
40 KB=&HFBF0
50 PP=&HF3F8
60 GP=&HF3FA
70 DIM F(25)
80 FOR I=65 TO 90
90 READ F(I-65):NEXT
100 SOUND 7,63:FOR I=8 TO 10:SOUND I,10:NEXT
110 P=10:GOSUB 190
120 A=PEEK(PP)-&HF0-P:IF A=0 THEN GOSUB 190:GOTO 170
130 IF A>3 THEN A=3
140 GOSUB 190
150 FOR I=0 TO A-1:T=(PEEK(KB+I)AND&B11011111)-65
160 SOUND 2*I,F(T)MOD256:SOUND 2*I+1,F(T)\256:NEXT
170 SOUND 7,64-2^(A)
180 GOTO 120
190 P=10-P
200 POKE PP,&HF0+P:POKE PP+1,&HFB
210 POKE GP,&HF0+P:POKE GP+1,&HFB
220 POKE &HF3F7,1
230 FOR W=0 TO 3:X=USR1(1):NEXT
240 RETURN
250 REM A-H
260 DATA 1710,0,0,1357,1439,1281,1141,1016
270 REM I-P
280 DATA 0,906,855,1,0,0,1,1
290 REM Q-Z
300 DATA 0,0,1524,1209,960,0,1615,0,1078,0
```

Programmabeschrijving orgel:

regel 10
Startadres van de routine die het toetsenbord beheert. Wordt normaal automatisch per interrupt opgeroepen.

regel 40
KB is het startadres van het KeyBuffer geheugen.

regel 50
PP is het zg. adres Put Point, waarop het aktuele einde van de keybuffer staat.

regel 60
GP is het adres Get Point, waarop het aktuele begin van de keybuffer staat.

regel 70
F bevat de aan onderhavige toets toegekende frequentie.

regel 80,90
In deze regels wordt, aan de hierboven beschreven toets, de frequentie toegekend (DATA regels 260 tot 300). De 0 staat voor niet gereserveerde toetsen.

regel 100
Alle kanalen aan, volume voor alle kanalen op 10 ingesteld.

regel 110
P is het verschil tussen de eerste en tweede buffer. Twee buffers zijn nodig, zodat een nieuwe toetsdruk in een buffer wordt opgenomen, terwijl de andere buffer nog open staat. De routine vanaf regel 190 initialiseert een keybuffer en roept de toetsencontrole op.

regel 120
Begin van het hoofdprogramma.
A bevat het aantal ingedrukte toetsen. Indien A=0 dan toetscontrole (GOSUB 190) en alle kanalen uit (GOTO170).

regel 130
De volgende 3 ingedrukte toetsen opnemen.

regel 140
Toetsenbord controleren

regel 150
De ingedrukte toetsen op volgorde uitlezen.

regel 160
Overeenkomstig de ingedrukte toets de frequentie instellen

regel 170
en de overeenkomstige kanalen inschakelen.

regel 180
Terug naar het begin van het hoofdprogramma.
regel 190
Routine toetsenbordcontrole. Bij elke controle wisselt P van 10 naar 0 en omgekeerd. Hierdoor wordt voortdurend tussen beide buffers gewisseld.
regel 200,210
Keybuffers wissen en startadres plaatsen
regel 200
Repeteervertraging uitschakelen.
regel 230
3 keer toetsenbordcontrole.

Programma: Synthesizer

Tot slot volgt een synthesizer programma, waarmee U diverse >SOUND< mogelijkheden van de MSX computer volledig kunt uitputten en op eenvoudige wijze uitproberen.

Het programma wordt zeer comfortabel via de joystickbesturing bediend. Het is mogelijk in het menu op het beeldscherm vrij te bewegen. Als U een waarde wilt veranderen drukt U slechts op de vuurknop. Nu kunt U door de beweging naar rechts de waarden verhogen, en door bewegen naar links verlagen. Er verschijnt, om deze mode aan te geven, een ster linksboven het gekozen begrip. Om weer in de normale mode terug te keren moet U de vuurknop nogmaals indrukken. Bij het verlaten van de invoermode verdwijnt de ster onmiddellijk van het beeldscherm.

```
10 REM Synthesizer
20 DEFINT A-Z
30 DEFSNG W,D,M,F
40 DIM W(7,3)
50 DIM X(8,4),Y(8,4)
60 SCREEN 1:WIDTH 29
70 TF#=1789772.5#
80 MF(2)=&HFFF:MF(5)=&H1F:MF(6)=2^16-1
90 PRINT" Kanaal 1 Kanaal 2 Kanaal 3"
100 PRINT"Toon: uit uit uit"
110 FOR I=1 TO 3:W(1,I)=1:NEXT
120 PRINT:PRINT"Fre:"
```

```
130 FOR I=1 TO 3:W(2,I)=300:NEXT
140 PRINT:PRINT "Ger: uit uit uit"
150 FOR I=1 TO 3:W(3,I)=1:NEXT
160 PRINT:PRINT"Vol:"
170 FOR I=1 TO 3:W(4,I)=10:NEXT
180 PRINT:PRINTSTRING#(29,"-")
190 PRINT" Ruis"
200 PRINT:PRINT"Fre:"
210 W(5,1)=
220 PRINT:PRINT" Envelope"
230 PRINT:PRINT"Fre:"
240 W(6,1)=10
250 PRINT:PRINT"Num:"
260 W(7,1)=9
270 FOR I=1 TO 4
280 FOR J=1 TO 3
290 X(I,J)=4+9*J
300 Y(I,J)=2*I
310 NEXT J,I
320 X(5,1)=5:Y(5,1)=14
330 X(6,1)=5:Y(6,1)=18
340 X(7,1)=5:Y(7,1)=20
350 FOR I=5 TO 7
360 FOR J=2 TO 3
370 X(I,J)=0:Y(I,J)=0
380 NEXT J,I
390 Y=4
400 FOR X=1 TO 3
410 LOCATE X(Y,X)-1,Y(Y,X),0
420 PRINTW(Y,X);
430 NEXT
440 X=1:Y=7:LOCATE X(Y,X)-1,Y(Y,X),0
450 PRINTW(Y,X);
460 SOUND 13,W(7,1)
470 Y=2
480 FOR X=1 TO 3
```

```
490 GOSUB 1120:NEXT
500 X=1:Y=5:GOSUB 1190
510 SOUND 7,63:R7=63
520 FOR X=1 TO 3:SOUND 7+X,W(4,X):NEXT
530 X=1:Y=6:GOSUB 1250
540 X=1:Y=1
550 ON STRIG GOSUB 700,700,700,700,700
560 EI=1
570 STRIG(EI) ON
580 GOTO 600
590 IF DX=0 AND DY=0 THEN 620
600 LOCATE X(Y,X),Y(Y,X),1
610 DX=0:DY=0
620 R=STICK(EI)
630 IF R=8 OR R=1 OR R=2 THEN DY=-1
640 IF R>3 AND R<7 THEN DY=1
650 IF R>1 AND R<5 THEN DX=1
660 IF R>5 AND R<=8 THEN DX=-1
670 IF X(Y+DY,X)=0 THEN DY=0
680 IF X(Y,X+DX)=0 THEN DX=0
690 X=X+DX:Y=Y+DY:GOTO 590
700 REM Strig
710 IF Y<>1 AND Y<>3 THEN 790
720 W(Y,X)=1-W(Y,X)
730 LOCATE X(Y,X),Y(Y,X),0
740 IF W(Y,X)=0 THEN PRINT"aan"; ELSE PRINT"uit";
750 BI=X-(Y=3)*3-1
760 R7=(R7 AND (NOT2^BI)) OR (W(Y,X)*2^BI)
770 SOUND 7,R7
780 GOTO 1100
790 IF Y<>4 AND Y<>7 THEN 950
800 IF LF THEN LF=0:RETURN ELSE LF=-1
810 LOCATE X(Y,X)-1,Y(Y,X)-1,0:PRINT"*";
820 FOR WA=0 TO 1000:NEXT
830 A=PEEK(&HF3E8)
840 IF (A AND 16) =0 THEN 1100
```

```
850 R1=STICK(EI):IF R1=0 THEN 830
860 IF R1>1 AND R1<5 THEN D=1
870 IF R1>5 AND R1>9 THEN D=-1
880 W=W(Y,X)+D
890 IF W>(16+2*(Y=7)) OR W<0 THEN 830
900 W(Y,X)=W
910 IF Y=4 THEN SPUND 7+X,W ELSE SOUND 13,W
920 LOCATE X(Y,X)-1,Y(Y,X),0
930 PRINTW;
940 GOTO 830
950 REM Frequenties
960 IF VF THEN VF=0:RETURN ELSE VF=-1
970 LOCATE X(Y,X)-1,Y(Y,X)-1,0:PRINT"*";
980 FOR WA=0 TO 1000:NEXT
990 DP=-1:DM=1:D=0
1000 A=PEEK(&HF3E8)
1010 IF (A AND 16) =0 THEN 1100
1020 R1=STICK(EI):IF R1=0 THEN DP=-1:DM=1:D=0:GOTO 1000
1030 IF R1>1 AND R1<5 THEN D=D+DP:DP=DP*2
1040 IF R1>5 AND R1<9 THEN D=D+DM:DM=DM*2
1050 W=W(Y,X)+D
1060 IF W<1 OR (W>MF(Y)) THEN 990
1070 W(Y,X)=W
1080 IF Y=2 THEN GOSUB 1120 ELSE IF Y=5 THEN GOSUB 1190 ELSE
GOSUB 1250
1090 GOTO 1000
1100 LOCATE X(Y,X)-1,Y(Y,X)-1,0:PRINT" ";
1110 LOCATE X(Y,X),Y(Y,X),1:RETURN
1120 REM Frequentie Reg Calc.
1130 W=W(2,X)
1140 F=TF#/16/W
1150 SOUND (X-1)*2,W-INT(W/256)*256
1160 SOUND (X-1)*2+1,INT(W/256)
1170 GOSUB 1310
1180 RETURN
1190 REM
```

```
1200 W=W(5,1)
1210 F=TF#/16/W
1220 SOUND 6,W
1230 GOSUB 1310
1240 RETURN
1250 REM
1260 F=TF#/256/W
1270 SOUND 11,W-INT(W/256)*256
1280 SOUND 12,INT(W/256)
1290 GOSUB 1310
1300 RETURN
1310 A$=LEFT$(STR$(F)+"          ",7)
1320 LOCATE X(Y,X)-1,Y(Y,X),0
1330 PRINTA$;
1340 RETURN
```

Programmabeschrijving:

regel 70
TF# bevat de halve maatfrequentie van de computer

regel 80
MF bevat de maximale registerwaarde voor:
toonfrequentie MF(2)
ruisfrequentie MF(5)
filterfrequentie MF(6)

regel 90 tot 200
Beeldschermopbouw en definitie van de bij de velden behorende gegevens.
W (I,J) bevat de gegevens van invoerregel I en kolom J.
Bij in/uit betekent de waarde 1 uit, de waarde 0 in. Bij alle andere waarden komt W(I,J) overeen met de betreffende registerinhoud.

regel 270 tot 380
Om het eenvoudig heen en weer springen tussen de invoervelden met de joystick mogelijk te maken, wordt hier elk invoerveld toegewezen aan de regel I, kolom J van de positie X(I,J),Y(I,J) op het beeldscherm. 0 betekent een ongeldig veld.

regel 390-530
Hier worden de bij het inschakelen aanwezige waarden door W(I,J)

in de frequentiewaarden resp. volumewaarden omgerekend en op het beeldscherm weergegeven.

regel 390 tot 430
Uitvoer van geluidssterkte

regel 440 tot 460
Uitvoeren en plaatsen van filternummer

regel 470-490
Uitvoeren en plaatsen van toonfrequentie

regel 500
Ruisfrequentie plaatsen en uitvoeren.

regel 510
Kanalen inschakelen

regel 520
Geluidssterktewaarden plaatsen

regel 530
Filterfrequenties plaatsen en uitvoeren

regel 540
Coördinaten van het eerste invoerveld.

regel 600
Cursor aanwijzen

regel 620
Joystick bewogen? Zo niet dan wachten.

regel 630-660
Bepalen van de richting

regel 670 tot 680
Nagaan of invoerveld in de aangevraagde richting geoorloofd is. Indien niet X(. . .)=0 dan beweging niet uitvoeren.

regel 690
Nieuw invoerveld plaatsen en opnieuw beginnen.

regel 710
Indien geen invoer van regel 1 (toon aan/uit) of regel 3 (ruis aan/uit) dan doorgaan.

regel 720
Waarden verwisselen (toestand aan/uit resp. 0/1)

regel 730
Cursor uitschakelen.

regel 740
Nieuwe toestand aangeven.

regel 750
Bitnummer voor het >SOUND< register berekenen.

regel 760
Byte voor register 7 berekenen.
regel 780
Einde strig
regel 790
Indien geen invoer regel 4 (volume) of 7 (filtercurve)
regel 800
Indien interrupt door 'tweede maal vuren' wordt uitgelokt de LF terugplaatsen en niet op interrupt letten. Anders LF plaatsen, zodat de volgende interrupt het einde van de invoer bewerkt.
regel 810
Ster uitvoeren als kenteken van invoer.
regel 820
Wachten tot vuurknop losgelaten wordt.
regels 830,840
Vuurknop voor de tweede keer ingedrukt? Zo ja dan einde invoer.
regel 850-880
Joystickrichting bepalen.
regel 890
Test op grenzen 0-16 bij volume 0-14 bij filternummer.
regels 900 tot 930
Nieuwe waarde plaatsen en uitvoeren.
regel 950
Frequentiewijziging aannemen.
regels 960 tot 1090
Analoog aan regels 800-930.
Verschil is de toename van de snelheid bij handhaven van een richting. Hiertoe worden DP resp. DM telkens met 2 vermenigvuldigd.
regel 1100
Einde strig routine, de ster wordt gewist.
regel 1110
De cursor wordt weer op het invoerveld weergegeven.

5. Machinetaal

5.1 Waarom eigenlijk machinetaal?

De meeste homecomputers hebben BASIC als voertaal. Zoals U reeds bemerkt heeft is deze taal niet moeilijk aan te leren. Speciaal het MSX BASIC munt uit door een grote hoeveelheid instrukties. Er ontstaat de indruk dat er geen wensen meer bestaan en alle programmeerproblemen hiermee opgelost kunnen worden.
Geeft U daarom het volgende programma in en let dan op de tijd.

```
5 SCREEN 2
10 HL=&H2000
20 A=1
30 VPOKE HL,A
40 HL=HL+1
50 IF HL<&H3000 THEN 20
60 IF INKEY$="" THEN 60
70 RETURN
```

Start het programma met >RUN< en let eens op wat er gebeurt. Als U terug wilt naar de invoer, druk dan op een toets.

Het volgende programma laadt het machineprogramma met dezelfde taak als het BASIC programma.

```
10 CLEAR 200,&HEFFF
20 FOR I=&HF000 TO &HF014
30 READ A
40 POKE I,A
50 NEXT I
60 DEFUSR1=&HF000
70 END
80 DATA &HCD,&H72,&H00,&H21,&H00,&H20,&H3E,&H01
90 DATA &HCD,&H4D,&H00,&H23,&H3E,&H30,&HBC
100 DATA &H20,&HF5,&HCD,&H9F,&H00,&HC9
```

Laad nu het machineprogramma met >RUN< en roep dan het geladen machineprogramma met >x=USR1(1)< op en verwondert U.

Zoals U gezien heeft loopt het:
- BASIC programma ongeveer 43 seconden
- Machinetaalprogramma minder dan 1 seconde

De lengte van het programma is:
Basic 97 bytes
Machinetaal 21 bytes

Analogie van de programma's:

Basic	Assembleertaal
5 SCREEN 2	- CALL &H0072
10 HL=&H2000	- LD HL,&H2000
20 A=1	- LD A,1
30 VPOKE HL,A	- CALL &H004D
40 HL=HK+1	- INC HL
50 IF HL<&H3000 THEN 20	- LD A,&H30
	- CP H
	- JR NZ,-11
60 IF INKEY$=" " THEN 20	- CALL &H009F
70 RETURN	- RET

Verklaring:

regel 5
CALL &H0072 roept de routine op, die de grafiekmode >SCREEN 2< inschakelt.

regel 10
Hier wordt de waarde voor de variabele HL resp. het register HL op het begin van het kleurgeheugen geplaatst. (LD is load = laden)

regel 20
In deze regel wordt in A de waarde van de uit te voeren kleur (1 = zwart) opgeslagen.

regel 30
Hier wordt de waarde van A in het beeldschermgeheugen geplaatst, hetgeen een zwarte streep toont.

Probeert U eens verschillende waarden voor het adres HL in het kleurge-

heugen (HL mag tussen &H2000 en &H3800 liggen) en voor A (de kode van de kleur) te zetten.

regel 40

Hier wordt de variabele HL, die het adres in het kleurengeheugen bevat, met een verhoogd, zodat het gehele beeldscherm volgeschreven wordt. (Eng.: INCrease = verhogen)

regel 50

Nagaan of HL groter is dan &H3000, dus of het einde van het kleurgeheugen bereikt is. Deze controle moet in machinetaal in 3 stappen uitgevoerd worden.

LD: laden van A met de vergelijkingswaarde (=High byte van &H3000).

CP H: Vergelijken met H, de high byte van HL (CP= ComPare= vergelijken).

JR: (Eng: Jump Relativ = relatieve sprong) NZ (Non Zero = niet nul). Men kan dus zeggen 'spring indien niet 0'

regel 60

Toetsenbordroutine met CALL &H009F oproepen. Zodra een toets ingedrukt wordt, volgt een terugkeer.

regel 70

RET beeindigt de routine.

Vervolgens tonen we U een assemblerlisting om U een voorbeeld te geven.

```
Adres kode      re-   Assemblerinstr.    Kommentaar
                gelnr
F000 CD7200      5    CALL &H0072        ; Routine grafiekmode
                                            >SCREEN 2< AAN
F003 21002      10    LD HL,&H2000       ; start kleurgeheugen
F006 3E01       20    LD A,&H01          ; kleur zwart (=1)
F008 CD4D00     30    CALL &H004D        ; uitvoerroutine teken op
                                            beeldscherm
F00A 23         40    INC HL             ; kleurgeheugenadres op-
                                            hogen.
F00B 3E30       50    LD A,&H30
F00D BC         60    CP H               ; H>&H30?
F00E 20F5       70    JR NZ,&HF003       ; Nee, dan nogmaals
F011 CD9F00     80    CALL &H009F        ; routine voor toetsen-
                                            bordcontrole.
F014 C9         90    RET                ; terug naar BASIC
```

De voordelen van machinetaal liggen dus voor de hand. Het zijn de verwerkingssnelheden en de zuinigheid met geheugenruimte. Sommige problemen zijn slechts met machinetaal effectief op te lossen, bijv. tekstverwerking, snelle opeenvolging van beelden bij grafieken (spellen en constructies). Ook sommige mathematische berekeningen met extreem nauwkeurige berekeningen, zijn slechts in machinetaal te verwezenlijken. Tot zover dan de voordelen.

Deze voordelen zouden voldoende moeten zijn om de noodzakelijkheid van machinetaalprogrammeren te benadrukken, ook al is de computer toegerust met een zeer goede BASIC, zoals bij de MSX computers. Er moet echter ook gezegd worden dat deze taal ook een groot nadeel heeft.

Machinetaal is de taal van de microprocessor (CPU) van de computer en daardoor de meest machine-georiënteerde taal. Een sterke machine-oriëntering heeft voor de programmeur tot gevolg, dat hij, om deze taal te begrijpen zeer abstract moet kunnen denken. De grondslag hiervoor is dat de CPU alleen met getallen kan werken, d.w.z. een machinetaalprogramma bestaat uitsluitend uit getallen en geen reeks van begrippen. In deze vorm zou de programmering in machinetaal van omvangrijke programma's nagenoeg onmogelijk. Daarom werd een soort tussentaal ontwikkeld, die machineprogramma's aanschouwelijker en begrijpelijker maakt. Deze taal

noemt men assembler. De assembleertaal wijst elke machinekode (dus een getal) een reeks van symbolen toe. Deze symbolen bestaan uit:

1 Instrukties, meestal een afkorting van het Engelse woord voor die instruktie, ook mnemonic genaamd.
2 Operands, die de adressen, constanten e.a. (met betrekking tot de instruktie) aangeeft.

Hiermee wordt het vervaardigen van machinetaalprogramma's vereenvoudigd tot het schrijven van assembleertaal. Deze assembleertaal wordt door een z.g. assembleerprogramma automatisch in machinekode vertaald. Bovendien is er nog de disassembler, een programma dat de getallenreeksen van de machinetaal weer in assembleertaal vertaalt. In het voorgaande voorbeeld heeft U een assemblerlisting gezien.

5.2 Getalstelsels

Omdat een computer intern alleen met getallen werkt, en voornamelijk met de getallen 0 en 1 (digitale computer), is het voor de machinetaalprogrammeur noodzakelijk iets over getalsystemen te weten. De volgende getalsystemen zijn voor ons van betekenis:

1. het decimale stelsel
2. het binaire stelsel
3. het hexadecimale stelsel.

Getalstelsels zijn, volgens een bepaald systeem, opgebouwde ordeningen van cijfers. Elk getal kan naar een ander getalstelsel omgerekend worden. In alle getalsystemen stijgt de plaatswaarde van een cijfer van rechts naar links.
Om de andere getalstelsels te kunnen verklaren gaan we uit van het decimale stelsel.

Het decimale stelsel

Duizend-tal	Honderd-tal	Tien-tal	Een-heid	plaatswaarde cijfers
7	3	5	6	

De plaatswaarde stijgt van rechts naar links.

Grondtal	verheven tot macht	wordt	uitspraak
10	0	1 E	enheid
10	1	10 T	iental
10	2	100 H	onderdtal
10	3	1000 D	uizendtal
10	4	10000 T	ienduizendtal
10	6	1000000 M	iljoental

Het decimale getal 1335 kan dan ook geschreven worden als:

1 D + 3 H + 3 T en 5 E – de laagste plaatswaarde staat het verst naar rechts

476 = 4 H + 7 T en 6 E

1335 is ook $1*1000 + 3*100 + 3*10 + 5*1$

maar ook $1*10^3 + 3*10^2 + 3*10^1 + 5*10^0$

De macht van een grondtal met een exponent 0 wordt gedefinieerd als 1.

Het binaire stelsel

Het binaire stelsel is op hetzelfde principe opgebouwd. Het verschil bestaat alleen uit machten van het grondtal 2 in plaats van 10.

Basis voor alle berekeningen is dus grondtal 2.

Het binaire getal 10101101 = decimaal 173

2^7		2^6	2^5	2^4	2^3	2^2	2^1	2^0	= plaatswaarde
	1	0	1	0	1	1	0	1	= getal
173=	2^7	$+2^6$	$+2^5$	$+2^4$	$+2^3$	$+2^2$	$+2^1$	$+2^0$	ofwel

173= $1*128 + 0*64 + 1*32 + 0*16 + 1*8 + 1*4 + 0*2 + 1*0$

Tot hiertoe hebben we de omrekening van binair naar decimaal getal gezien. Deze handeling is natuurlijk ook omkeerbaar. Ter verduidelijking van de omkeer nemen we het hierboven berekende decimaalgetal 173.
We moeten hiervoor nagaan welke macht van het grondtal 2 nog in dit getal opgenomen is. Ter ondersteuning: Het binair systeem kan als een getal van n plaatsen gebruikt worden. Omdat in dit computergebied in hoofdzaak 8-cijferige binaire getallen gebruikt worden, kunnen slechts de volgende machten van 2 gebruikt worden

machten van 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
omgerekende waarden	128	64	32	16	8	4	2	1	
In dit geval is 2^7 de hoogst voorkomende macht van 2									
Het verschil tussen 173 en 128 is 45 dus positief									1
Het verschil tussen 45 en 64 is geen macht van 2									0
Het verschil tussen 45 en 32 is 13 dus positief									1
Het verschil tussen 13 en 32 is geen macht van 2									0
Het verschil tussen 13 en 8 is 5 dus positief									1
Het verschil tussen 5 en 4 is 1 dus positief									1
Het verschil tussen 1 en 2 is geen macht van 2									0
Het restant is									1

We hebben dus achtereenvolgens de navolgende machten van 2 verkregen:

1 0 1 0 1 1 0 1 en derhalve de binaire waarde van 173.
Ter onderscheiding van de decimale getallen zullen we de binaire getallen met &B voorstellen dus
173 = &B10101101

Bit en byte

Een bit is de kleinste informatie-eenheid waaruit alle andere informaties samengesteld zijn. Bit is de afkorting van 'binairy digit', wat zoveel betekent als binair cijfer. Er wordt van een geplaatst bit gesproken als het bit de toestand 1 aangenomen heeft, of van een teruggeplaatst bit als het de toestand 0 heeft.

De MSX computers hebben een 8-bit processor, d.w.z. hij kan een binair getal van 8 bits verwerken. Dit komt overeen met decimaalwaarden van 0 tot 255.

Binair getal:	10110111
geeft een	90990999 bit (g=geplaatst, t=teruggeplaatst bit)
en een	76543210 plaatswaarde van de bits, ofwel bitnummer

Elk bit, elk cijfer, van een binair getal is een bitnummer toegewezen.
De bit met de laagste plaatswaarde, dus het meest rechts geplaatste, heeft nummer 0. Van rechts naar links wordt doorlopend genummerd. Het

bitnummer komt overeen met een macht van het grondtal 2, dat overeenkomt met diens plaatswaarde.
Bij computers is het vaak nuttig dat men zich de bit-toestand als een schakelaar voorstelt.

schakelaar aan = 1
schakelaar uit = 0

Bij een aantal van 8 schakelaars zijn de waarden van 0 tot 255 als 256 schakelsituaties voor te stellen.
Een samenvatting van 8 bits noemt men een byte. Een byte kan door de computer op een geheugenplaats weggezet worden. Hoe worden echter getallen weggezet die groter zijn dan 255? Voor dit doel deelt men een getal in 2 helften, t.w. de low byte (Eng.: laagwaardig) en de high byte (Eng: hoogwaardig). Deze bytes worden nu in twee opeenvolgende geheugenplaatsen afgelegd.

De high en low byte zijn op de navolgende manier te berekenen:

Getal, gedeeld door 256 = high byte
Rest van deze deling = low byte

Ter herinnering: Het getal 255 is de maximaal weer te geven waarde in een byte, omdat die uit 8 bits is samengesteld.
Voorbeeld: Het getal 34065 moet in een high byte en low byte verdeeld worden.

34065/256 = 133 rest 17
34065 = 133*256+17
133 = high byte
17 = low byte

De algemene formules in BASIC geschreven luiden:

1	HB=INT(getal/256)	HB=high byte
	LB=getal –HB*256	LB=low byte

Deze formule is voor getallen van willekeurige grootte te gebruiken.

2	HB=getal/256	HB=high byte
	LB=getal MOD 256	LB=low byte

De tweede formule is bruikbaar voor getallen tussen –32768 en 32767.
Een getal dat tussen 256 en 65535 ligt heeft voor de opslag dus 2 bytes

nodig. Om een vereenvoudiging in de opslag en weergave van getallen in deze vorm aan te brengen, is de invoering van een ander getalstelsel nuttig.

Het hexadecimale stelsel

Het grondtal van het hexadecimaal systeem is het getal 16
Ter herinnering:

Grondtal van decimale stelsel = 10
Grondtal van binaire getallen = 2

Het weergeven van cijfers, waarvan de waarde groter is dan 9 wordt in het hexadecimaal systeem weergegeven met de letters A t/m F.

decimaalsysteem

0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18, . . .

hexadecimaalsysteem

0,1,2,3,4,5,6,7,8,9, A, B, C, D, E, F,10,11,12, . . .

Laat ons eerst de hexadecimale getallen omzetten naar decimale getallen:

Grondtal	verheven tot macht	wordt
16	0	1
16	1	16
16	2	256
16	3	4096

&H3ABF = 3∗16^3 + 10∗16^12 + 11∗16^1 + 15∗1 6^0
&H3ABF = 3∗4096 + 10∗256 + 11∗16 + 15∗1
&H3ABF = 12288 + 2560 + 176 + 15 = 15039
&H3ABF = 15039

Nog een voorbeeld:
&H1A3E = 1∗16^3 + 10∗16^2 + 3∗16^1 + 14∗16^0
&H1A3E = 1∗4096 + 10∗ 256 + 3∗16 + 14∗1
&H1A3E = 4096 + 2560 + 48 + 14
&H1A3E = 6718

Nu volgt de omrekening van decimale getallen naar hexadecimaal.
De behandeling gelijkt op die van de vorige bladzijde. Nemen we aan dat

het decimaal getal 45380 moet in hexadecimaalsysteem overgezet worden.

stap 1 We overleggen welke de grootst mogelijke macht van 16 is die in dit getal kan voorkomen (ter beschikking staan de getallen in bovenstaande omrekentabel hexadecimaal naar decimaal).

stap 2 We delen het genoemde getal (45380) door deze waarde (4096) en herleiden het decimale getal van de uitkomst tot een hexadecimaal getal.

45380/4096 = 11 met rest 324
Decimaal 11 is hexadecimaal B

stap 3 Hetzelfde doen we met de rest (324) dus
324/256 = 1 rest 68
en herleid decimaal naar hexadecimaal 1
Deze stap wordt herhaald totdat de rest 0 is dus
68/16 = 4 rest 4
Het volgende cijfer (hexadecimaal) wordt 4
en de rest 4/1 = 4 dus 4e cijfer 4
Het herleide getal luidt dus &HB144

Het voordeel van het hexadecimaal systeem bestaat uit de direkte uitleesbaarheid van de low en high byte.

Voor &H3ABF geldt:

Het high byte is samengesteld uit de beide hexadecimale cijfers (3 en A) samen. Het heeft de decimale waarde van ***16^1 + 10*16^0) = 58

Het low byte is uit beide laatste hexadecimale cijfers samengesteld dus:
(11*16^1 + 15*16^1) = 191

Geeft U nu het volgende in:

```
PRINT PEEK(&H1D), PEEK(&H1E)
```

Op deze beide adressen &H1D en &H1E staat het sprongadres waarnaar het operationele systeem afbuigt, als een routine, bijv. in een insteekmodule, opgeroepen moet worden. Voor een sprongadres is een waarde van 0 tot 65535 (dus tot &HFFFF) mogelijk. Dit getal is met behulp van high byte en low byte opgeslagen. We willen nu dit sprongadres berekenen. Met de bovenstaande BASIC instrukties verkrijgen we aan adres &H1D de waarde

23, en op adres &H1E de waarde 2. Decimaal wordt dan het sprongadres 2*256 +23= 535.
We zullen nu dezelfde berekening in hexadecimaal uitvoeren.
23 = &H17 en 2 &H2. De waarde van het sprongadres verkrijgen we nu door het aan elkaar schrijven van de high byte en de low byte, dus &H217.
Het is dus even eenvoudig een hexadecimaal getal in high byte en low byte te verdelen, als de high byte en low byte tot een hexadecimaal getal samen te voegen.
Algemeen staat het low byte van een getal op het laagste geheugenadres, daarna volgt dan de high byte.
U leerde aldus het eerste voordeel van het hexadecimaal stelsel kennen. Bovendien is het overzetten van binair naar hexadecimaal systeem makkelijk te volbrengen. Hiervoor verdeelt men het binaire getal in 2 blokken van 4 bits. Het blok van 0 tot het 3e bit noemt men de low-nibble en het blok van de 4e tot de 7e bit de high nibble. Elke nibble bevat een hexadecimaal getal. Het is eenvoudig in te zien dat een 4 bits binair getal maximaal de waarde 15 kan aannemen (15 = 8+4+2+1). Alle waarden van 0 tot 15 kunnen ook hexadecimaal weergegeven worden (0, 1, 2 . . .9,A,B, . . .F).
Bekijk onderstaand voorbeeld:

1 1 0 1	1 0 0 1
high	low
nibble	nibble
8+4+1	8+1
13	9
&HD	&H9

dus &B11011001 = &HD9
Met enige oefening kunt U direkt uit een 4 bit getal het daarbij behorend hexadecimaal getal, en omgekeerd, aflezen. De volgende tabel kan U daarbij van dienst zijn.

Binair systeem	Hexadecimaal systeem	Decimaal systeem
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Op eenzelfde wijze vindt de omzetting van hexadecimaal naar binair plaats. Elk hexadecimaal cijfer wordt door de vier overeenkomstige bitcombinaties vervangen. Bijv. &HC7 = &B1100 0111
Het begrip voor de omzettingen naar de diverse getalsystemen is een elementaire basis voor het programmeren in machinetaal en ook bij een vergevorderde BASIC programmering onvermijdbaar.

5.3 De Z80A processor

Opbouw van de CPU

De MSX computers bezitten een Z80A CPU (centrale eenheid). We herinneren ons dat de CPU als de hersenen van de computer betiteld kan worden. Daarmee is de betekenis van deze MPU (Eng: Micro Processing Unit) geen probleem meer.
In dit hoofdstuk zullen we ons met de opbouw en de funktie van de afzonderlijke, in de CPU opgenomen, elementen bezighouden.

1. CU (Eng: Controll Unit of controle-eenheid).
   Alle handelingen in een computer worden door de CU gecontroleerd en bestuurd.
2. Controlebus
   De controlebus is de 'lange arm' van de CU. De elementen buiten de CPU worden hierin bestuurd en gecontroleerd.
3. Stapelplaats SP (Eng: Stack-Pointer)
   Met behulp van de SP's worden gegevens en sprongadressen van routines in de RAM opgeslagen. Omdat in de SP adressen opgeslagen worden is het een 16-bit register.
4. Instruktieteller PC (Eng: Programma Counter)
   De PC verwijst naar het geheugenadres, waar de te verwerken instruktie is opgeslagen
5. Register B tot L
   De CPU bezit meerdere registers waarin gegevens opgeslagen worden.
6. Flags (Eng: Flag = vlag, kenteken)
   Flags dienen als kenteken voor bepaalde gebeurtenissen, die bij rekenoperaties in de CPU ontstaan. Flags kunnen geplaatst (flag aan) of niet geplaatst (flag off) zijn
7. Accumulatoren (Lat: samenvoegen, verzamelen)
   De accu(mulator) is het belangrijkste register van de CPU. Hij wordt vaak het rekenregister genoemd.
8. ALU (Eng: Arithmetical Logical Unit), logische eenheid, rekeneenheid
   De ALU voert diverse arithmetische en logische operaties uit. Afhankelijk van het resultaat van de operatie worden flags beïnvloed
9. Rotatie. Voert rotatie- en schuifroutines uit

Zoals in punt 5 aangegeven, bevat de CPU meerdere registers. Tot een beter begrip hebben we die samengevoegd tot een vijftal groepen:
1 De accu's
2 De Flags
3 De 'Verbonden zes' 8 bit registers
4 De 'Onverbrekelijke vier' 16 bit registers
5 Interrupt en opfrisregisters.

De accumulator

De accu, resp. het A register is het belangrijkste register van de Z80. De meeste arithmetische en logische instrukties gebruiken dit register. Bij de uitvoering van een vergelijking wordt principieel met de inhoud van de

accu vergeleken. Zoals alle registers, met uitzondering van SP, PC IX en IY, is het A register een 8 bits register.

De flags

Het F, resp. het Flagregister is 8 bits breed (zoals A,B,C,D,E,H, en L). Het heeft echter andere funkties. In het flagregister worden de afzonderlijke bits als aanwijzing gebruikt voor bepaalde gebeurtenissen, die bij de operaties van de ALU's (rekenwerk) ontstaan. De afzonderlijke bits van het F register hebben de volgende betekenis:

S Z H P/V N C – Flagbetekenis
7 6 5 4 3 2 1 0 – bitnummer
C = Carry-overdracht
N = subtraktie
P/V pariteit en overflow
H = half-carry
Z = zero-null
S = sign, resp. voorteken.

C flag (bit 0)
Treedt bij een optelling of aftrekking een transport op, dan wordt dit bit geplaatst, anders teruggezet.

N en H flag (bit 1, bit 4)
Deze flags worden intern door de Z80 gebruikt. Ze hebben voor ons doel geen betekenis.

P/V flag (bit 2)
Deze flag heeft een dubbele funktie. Ze wordt geplaatst als een overflow (V), meer cijfers dan beschikbare posities, intreedt, overigens geeft het de pariteit (P) van een byte aan.

Z flag (bit 6)
Deze flag wordt geplaatst als het resultaat van een aftrekking 0 is, anders wordt ze teruggeplaatst. Bij een vergelijking wordt dit bit geplaatst als een gelijkheid optreedt.

S flag (bit 7)
Is het resultaat van een optelling, resp. aftrekking groter dan 127, dan wordt dit bit geplaatst. Zoals we later zullen zien hebben de CPU bytes die bij de arithmetiek groter zijn dan 127 een negatief getal tot gevolg.

De bits 3 en 5 van het flagregister worden niet gebruikt.

DE 'VERBONDEN ZES' 8 bit registers

Tot deze groep behoren zes 8 bits registers: B, C, D, E, H en L
Deze registers zijn in staat paren te vormen om samen een 16 bit breed register aan te maken. In de C, E en L worden dan de low, en in B, D en H de high bytes opgeslagen.

B/C (Byte-Counter = byteteller)
Het B register, resp. BC registerpaar wordt veelvuldig als teller, voor bijv. lussen, gebruikt.

Het DE registerpaar staat vrij ter beschikking. Dit registerpaar wordt vaak als tussenopslag van adressen of gegevens gebruikt.

H/L (high/low)
Het registerpaar H/L wordt vaak voor de opslag van adressen gebruikt.

Een gewenning aan de benaming van de registers is zeer aan te raden, om de instrukties op de hierboven beschreven manier te gebruiken. In principe kan men ook het L of E register als teller gebruiken. Een bijzonderheid van de Z80 is, dat alle bovenvermelde registers nogmaals met dezelfde funkties ter beschikking staan. Deze tweede registerset kunnen we wel gebruiken, maar er kan slechts een set tegelijk in gebruik zijn.

DE 'ONAFSCHEIDELIJKE VIER' 16-bit registers

Tot deze groep behoren vier 16-bit registers t.w. SP, PC, IX, IY.
Het SP register is een vast 16 bits register, d.w.z. het kan niet in twee 8-bits brede registers worden gesplitst. De SP wijst naar de adressen in het geheugen, waar sprongadressen of tijdelijk opgeslagen gegevens staan. Het adres heeft betrekking op een geheugenplaats, die binnen het bereik van de RAM ligt en die stack of stapel genoemd worden. Het gebruik van de stack voor gegevensopslag gaat als volgt in zijn werk:
Bij het inschakelen van de computer wordt de SP op het hoogste adres in de stack geplaatst (&HF000). Moet nu een byte op de stack gelegd worden, dan wordt de SP automatisch met een verlaagd en het byte in het adres, dat de SP nu aangeeft, opgeslagen. Hij wijst dus steeds naar het laatst ingevoerde item in de stack. Bij het 'halen uit de stack' verloopt de handeling in

omgekeerde volgorde. Eerst wordt de byte op het adres gelezen waar de SP op staat, daarna wordt de SP met één verhoogd. Op deze wijze is het mogelijk routines in willekeurige volgorde met elkaar te verbinden.
De PC is een bijzonder register. Het kan vanuit een programma noch beschreven, noch gewijzigd worden. De PC wordt intern beheerd en verwijst altijd naar het adres van de aktuele opdracht.

IX/IY registers worden voornamelijk gebruikt voor de opslag van adressen, resp. relatieve adressen. Ook deze beide registers behoren, evenals de overige onder 2.5 vermelde, tot de 16-bits registers.
Hierbij is het niet mogelijk afzonderlijk op high, resp. low bytes (zoals bij de BC, DE en HL) terug te vallen. Het gebruik van deze indexregisters komt overeen met de beide HL registers. Het verschil zullen we bij de geïndexeerde adressering leren.

Interrupt- en refreshregisters
Deze beide registers zijn toegevoegd aan de CU.

I, resp. Interruptregister. (Eng: interrupt = onderbreken)
Als een interrupt optreedt, d.w.z. als een programma wordt onderbroken, dan bevat dit 8 bit-register het bovendeel van het adres, waarheen afgebogen moet worden. Het onderste deel wordt door het element van de computer geleverd dat de interrupt veroorzaakte.

R resp. refreshregister (Eng: refresh = opfrissen)
Dit register wordt door de hardware als teller gebruikt om op gezette tijden de inhoud van het dynamisch geheugen op te frissen. Hierdoor wordt voorkomen dat opgeslagen informatie verloren gaat. Door het voortdurend herladen van dezelfde geheugeninhoud binnen zeer korte tijd wordt het verlies van gegevens voorkomen.

Het uitvoeren van een instruktie binnen de CPU ziet er dan als volgt uit:
De byte van het adres waarop de PC staat wordt gelezen en de PC wordt met een verhoogd, d.w.z. de PC staat nu op het volgende byte. Het gelezen byte wordt als instruktie geïnterpreteerd. Daarna worden de eventueel bij deze instruktie behorende gegevens gelezen (de PC wordt dan weer opgehoogd). Vervolgens wordt de instruktie uitgevoerd, en de bewerking begint opnieuw.

Nu we de Z80A wat beter kennen, zullen we ons met de invoer in machinetaal bezig gaan houden.

5.4 Invoer van machinetaalprogramma's

Om de instrukties van de Z80 uit te kunnen proberen, moeten we eerst eens nagaan op welke wijze een machinetaalprogramma via BASIC ingegeven en opgeslagen wordt. Evenals bij BASIC, waar we elke instruktie aan een regelnummer verbonden, wordt aan elke machinetaalinstruktie een adres toegewezen.

BASIC		Machinetaal		
regel	instruktie	adres	instr	kode
5	HL=HL+1	&HF009	INC HL	&H23
10	RETURN	&HF00A	RET	&HC9

- In BASIC wordt aan een regelnummer een opdracht gekoppeld.
- In machinetaal behoort bij elke instruktie een adres.

Een machinetaalprogramma is daardoor een reeks van instruktiekodes die in opeenvolgende adressen van het geheugen zijn opgenomen.
Vanuit BASIC hebben we de mogelijkheid, met behulp van de POKE instruktie de kodes op de betreffende adressen te schrijven. Een oproep van dit machinetaalprogramma komt dan tot stand met de >USR< instruktie. Vooraf dient dan wel het startadres met >DEFUSR< vastgelegd te worden. Het startadres is meestal het geheugenadres dat de eerste machinekode bevat. Om te voorkomen dat ons machinetaalprogramma overschreven wordt, moeten we het geheugengebied eerst met de >CLEAR< instruktie reserveren. We zullen met >CLEAR 200,&HEFFF< vaak het gebied van &HF000 tot &HF37F reserveren, waardoor &H380 bytes (ca 1KB) voor het machineprogramma ter beschikking staan. Een typisch BASIC programma, voor het laden van machinetaalprogramma's heeft de navolgende opbouw:

```
10 CLEAR 200,&HEFFF
20 FOR I=startadres TO eindadres
30 READ A
40 POKE I,A
50 NEXT I
60 DEFUSR1 = startadres
70 END
80 DATA . . .
90 DATA . . . enz.
```

In de DATA regels staan de kodes, die het eigenlijke machinetaalprogramma vormen. Het eindadres V (V-variabele, deze afkorting zullen we in de toekomst achter de woorden schrijven waar sprake is van variabelen) moet natuurlijk groter zijn dan &HEFFF en het startadres (V) kleiner dan &HF380 zijn. Het oproepen van het geladen programma volgt met >X=USR1 (parameter)<. Normaal gesproken zullen we &HF000 als startadres gebruiken. Eindadres (V) verkrijgen we uit het Startadres (V) plus lengte van het programma in bytes –1. De lengte van het programma komt overeen met het aantal ingevoerde kodes in de DATA regels.

Met behulp van de instrukties >DEFUSR nr< en >USR nr(parameter)< is het bij MSX BASIC mogelijk, tot 10 verschillende machineprogramma's te definiëren. We zullen in het vervolg steeds >USR1< gebruiken. De achter >USR< tussen haakjes staande parameter, heeft voor ons geen betekenis. Hier moet formeel een willekeurig getal of variabele ingevoerd worden.

Voor het ingeven van kleine programma's is het volgende programma zinvol:

```
10 CLEAR 200,&HEFFF:CLS
20 INPUT "Startadres &H";A$
30 ST=VAL("&H"+A$):IF ST<&HEFFF THEN BEEP:GOTO 20
40 AD=ST
50 IF AD>=&HF380 THEN BEEP: PRINT
"overschrijden":END
60 PRINT"Adres &H";HEX$(AD);" : ";
70 INPUT"Wert &H";A$
80 IF A$="" THEN DEFUSR1=ST:END
90 W=VAL("&H"+A$):A$="":IF W<0 OR W>255 THEN BEEP:GOTO 50
100 POKE AD,W:AD=AD+1:GOTO 50
```

U geeft de hexadecimale kodes direkt in, het programma zal het 'poken' voor U uitvoeren. Het hex. teken &H hoeft U niet mee in te geven.

Nu we de invoer van machinetaalprogramma's kennen, zullen we de instrukties van de Z80 eens wat nader bekijken.

Opmerking: Bij de verklaring van de instrukties zullen we vaak de analogie met BASIC instrukties aanhalen. Daartoe moeten we ons een register in BASIC als een variabele met dezelfde naam voorstellen. (register HL in machinetaal komt overeen met de variabele HL in BASIC)

5.5 De instrukties

De instrukties van de Z80 in 5 groepen onder te verdelen.

1. Transfer van gegevens
2. Bewerken van gegevens en testen
3. Sprongen
4. Stuurinstrukties
5. In- en uitvoer

Transfer van gegevens

Deze instrukties dienen voor de overdracht van gegevens.
Gegevens kunnen overgedragen worden van

a. register naar register.
   Dit komt overeen met het toewijzen in BASIC zoals A=B of SP=HL. De machinetaalinstruktie heeft het volgende format: LD A,B (LD=laad)

b. register naar geheugenplaats
   Bij de overdracht van register naar geheugenplaats is de BASIC instruktie >POKE geheugenadres,variabele<, zoals bijv. >POKE &HF000,HL, gelijkluidend aan de machinetaalinstruktie LD (&HF000),HL

c. geheugenplaats naar register.
   De gegevensoverdracht van geheugen in een register, bijv. LD H, (&HF005), komt overeen met de BASIC instruktie: >H=PEEK (&HF005)<

Bewerken van gegevens en testen

De instrukties ter bewerking van gegevens kunnen weer in 5 groepen onderverdeeld worden:

- Arithmetische operaties (bijv. ADDition, SUBtraction)

- Logische operaties (bijv. AND, OR)
- Getalinstrukties (INCrease=verhogen, DECrease=verlagen)
- Bitmanipulaties (SET en RESet)
- Schuiven en verwisselen van bits (rotate = roteren, shift = schuiven).

Bij het uitvoeren van deze instrukties worden registers of geheugenplaatsen (in de RAM) veranderd. Een aantal instrukties zijn gelijk aan het BASIC.

Assembler			BASIC
SUB	A,B	(SUBtraction)	A=A–B
ADD	HL,BC	(ADDition)	HL=HL+BC
AND	C		A=A AND C
OR	HL		A=A OR PEEK (HL)

Getest wordt op, ofwel afzonderlijke bits in de registers resp. geheugenplaatsen (bit-instrukties), ofwel vergelijken van register- of geheugeninhoud met de accu (CP = compare-instruktie). De ALU bepaalt aan de hand van de afloop van deze test of de flag in het F register geplaatst wordt of niet.

Sprongen

Met behulp van deze instrukties is het mogelijk aftakkingen in het machinetaalprogramma in te bouwen.
Men onderscheidt 3 sprongsoorten:

- direkte sprong naar een 16-bit adres (JP = jump)
- relatieve sprong naar aktueel adres (JR = Jump Relative)
- routinesprongen (CALL en RET terugsprong).

Men noemt een sprong voorwaardelijk als de beslissing van de sprong afhankelijk is van de status van een flag. Een voorwaardelijke sprong, waarbij de sprong afhangt van de status van een flag is bijv. JR NZ,&HF003.

Analogie:

Assembler	BASIC
JP	GOTO
CALL	GOSUB
RET	RETURN
JR	gelijk aan FOR-NEXT lus.

Stuurinstrukties

Met deze instrukties kan een programma onderbroken worden. Ook interruptbesturing is met deze instrukties mogelijk.

In-/uitvoer instrukties

De I/O instrukties dienen voor de communicatie met in/uitvoerrandapparatuur. Afhankelijk van het telkens aangesproken I/O poortadres worden de diverse opdrachten door deze instrukties uitgevoerd. Met deze I/O instrukties vaak aangeroepen IC's zijn:

PPI Programmable Peripheral Interface
PSG Programmable Soundgenerator
VDP Video Display Processor en daarmee de randapparaten:
toetsenbord, luidspreker, monitor, printer en cassetterecorder.

5.6 Voorbeeld van machinetaalprogrammering

Wij zullen nu stap voor stap een machineprogramma ontwikkelen. Het programma moet dezelfde opdracht hebben als het, in het hoofdstuk over grafiek beschreven, BASIC programma voor het invers weergeven van een teken.

Doel van dit machinetaalprogramma is het inverteren van de eerste 128 tekens van de tekengenerator, en wel zodanig, dat ze zowel normaal als invers ter beschikking staan. Dit probleem zullen we eerst in BASIC oplossen, maar wel zodanig dat het zo dicht mogelijk de machinetaalprogrammering benadert.

Voor de oplossing van het probleem hebben we de volgende informatie nodig:

Basisadres van de tekengenerator:

Het basisadres verkrijgen we met >PRINT BASE(2)<. Zij luidt 2048 ofwel &H0800.

Startadres van het eerste teken, dat invers weergegeven moet worden (spatie):

Omdat we vanaf de spatie alle tekens tot kode 128 willen behandelen, hebben we dit adres nodig. Zoals in het grafiek-hoofdstuk behandeld, berekent men de positie binnen de tekenset met:

Positie = ASCII*8

De ASCII kode van de spatie is 32, de berekening wordt dan

32*8=256

Deze waarde wordt bij het basisadres geteld om de plaats, waar de spatie staat, te verkrijgen.

2048 + 256 = 2304 ofwel &H900

Daarmee hebben we het aktuele geheugenadres van waaruit gelezen moet worden. Dit adres slaan we op in variabele (HL).

HL =2304 of HL=&H900

Adres van de geheugenplaats waar het eerste reserve teken opgeslagen moet worden (reserve spatie).

De reversed matrices moeten de kode van de normale tekens +128 hebben. De reversed spatie heeft aldus de kode 128+32=160. Daardoor wordt het startadres van de eerste spatie:

2048+160*8=3328 ofwel &HD00

De waarde &HC00 wordt ook in een variabele (DE) opgeslagen:

DE= 3328 of DE=&HD00

Teller voor het aantal te veranderen tekens (96).

De tekens met de kode 32 tot 127 moeten geïnverteerd worden, dat zijn 96 tekens.

Omdat elk teken door 8 bytes wordt weergegeven, moeten we dus 96*8 = 768 keer de lus voor het lezen en schrijven doorlopen. Deze waarde slaan we op in de variabele BC.

BC=76- of BC=&H300

De eerste regels van ons BASIC programma luiden nu:

```
10 HL=&H900
20 DE=&HD00
30 BC=&H300
```

Voor de machinetaalprogrammering:

Binnen de machinetaalprogrammering worden de hierboven berekende parameter niet in variabelen maar in registers opgeslagen. Jammer genoeg zijn in deze programmeertaal slechts enige registers voor ons doel geschikt, namelijk de registers HL, DE en BC. In die registers kunnen we nu de berekende waarden opslaan.

De instruktie daarvoor luidt:

LD Register, waarde

(LD=load=laden)

dus laad het register X met de waarde Y. Onze regels hebben in de assembleertaal het volgende format:

```
10 LD HL,&H900
20 LD DE,&HD00
30 LD BC,&H300
```

U zult zich nu wel afvragen wat dit te maken heeft met de waarden die in de DATA regels zijn opgenomen. Een machinetaalprogramma is, zoals reeds gezegd, een reeks van getallen, voor de vertaling van de assemblerinstrukties naar machinekode zijn er twee mogelijkheden. De komfortabele methode is de automatische vertaling:
De assembler vertaalt de, voor de programmeur makkelijk te bevatten, assembleertaal in die kodes. Is er echter geen assembler bij de hand, dan kan men met behulp van tabellen de assembleertaal in de betreffende kodes veranderen. In de vorm van kodes wordt dan een machinetaalprogramma in DATA regels opgeslagen.

Bijv. regel 10 van het machinetaalprogramma
10 LD HL,&H900 bestaat uit de kode's 21 00 en 09

De overeenkomstige DATA regel hiervoor luidt:

```
100 DATA &H21,&H00,&H09
```

waarbij de kode 21 de assemblerinstruktie (LD HL,waarde) voorstelt.
Als we de laatste 2 waarden op de gebruikelijke wijze, eerst low byte en daarna high byte, lezen, verkrijgen we de startwaarde van de standaardtekenset.

```
LB HB HB LB                    LB=low byte
00 09=09 00=&H0900=2304        HB=high byte
```

Dezelfde werkwijze passen we nu op de andere 2 regels toe:
20 LD DE,&HD00 wordt 11 00 0D
Hierbij staat de kode 11 voor: LD DE, waarde
De kode 00 0D voor &HD00

Om de volgende programmastap te volbrengen gaan we weer terug naar BASIC:

We lezen met >A=VPEEK(HL)< een reeks punten in de vorm van bytes van het standaardteken in en schrijven die, nadat we die met >A=A OR 255< geïnverteerd hebben, met >VPOKE DE,A< weer naar het adres dat we voor het reversed teken hebben voorzien.
De volgende BASIC regels:

```
40 A=VPEEK(HL): REM lezen
50 A=A XOR 255: REM inverteren
60 VPOKE DE, A: REM schrijven
```

In de machinetaal hebben we voor lezen en schrijven van een geheugenplaats twee routines nodig. Routines roept men op met

```
CALL adres
```

De adressen en de funktie van de routine moet men natuurlijk kennen. In het hoofdstuk systeemroutines kunt U een aantal van dergelijke adressen en funkties vinden. De volgende programmaregels leiden tot:

40	CALL	&H004A	komt overeen met kode	CD004A
50	XOR	255	komt overeen met kode	EEFF
60	EX	DE,HL	komt overeen met kode	EB
70	CALL	&H004D	komt overeen met kode	CD004D
80	EX	DE,HL	komt overeen met kode	EB

De regel 40 roept de routine op die een byte uit de VRAM leest. Het adres van het te lezen byte wordt bepaald door de inhoud van het HL register. De routine slaat de gelezen waarde op in de accu (A), d.w.z. in de volgende regel (50) is de waarde, die in BASIC met >VPEEK(HL)< in de accu gelezen werd, beschikbaar.

De regel 50 in machinetaal is nagenoeg identiek aan de BASIC regel. Zij geeft een geïnverteerde waarde van de byte, die weer in een reeks van 8 punten van de tekendefinitie staat.

Regel 60: De in de volgende regel opgeroepen routine moet ook het adres van de te beschrijven byte in het HL register overdragen. Dit doeladres willen we nu juist beschrijven met de informatie die weer in DE opgeslagen is. Daarvoor moeten DE en HL vooraf verwisseld worden. Dit bewerkt nu de EX instruktie (zoals >SWAP<).

In regel 70 wordt de routine voor het schrijven van een byte in de VRAM opgeroepen. Weer bevat HL het adres van de te beschrijven geheugenplaats. De waarde die op deze geheugenplaats opgeslagen moet worden, moet bij de oproep van de routine in de accu A opgenomen zijn. Dat is in ons programma het geval.
Alle logische instrukties hebben principieel betrekking op de accu, d.w.z. XOR inverteert de inhoud van de accu, ook als de A niet expliciet is vermeld.

Regel 80: Om de oude toestand voor de volgende oproep voor de VPEEK routine (regel 40) te herstellen wordt nogmaals met EX gewisseld.

De volgende programmastappen verhogen de opgeslagen waarden in HL en DE telkens met 1. Daardoor wordt bereikt dat de voor ons programma belangrijke geheugenplaatsen gelezen en beschreven worden.
Omdat we slechts 96 tekens willen veranderen, moet nog een teller aanwezig zijn, die naar 0 terugtelt en via een vraag het programma beëindigt. zodra alle tekens zijn gewijzigd. Deze teller is BX. BC wordt telkens met de waarde van 1 verminderd. De laatste regel stelt de vraag of BC=0.

```
 70 HL=HL+1 : REM  volgende geheugenplaats
 80 DE=DE+1 : REM  volgende geheugenplaats
 90 BC=BC-1 : REM  volgend teken
100 IF BC><0 then 40: REM indien alle tekens gelezen dan einde.
```

Naar machinetaal:

In de machinetaal verhoogt de instruktie

INC register

een registerinhoud met de waarde 1. INC staat daarbij voor het Engels increase = verhogen.

De instruktie

DEC register

doet het tegengestelde. Hij vermindert de registerinhoud met 1. DEC staat voor decrease, het Engelse woord voor verminderen. De volgende regels van het machineprogramma hebben dan als format:

90	INC	HL	komt overeen met kode	23
100	INC	DE	komt overeen met kode	13
110	DEC	BC	komt overeen met kode	OB

De volgende instrukties realiseren de controle, en daardoor de lus in regel 100 van het BASIC programma.

LD A,B

Laad de accu (A) met de waarde van B, dus het high byte van register BC.

OR C

OR C heeft, zoals alle logische instrukties (zie boven) betrekking op de accu. OR C betekent eigenlijk:
Verbind de inhoud van de accu met de inhoud van het C register met 'OR', en sla het resultaat weer op in A.

A=A OR C

De oorspronkelijke inhoud van A is door de voorafgaande instruktie de inhoud van B. Eigenlijk wordt dus B met C via de 'OR' verbonden. De omweg via de accu is noodzakelijk omdat alle logische instrukties de accu gebruiken.
De 'OR' verbinding heeft de eigenschap dat alle bits in het resultaat geplaatst zijn, die in een van beide verbonden waarden stonden. Dat betekent dat het resultaat van de verbinding slechts nul is, als beide verbonden waarden (hier B en C) gelijk zijn aan 0. Als dus na OR C = 0, dan is ook BC = 0, dus kan de lus beeindigd worden.
De arithmetische/logische instrukties beinvloeden allen de Z flag. Deze flag geeft aan of het resultaat van een instruktie 0 is, (Z=zero) of niet (NZ= Non Zero). Hierdoor kan een voorwaardelijke sprong volgen:

JR NZ, sprongdoel of JR NZ, verschil

(JR=Jump Relatieve= relatieve sprong)
(NZ=Non Zero= niet gelijk aan nul)
Deze spronginstruktie betekent zoveel als 'spring, als het resultaat geen nul is'.
Het sprongdoel wordt door het verschil in adres, van de na dit sprongbevel volgende instruktie, naar het doeladres aangegeven.
Negatieve getallen worden complementair aangegeven (d.w.z. 256-waarde).
Op de berekening van deze waarde gaan we hier niet in. Voor ons geval bedraagt het sprongverschil &HEF. Is aan de voorwaarde niet voldaan, hier dus de Z flag geplaatst, dan wordt de volgende instruktie verwerkt. Dit is het programma beëindigende RET (van RETURN) die een terugkeer naar BASIC veroorzaakt.

```
120 LD      A,B       komt overeen met kode    78
130 OR      C         komt overeen met kode    B1
140 LR      NZ, $-17  komt overeen met kode    20EF
```

De complete listing van het programma:

```
10  HL=2304                    of 10 HL=&H900
20  DE=3328                    of 20 DE=&HD00
30  BC=778                     of 30 BC=&H300
40  A=VPEEK (HL)
50  A=A XOR 255
60  VPOKE DE,A
70  HL=HL+1
80  DE=DE+1
90  BC=BC-1
100 IF BC<>0 THEN 40
```

En nu de BASIC lader, die het machinetaalprogramma met dezelfde opdracht laadt, zoals we hiervoor voordeden.

```
5 SCREEN 0
10 CLEAR 200,&HEFFF
20 FOR I=&HF000 TO &HF01A
30 READ A$
40 POKE I,VAL("&H"+A$)
60 NEXT
70 DEF USR1=&HF000
80 LOCATE 0,7:END
90 DATA 21,00,08,11,00,0C,01,00
100 DATA 04,CD,4A,00,EE,FF,EB,CD
110 DATA 4D,00,EB,,13,0B,78,B1
120 DATA 20,EF,C9
```

Start dit programma met >RUN< en roep het met >X=USR1< op.

De assemblerlisting ziet er als volgt uit:

Adreskode	regelnr.	assembler	kommentaar
F000 210009	10	LD HL,&H900	; startadres tekengenerator
F003 11000D	20	LD DE,&HD00	;
F006 010003	30	LD BC,&H300	;
F009 CD004A	40	CALL &H004A	; byte van adres HL uit VRAM lezen
F00C EEFF	50	XOR 255	; invers
F00E EB	60	EX DE,HL	; doeladres naar HL
F00F CD004D	70	CALL &H004D	; byte schrijven op adres HL in VRAM
F013 EB	80	EX DE,HL	; oude situatie herstellen
F014 23	90	INC HL	; HL met 1 verhogen
F015 13	100	INC DE	; DE met 1 verhogen
F016 0B	110	DEC BC	; BC met 1 verlagen
F017 78	120	LD A,B	; High byte van BC in accu laden
F018 B1	130	OR C	; Met low byte van BC vergelijken
F019 20EF	140	JR NZ,$-17	; Indien niet nul dan doorgaan
F01A C9	150	RET	; anders terug naar BASIC

Verklaring bij de assemblerlisting:

Adressen:

In deze kolom worden de adressen van de geheugenplaatsen geschreven, waarin de kodes van de machinetaal opgeslagen worden. Deze adressen zijn, evenals de kodes, in hexadecimale cijfers weergegeven.

Kode:

Hier staan de kodes van de afzonderlijke instrukties in de hexadecimale schrijfwijze. Twee cijfers vormen steeds een byte. Het aantal bytes is aldus eenvoudig vast te stellen.

Bijv. regels 10 en 20:

adreskode F000 210009 en F003 11000D

Het aantal kodes in regel 10 is 3, n.l. hexadecimaal 21=byte 1, 00= byte 2 en 09= byte 3. Deze instruktie heeft dus 3 bytes nodig, vandaar het adres van de volgende regel F003. De kodes vindt U ook in de DATA regels van het BASIC programma terug.

Regelnummer:
Hier staat het regelnummer nogmaals in decimale vorm.

Assemblerinstruktie:
Onder deze instruktie zijn de mnemonics met de daarbij behorende operands (konstanten, adressen, e.d.) opgenomen.
Bijv.

```
CALL &H0072
CALL = Mnemonic
&H0072 = Operand, in dit geval een adres.
```

Kommentaar:
Voor een beter begrip van het machineprogramma kan men hier verklaringen opnemen.

5.7 De monitor

Een onontbeerlijk hulpmiddel voor het vervaardigen van een machineprogramma is een zoganaamd monitorprogramma. Een monitor dient om geheugeninhouden te kunnen bekijken en te veranderen. Geheugengebieden kunnen op kassette opgeslagen worden, resp. van kassettes geladen worden.
Professionele monitoren hebben bovendien nog een aantal andere funkties zoals:

- disassembler.
  Hiermee kunnen geheugenplaatsen, die programma's weergeven, weer in assemblertaal vertaald worden.
- zoekfunktie.
  Hiermee kan naar een bepaalde bytevolgorde in het geheugen gezocht worden.
- start funktie.
  Hiermee kan het machineprogramma voor testdoeleinden opgeroepen worden.

- schuiffunktie.
  De geheugengebieden kunnen hiermee verschoven en of gekopieerd worden.
- miniassembler.
  Hiermee kunnen afzonderlijke assembler-instrukties in een kode vertaald worden.

In het volgende programma zijn enige mogelijkheden hiervan opgenomen.
Na de start van het programma geeft een ster aan dat invoer verwacht wordt. De faciliteiten van de monitor zijn door het ingeven van een letter op te roepen:

m	- monitor geheugengebied aanwijzen
S	- save
L	- load
a	- wijzigen van geheugeninhoud
g	- machineprogramma starten
r	- slot selecteren.

De afzonderlijke funkties:

Monitor (m)
M roept de basisfunktie van het monitorprogramma op, t.w. het tonen van geheugeninhouden. Direkt na de m dienen start- en eindadres van het geheugengebied uit- resp. ingevoerd te worden.
De invoer van elk getal wordt met RETURN afgesloten. De uitvoer van de geheugeninhoud volgt in het gebruikelijke format van een HEX DUMP (uitvoer geheugeninhoud in hexadecimale vorm).

Op de eerste plaats staat het aktuele adres, daarna volgen op dit, en de 7 volgende, adressen staande waarden in hexadecimale vorm.
Aan het einde van de regel staat het teken van de laatste acht kodes weergegeven, waarbij volgens de ASCII-kode geinterpreteerd wordt. Kodes, groter dan 127, worden met 128 verminderd, besturingstekens worden als punten weergegeven.

Save (S)
Hier moet start- en eindadres van het te beveiligen gebied ingegeven worden. Daarna moet de naam, waaronder opslag moet plaatsvinden, ingegeven worden.

Load (L)

Na de L moet de naam van de te laden gegevens vermeld worden.

Oproep machinetaalprogramma (g)

Na het ingeven van de letter g wordt het startadres van het op te roepen programma ingevoerd.

Wijzigen van geheugeninhoud (a)

Het adres van het eerste te wijzigen byte moet ingegeven worden. Daarna wordt telkens het aktuele adres weergegeven en dient U de gewenste waarde in te voeren.

Slot keuze (r)

Het nummer van het gewenste slot (0 tot 3) moet ingegeven worden. Direkt wordt dan de inhoud van de aangegeven slots getoond. Het gebied vanaf &HF000 mag alleen met slot 2 gelezen worden.

```
10 CLEAR 200,&HEFFF:MAXFILES=2
20 OPEN "CRT:" FOR OUTPUT AS #1
30 OPEN "LPT:" FOR OUTPUT AS #2
40 DEFSNG A-Z
50 KN=1:REM Kanaalnummer
60 AN=6
70 FOR I=1 TO AN:READ BF$(I):NEXT
80 DATA m,a,s,l,g,r
90 FOR I=&HF000 TO &HF022
100 READ A$:POKE I,VAL("&H"+A$):NEXT
110 DATA CD,8A,2F,EB,C1,E1,E5,C5,D5,CF,2C,CD,1C,52,FE,04
120 DATA D2,5A,47,E3,CD,0C,00,CD,CF,4F,E1,C1,D1,E5,C5,21
130 DATA F6,F7,C9
140 DEFUSR1=&HF000
150 SCREEN 0:WIDTH 38
160 LOCATE 10,2:PRINT"M o n i t o r"
170 ON KEY GOSUB 880,920,960,1000
180 KEY 1,"D.an":KEY(1) ON
190 KEY 2,"Help":KEY(2) ON
200 KEY 3,"Ende":KEY(3) ON
```

```
200 KEY 3,"Einde":KEY(3) ON
220 LOCATE 0,5
230 Y=CSRLIN:FOR I=1 TO AN:PRINT BF$(I);" : ":NEXT
240 LOCATE ,Y
250 LOCATE 5:PRINT"tonen"
260 LOCATE 5:PRINT"wijzigen"
270 LOCATE 5:PRINT"opslaan"
280 LOCATE 5:PRINT"laden"
290 LOCATE 5:PRINT"ML-prog. starten"
300 LOCATE 5:PRINT"slots selecteren"
310 PRINT
320 PRINT"Key 1 : printer aan/uit"
330 PRINT"Key 2 : commando's"
340 PRINT"Key 3 : einde"
350 PRINT"Key 4 : onderbreken"
360 PRINT
370 PRINT:LOCATE 0,CSRLIN-1:PRINT"* ";
380 A$=INKEY$:IF A$="" THEN 380
390 PRINTA$;" ";
400 FOR I=1 TO AN:IF (ASC(A$)OR2^5)<>ASC(BF$(I)) THEN NEXT
410 ON I GOSUB 440,610,690,740,780,830,870
420 GOTO 370
430 A$=""
440 REM Monitor
450 GOSUB 1040:REM invoer halen
460 IF (SN<>2) AND ((E-2^16*(E<0))>=49152!) THEN RETURN
470 FOR I=S TO E STEP 8
480 PRINT#KN,RIGHT$("000"+HEX$(I),4);" ";
490 A$=""
500 FOR J=0 TO 7
510 REM Byte lezen
520 B=USR1(I+J),SN
530 PRINT#KN,RIGHT$("0"+HEX$(B),2);" ";
540 C=B AND 127
550 IF C<32 OR C=127 THEN C=46
560 A$=A$+CHR$(C)
```

```
560 A$=A$+CHR$(C)
570 NEXT J
580 PRINT#KN,A$
590 NEXT I
600 RETURN
610 REM Wijzigen
620 INPUT S
630 PRINT"&H";RIGHT$("000"+HEX$(S),4);" : ";
640 INPUT W$
650 IF (ASC(W$)OR2^5)=ASC("e") THEN RETURN
660 W=VAL(W$)
670 POKE S,W
680 S=S+1:GOTO 630
690 REM Saven
700 GOSUB 1040
710 INPUT"Naam : ";N$
720 BSAVE "CAS:"+N$,S,E
730 RETURN
740 REM Laden
750 INPUT"Naam : ";N$
760 BLOAD "CAS:"+N$
770 RETURN
780 REM Start Programma
790 INPUT S
800 DEFUSR2=S
810 S=USR2(1)
820 RETURN
830 REM Slot Keuze
840 INPUT "Slotnummer : ";SN
850 IF SN>3 THEN 840
860 RETURN
870 BEEP:LOCATE 0:PRINT"    ";:RETURN
880 REM printer aan/uit
890 IF KN=1 THEN KN=2 ELSE KN=1
900 IF KN=1 THEN KEY 1,"p.aan" ELSE KEY 1,"p.uit"
```

```
910 RETURN
920 REM Help
930 FOR I=1 TO AN+4:PRINTCHR$(13):NEXT
940 LOCATE ,CSRLIN-AN-3
950 RETURN 230
960 REM einde
970 DEFUSR1=&H3E:X=USR1(1):REM norm. toetsdefinities
980 CLS
990 END
1000 REM Stop
1010 PRINT
1020 BEEP
1030 RETURN 370
1040 REM invoer start/einde
1050 Y=CSRLIN
1060 INPUT S
1070 LOCATE 14,Y:INPUT E
1080 RETURN
```

Programmabeschrijving:

Het overgrote deel van de monitor is met REM-regels gedokumenteerd. Het in regel 90 tot 130 ingelezen machineprogramma verandert de >USR< instruktie in een >PEEK< instruktie, waar eventueel een slotnummer aangegeven kan worden. Het uitlezen van de adressen vanaf &HC000 is alleen met slot 2 mogelijk. Het format hiervoor is:

USR1 (adres), slotnummer

De assemblerlisting voor uitbreidingen vindt U in de 'snuffelhoek'.
De hoofdlus van het programma vormen de regels 370 tot 420. Meer invoer is via de funktietoetsen mogelijk.
Belangrijk te vermelden is nog dat diverse getallen, zowel in hexadecimale als in decimale vorm ingegeven kunnen worden.

6. Systeemroutines

6.1 Het gebruik van systeemroutines

Het nu volgende hoofdstuk moet U de mogelijkheid geven machinetaalroutines te gebruiken. Aansluitend aan deze tekst vindt U een lijst van belangrijke systeemroutines. Een groot aantal van deze routines kunnen eenvoudig gebruikt worden, andere vragen weer een gedegen kennis van het programmeren in machinetaal.
In het allereenvoudigste geval kan een systeemroutine direkt opgeroepen worden:
Voorbeeld: wachten op toets

Het adres van deze routine is &H009F. Omdat geen parameters aangegeven moeten worden kan men volstaan met een >USR< instruktie voor dit startadres, dus:

```
DEFUSR1=&H009F
```

Zodra nu op een toets gewacht moet worden hoeft U slechts in te geven:

```
X=USR1(1)
```

Zodra aan een routine parameters verbonden moeten worden, is een minimachinetaalprogramma noodzakelijk dat de waarde inleest.

Voorbeeld: LD (HL),E met slotkeuze

Deze routine is noodzakelijk als we waarden willen schrijven in het verdekte RAM gebied (&H0 tot &H7FFF), zonder dit gebied met >OUT< te selecteren. Drie verschillende waarden moeten doorgegeven worden:

HL - adres
E - waarde
A - slotnummer.

Dit vindt plaats met het mini-assembler programma:

```
LD HL, adres
LD E, waarde
LD A, slot
CALL &H0014
RET
```

Met behulp van de instruktielijst van de laadinstrukties van de Z80 kunt U eenvoudig de kode bepalen.

```
LD HL,nn          kode 21 n n
LD E,n            kode 1E n
LD A,slot         kode 3E n
CALL &H0014       kode CD 14 00
RET               kode C9
```

De kode van de CALL instruktie is &HCD en die van de RET instruktie &HC9.
De BASIC lader heeft de volgende vorm:

```
10    CLEAR 200, &HEFFF
20    FOR I=HF000 TO &HF00A: READ A: POKE I,A:
      NEXT
30    DEFUSR1=&HF000
40    DATA
      &H21,0,0,&H1E,0,&H3E,0,&HCD,&H14,&H00,&HC9
```

Gemakshalve zijn de over te dragen waarden (adressen, waarde, slots) met nullen aangegeven.
De oproep van de routine ziet er dan als volgt uit:

AD bevat adressen
WA bevat de waarden
SL bevat het slotnummer.

```
100 POKE &HF001,AD-INT(AD/256)*256
110 POKE &HF002,INT(AD/256)
120 POKE &HF004, WE
130 POKE &HF006,SL
140 X=USR1(1)
```

Door de >POKE< instrukties worden telkens de aktuele waarden op de juiste plaatsen in het machinetaalprogramma geplaatst. Bij het oproepen van het machineprogramma worden ze dan volgens de assembler-instrukties geladen en overgedragen.

Moet een waarde, die door de machinetaalroutine is bepaald, in BASIC teruggegeven worden, dan wordt die op een vooraf bepaald adres opgeslagen.

De STRIG routine geeft in de accu 0 of 255 aan. Na het oproepen van de routine wordt de accu, bijv. op geheugenplaats &HF200 (dus binnen het met CLEAR gereserveerde geheugengebied) geschreven. Hiervoor dient de instruktie

LD adres,A

die het kode 32 bezit. Bij het overeengekomen adres &HF200, dus:

LD (&HF200),A Kode 32 00 F2

Daarna wordt met RET naar BASIC teruggekeerd, waar met >PEEK (&HF200) de verkregen waarde geladen kan worden.

6.2 Routines

Adres &H0000 : RESET

De oproep van deze routine veroorzaakt hetzelfde als een uit/inschakelen van de computer, resp. het indrukken van de RESET-toets.

Adres &H0008 : RST &H08 test op volgend byte

Er wordt getest of het in adres HL staande byte gelijk is aan het na RST &H08 volgende byte. Bij ongelijkheid wordt een 'syntax error' uitgevoerd. Anders wordt naar RST&H10 routine, vanaf adres &H4666, afgebogen.

Adres &H000C : LD A, (HL) met slotkeuze

A bevat bij de overdracht het gewenste slotnummer (0 tot 3). Als resultaat krijgt men in de accu en het E register de waarde van het in adres HL staande slot.

Adres &H0014 : LD (HL), E met slotkeuze

Bij overdracht bevat A het slotnummer, HL het adres en E de te schrijven waarde.

Adres &H0018 : RST &H18-uitvoer van tekens

Geeft het teken uit de accu aan het aktuele apparaat door. Standaard is dit het beeldscherm. Door het plaatsen van een waarde, anders dan nul op adres &HF416 wordt de printer geselecteerd.

Adres &H0020 : RST &H20-vergelijk HL met DE

De inhoud van het HL register wordt verminderd met de inhoud van het DE register. Afhankelijk van de uitkomst worden de flags beïnvloed. HL en DE worden hierdoor NIET gewijzigd.

Adres &H0028 : RST &H28 test op variabelentype

Verstrekt het type van de aktuele variabele, waarna geldt:

	Carry=0	(NC)	type 8 DEL
	Carry=1	(C)	type 2,3 of 4, en wel:
Sign	flag = 1	(M)	type 2 INT
Zero	flag = 1	(Z)	type 3 String
Sign	flag = 0	(P)	type 4 SNG

Adres &H0038 : RST &H38-interruptsprong bij INT MODE

Dit is het punt waar de routine moet inspringen die 50 keer per seconde door de standaard interrupt wordt opgeroepen.

Adres &H003E : standaard key reservering

Deze routine legt de funktietoetsen vast met de oorspronkelijke, bij inschakelen aanwezige, teksten.

Adres &H0047 : VDP register write

Het VDP register, met het in register C aangegeven nummer, wordt beschreven met de in register B opgenomen waarde. Dus VDP(C)=B.

Adres &H004A : Video RAM read

Het op adres HL in de video RAM staande byte wordt door deze routine in de accu geladen. BASIC: A=VPEEK(HL)

Adres &H004D : Video RAM write

De waarde van de accu wordt door deze routine op het adres HL van de video RAM opgeslagen. BASIC: VPOKE HL,A

Adres &H005F : Select SCREEN

Hier wordt de SCREEN mode naar de in de accu aanwezige waarde (0, 1, 2 of 3) geschakeld.

Adres &H0093 : PSG register write

Deze routine schrijft de, in register E opgenomen, waarde in het PSG register met nummer A. De programmering van de PSG in machinetaal is, voor het verkrijgen van komplexe klanken, zeer belangrijk. In BASIC komt deze routine overeen met de >SOUND A,E< instruktie.

Adres &H0096 : PSG register read

Na het oproepen van deze routine bevat de accu de waarde van het PSG register met het nummer, dat de accu voor het aanroepen bevatte.

Adres &H009F : wachten op toetsdruk

Deze routine wacht net zo lang totdat een toets wordt ingedrukt. De bij de toets behorende ASCII kode wordt geregistreerd en in de accu opgeslagen. Daarna volgt terugkeer.

Adres &H00AE : Invoer van regelbegin tot CR.

Door het oproepen van deze routine krijgt U de invoer van een gehele regel terug. Omdat een invoerregel tot 255 tekens lang kan zijn, moet die in de RAM worden opgeslagen. Dit gebeurt vanaf adres &HF55E. In het gebied van &HF55E tot &HF65D is aldus de laatste invoer opgeslagen. Bij terugkeer uit deze routine bevat HL het startadres van deze invoerbuffer minus 1.

Adres &H00C0 : BEEP

Voert een pieptoon uit

BASIC: BEEP

Adres &H00C3 : Clear screen

Wist het gehele gebied van het beeldscherm in alle modi.

BASIC: CLS

Adres &H00C6 : cursor plaatsen

De cursor wordt door deze routine op positie HL geplaatst, daarbij geldt:

H=regel

L=kolom

BASIC: LOCATE L,H

Adres &H00CC : KEY OFF

Schakelt de KEY-toewijzing uit.

Adres &H00CF : KEY ON

Schakelt de KEY-toewijzing in.

Adres &H00D5 : stick controle.

Deze routine, levert na overdracht van A op de gebruikelijke wijze de waarde van de richting aan de accu (zie handboek).

0 = toetsenbord

1 = joystick 1

2 = joystick 2

BASIC: A=STICK(A)

Ofschoon deze routine gelijk is aan die in BASIC, is ze zeer belangrijk, omdat juist de spelen een snelle controle van de joystick nodig maken.

Adres &H00D8 : STRIG controle

Deze routine levert na overdracht van A (zie boven)

0 in accu als 'vuurknop' (space) niet is ingedrukt.

255 in accu als 'vuurknop' (space) wel is ingedrukt.

Adres &H0132 : CAP aan/uit

Schakelt CAP aan of uit.

A=0 CAP aan

Adres &H0028 : RST &H28 test op variabelentype

Verstrekt het type van de aktuele variabele, waarna geldt:

	Carry=0	(NC)	type 8 DEL
	Carry=1	(C)	type 2,3 of 4, en wel:
Sign	flag = 1	(M)	type 2 INT
Zero	flag = 1	(Z)	type 3 String
Sign	flag = 0	(P)	type 4 SNG

Adres &H0038 : RST &H38-interruptsprong bij INT MODE

Dit is het punt waar de routine moet inspringen die 50 keer per seconde door de standaard interrupt wordt opgeroepen.

Adres &H003E : standaard key reservering

Deze routine legt de funktietoetsen vast met de oorspronkelijke, bij inschakelen aanwezige, teksten.

Adres &H0047 : VDP register write

Het VDP register, met het in register C aangegeven nummer, wordt beschreven met de in register B opgenomen waarde. Dus VDP(C)=B.

Adres &H004A : Video RAM read

Het op adres HL in de video RAM staande byte wordt door deze routine in de accu geladen. BASIC: A=VPEEK(HL)

Adres &H004D : Video RAM write

De waarde van de accu wordt door deze routine op het adres HL van de video RAM opgeslagen. BASIC: VPOKE HL,A

Adres &H005F : Select SCREEN

Hier wordt de SCREEN mode naar de in de accu aanwezige waarde (0, 1, 2 of 3) geschakeld.

Adres &H0093 : PSG register write

Deze routine schrijft de, in register E opgenomen, waarde in het PSG register met nummer A. De programmering van de PSG in machinetaal is, voor het verkrijgen van komplexe klanken, zeer belangrijk. In BASIC komt deze routine overeen met de >SOUND A,E< instruktie.

Adres &H0096 : PSG register read

Na het oproepen van deze routine bevat de accu de waarde van het PSG register met het nummer, dat de accu voor het aanroepen bevatte.

Adres &H009F : wachten op toetsdruk

Deze routine wacht net zo lang totdat een toets wordt ingedrukt. De bij de toets behorende ASCII kode wordt geregistreerd en in de accu opgeslagen. Daarna volgt terugkeer.

Adres &H5439 : GET ADR tussen haken

Leest een, tussen haken geplaatste, 16 bit waarde

Adres &H0103 : Omvang puntcoördinatie tekst

Bij overdracht puntcoördinatie voor >SCREEN 2 of 3<, waarbij de X-coördinaten in het DE register overgedragen worden, test deze routine de toelaatbaarheid van deze coördinaten.

C=0 coördinaten ontoelaatbaar.

C+1 coördinaten accepteren.

Indien >SCREEN 3< (multicolor) ingeschakeld, worden de coördinaten door 4 gedeeld, omdat een punt in deze mode uit 4*4 punten van >SCREEN 2< bestaat.

Adres &H0111 : Berekening puntadres.

Bij overdracht van X-coördinaten in C en Y coördinaten in E wordt het puntadres berekend en zowel aan het HL-register als op adres &HF92A/B overgedragen. De plaatswaarde die overeenkomt met de te plaatsen punt, wordt op adres &HF02C opgeslagen.

Adres &H0114 : Puntadres lezen.

Het adres van de laatst gelezen punt wordt in het HL register en de bitwaarde van de punt in de accu opgeslagen.

Adres &H0117 : Puntadres schrijven.

De huidige HL waarde wordt als een puntadres (&H92A/B) en de accu-inhoud als bitwaarde van een punt opgeslagen (&H92C).

Adres &H011A : Kleurkodetest

Er wordt nagegaan of de in de accu ontvangen kleurkode (<16) toelaatbaar is. Zo ja, dan wordt C teruggeplaatst en de kleur als aktuele puntkleur op adres &HF3F2 opgeslagen.

6.3 Systeemwijzigingen

Tenslotte geven we U nog een mogelijkheid om het operationele systeem, resp. de BASIC interpreter te veranderen. De programmeurs die het MSX operationele systeem hebben ontwikkeld, stond een techniek voor ogen die men als PATCHEN betitelt. (Eng: to patch = lappen, verstellen)

Het principiële probleem bij een wijziging is dat het ROM geheugen niet te wijzigen is. Een ingreep is alleen in de RAM liggende delen mogelijk. Om op vele plaatsen in te kunnen grijpen, wordt vanuit de belangrijke ROM

routines met routines naar het RAM vertakt. Bevindt het systeem zich nog in de uitgangstoestand, dan zijn alle in RAM gevulde adressen met kode &HC9, de kode voor RET, geladen. Dat betekent dat het aanroepen van dit adres geen invloed heeft en het ROM programma zonder veranderingen wordt voortgezet. Dit 'patchbereik' ligt in het RAM-systeem van &HFD9A tot &HFFC9.
Willen we nu een routine wijzigen, dan bestaat de mogelijkheid een 'patch' op de bij deze routine behorende RAM sprongadressen te leggen, d.w.z. dat de RET instruktie vervangen wordt door een kode met een andere betekenis, bijv. een spronginstruktie. Hierdoor wordt de RET kode bedekt (patched) met een lap (nieuwe routine) Voor elke patchsprong staan voor wijziging 5 bytes ter beschikking. Meer mogen er in geen geval veranderd worden, omdat dan de volgende routine gemanipuleerd wordt. Om de behandeling van de systeemroutine te verduidelijken volgt hier een voorbeeld.

Opdracht is de >INPUT< instruktie in BASIC zodanig te veranderen dat geen vraagteken meer uitgevoerd wordt. De INPUTroutine begint op adres &H23CC. Vertaalt U nu vanaf dit adres met de disassembler en U krijgt:

```
23CC  CDE0F0    CALL    &HFDE0
23CF  3E3F      LD      A,&H3F
23D1  DF        RST     &H18
23D2  3E20      LD      A,&H20
23D4  DF        RST     &H18
23D5  ..
```

Als eerste instruktie ziet U de sprong in het RAM patchgebied. Willen we de inputroutine modificeren dan staan ons de adressen &HFDE0 tot &HFDE4 ter beschikking. De kode voor '?' wordt in de volgende instruktie geladen en met RST &H18 uitgevoerd. Deze beide instrukties moeten oversprongen worden. Het programma moet op adres &HF23D2 voortgezet worden. De patch ziet er dan als volgt uit:

```
Startadres patch: &HFDE0
POP AF ; terugspringadres ophalen
JP &H23D2 ; vraagteken overspringen.
```

De kodes hiervoor zijn: F1, C3, D2, 23

Het volgende miniprogramma initialiseert de patch:

```
10 POKE &HFDE1,&HC3
20 POKE &HFDE2,&HD2
30 POKE &HFDE3,&H23
40 REM: aktiveren:
50 POKE &HFDE0,&HF1
60 REM  uitschakelen met &HFDE0,&HC9
```

Belangrijk is, dat het startadres van de patch, in dit voorbeeld &HFDE0, als laatste met de nieuwe waarde geladen wordt. Om uit de verandering terug te keren en weer in de uitgangssituatie te geraken, moet op deze plaats weer RET, dus &HC9, geschreven worden.

Het volgende programma geeft met >PRINT< gelijktijdig op printer en beeldscherm uitvoer. Het programma modificeert de RST &H18 routine zodanig dat ze twee keer opgeroepen wordt, een keer voor het beeldscherm en vervolgens voor de printer.

```
10 CLEAR 200,&HF300
20 POKE &HFEE5,0
30 POKE &HFEE6,&HF3
40 REM INIT POKE &HFEE4,&HC3
50 REM uit:POKE &HFEE4,&HC9
60 REM niet in direct mode
70 FOR I=&HF300 TO &HF30B
80 READ A$
90 POKE I,VAL("&H"+A$)
100 NEXT
110 DATA F5,3A,61,F6,32,15,F4,F1
120 DATA CD,63,1B,C9
```

Assemblerlisting:

```
FEE4 JP &H3000  RST &H18          ; patch
F300    F5      PUSH AF           ; ASCII kode redden
F301    3A61F6  LD A, (&HF661)    ; POS na
F304    3215F4  LD (&HF415),A     ; LPOS laden
F307    F1      POP AF            ; ASCII kode ophalen
```

```
F308   CD631B   CALL &H363     ; uitvoer op printer
F30B   C9        RET            ; doorgaan met beeld-
                                  scherm.
```

7. Peeks en pokes

7.1 Inleiding

In de voorgaande hoofdstukken hebben we veelvuldig de PEEK's en POKE's gebruikt. Hierna zal in het kort verklaard worden wat PEEK's en POKE's eigenlijk zijn. Daarna volgt een lijst met de belangrijkste POKE-adressen met hun funktie en gebruik.

Zoals U weet hebben MSX computers 32K ROM. Daarin is het operationele systeem opgenomen, dat het U mogelijk maakt op een eenvoudige wijze te programmeren. Tijdens het verloop van dit interne, geheel in machinetaal geschreven programma, komt een hoeveelheid informatie vrij die opgeslagen moet worden. Dat is bijv:
- welke beeldschermmode is gekozen
- welke kleuren zijn gekozen.
- informatie over ingeschakelde BASIC interrupt (>ON STRIG< etc.)
- beeldscherm regellengte >WIDTH<

De noodzaak deze informatie op te slaan leidt tot de volgende konsekwenties:

- Omdat in de ROM (geheugen met VASTE gegevens) geen informatie opgeslagen kan worden, moet een deel van de RAM hiervoor gebruikt worden. Bij MSX BASIC is dit het RAM-gebied van &HF380 tot &HFFFF.

- Omdat er op het terrein van de machinetaal geen variabelen, zoals in de betekenis van het BASIC voorkomen, worden telkens een of meer geheugenadressen van de RAM voor opslag van de informatie gebruikt.

- Deze, door het systeem vastgelegde geheugenplaatsen, worden door het systeem zelf gebruikt en onder voorwaarden veranderd. Omdat ze in het gebied van de RAM liggen, kunnen we ze met >PEEK< lezen en met >POKE< veranderen.

Met behulp van de PEEK's en POKE's zijn vaak dingen mogelijk, waarin het BASIC niet voorziet. Omdat een >POKE<, in tegenstelling tot de

>PEEK<, direkt ingrijpt op de werking van het systeem, kan een onvoorzichtig gebruik van deze instruktie een ineenstorten van het systeem, maar minstens een foutieve handeling van de computer tot gevolg hebben.

7.2 De instrukties

>PEEK adres< leest de waarde (1 byte) die op het aangegeven adres staat.

>POKE adres, waarde< schrijft de aangegeven waarde (1 byte) in het aangegeven adres.

Vaak vormen twee opeenvolgende adressen een paar, dat samen gelezen/beschreven moet worden.
Daarbij bevat het adres met het laagste adres het low byte (zie hoofdstuk 4) en de geheugenplaats met het hoogste adres het high byte. Om de waarde van beide geheugenplaatsen te verkrijgen moet de waarde van het high byte (HB) met 256 vermenigvuldigd worden en de waarde van het low byte (LB) hieraan toegevoegd worden. Voor het plaatsen van een 2byte-getal zie hoofdstuk 4.
Hier volgt nu een lijst van de belangrijkste POKE adressen en hun gebruik.

7.3 De adressen

F3AE LINL40 Regellengte van de tekstmode (max. 40 tekens)
Indien >SCREEN 0< wordt gekozen, dan wordt >WIDTH< op deze waarde ingesteld.
Voorbeeld: >POKE &HF3AE,20: SCREEN 0<

F3AF LINL32 Regellengte in grafiek 1 mode (max. 32 tekens).
Als >SCREEN 1< wordt gekozen dan wordt >WIDTH< op deze waarde ingesteld.
Interessant is dat in beide laatstgenoemde adressen ook grotere waarden dan de maximale mogelijk zijn. De invoer van een

dergelijke waarden heeft tot gevolg dat men met de cursor zijwaarts uit het beeldscherm kan lopen.

F3B0 LINLEN regellengte
Bevat de aktuele lengte van een beeldschermregel (>WIDTH<). Deze waarde kan direkt met >POKE< gewijzigd worden, wat overeenkomt met de >WIDTH< instruktie zonder >CLS<. Door >SCREEN< wordt de door >WIDTH< veranderde beeldscherminstelling weer op de bovenvermelde standaardwaarde geplaatst.

F3B1 CRTCNT Cathode Ray Tube CouNT = Beeldschermregeltel-ler
Het aantal regels per beeldscherm, d.w.z. er volgt, gelijktijdig met de opslag van de laatste regel, waarvan het nummer hier opgeslagen is, een uitvoer van de reservering.

F3B2 TABCNT TABulator CouNTer
Bevat het aantal spaties dat bij de tabulatorstops gegeven moet worden.

F3B4 TXTNAM TeXT NAaMtabel
Standaardadres van de naamtabel in de tekstmode

F3B7/8 TXTCGP TeXT Character Generator Pattern
Standaardadres van het tekenmodel generator in de tekst mode.

F3BD/E T32NAM Tekstmode (max. 32 tekens) NAaMtabel
Standaardadres van naamtabel in mode grafiek I

F3BF/C0 T32COL Tekstmode (max. 32 tekens) COLor tabel

F3C1 T32CGP Tekstmode (max. 32 tekens) Character Generator Pattern

F3C3/4 T32ATR Tekstmode (max. 32 tekens) sprite ATtRibutentabel

F3C5/6 T32PAT Tekstmode (max. 32 tekens) PATterntabel

F3C7/8 GRPNAM GRaPhic NAMe tabel

F3C9/A GRPCOL GRaPhic COLor tabel

F3CB/C GRPCGP GRaPhic Character Generator Pattern

F3CD/E GRPATR GRaPhic ATtRibute tabel

F3CF/D0 GRPPAT GraPhic PATtern tabel

F3D1/2 MLTNAM MuLTicolor NAMe tabel

F3D5/6 MLTCGP MuLTicolor Character Generator Pattern

F3D7/8 MLTATR MuLTicolor ATtRibute tabel

F3D9/A MLTPAT MuLTicolor PATtern tabel

F3DB CLIKSW CLIck SWitch

Geluid bij indrukken van toetsen, flag 0 = uit

F3DC CSRY CurSoR Y-positie

F3DD CSRX CurSoR X-positie

De cursorposities hebben altijd betrekking op de aktuele >WIDTH< definitie.

F3DE FNDFLG FuNction Display FLag

Voor gebruik zie hoofdstuk grafiek.

F3DF tot

F3E6 RGOSAV – RG7SAV ReGister in SAVe

Hier wordt de inhoud van de VDP registers 0 tot 7 opgeslagen. De VDP registers 0 tot 7 zijn slechts leesregisters. Omdat de aktuele waarde opgevraagd kan worden, is die hier opgeslagen. Deze geheugenplaatsen mogen niet gewijzigd worden omdat dit verwarring kan stichten.

F3E8 TRGFLG TRiGgerFLaG

Bit 4 van deze byte is 0, zodra de vuurknop ingedrukt wordt.

F3E9 FORCLR FOReground CoLoR

Aktuele schrijfkleur

F3EA BAKCLR BAcKground CoLoR

F3EB BDRCLR BorDeR CoLoR

F3F2 ATRBYT ATtRibute BYTe

Aktuele kleur voor hoog oplossende grafiek

F3F6 SCNCNT SCaN CouNTer

Teller die telkens van 3 naar 0 telt bij het doorlopen van de interrupt. Bij 0 worden interrupts van het toetsenbord en de STRIG getest.

F3F7 REPCNT REPeat CouNTer

Repeteerteller voor toetsenbord repeteerfunktie. Normaal op 13 geplaatst als toetst wordt ingedrukt. Verhindert dat een toets meerdere keren geregistreerd wordt. Indien direkt repeteerfunktie wordt gewenst REPCNT met 1 laden. Zie programma toetsenbordcontrole in hoofdstuk I/O

F3F8 PUTPNT PUT PoiNTer

Verwijst naar adres, waaronder de nieuwe invoer tijdelijk opgeslagen moet worden.

F3F9 GETPNT GET PoiNTer

Wijst naar het aktuele begin van de invoerbuffer, na de laatste buffercontrole. Het verschil tussen PUTPNT en GETPNT geeft het aantal intussen gedrukte toetsen aan.

F414 ERRNUM ERRor NUMber

Nummer van de laatstgeconstateerde fout.

F415 LPTPOS Line PrinT POSition.
Positie van de printkop in de aktuele regel

F416 PRTFLG PRiNTFLaG
Bevat PRTFLG een 1 dan geldt de algemene uitvoerroutine OUTDO via de printer, die door RST &H18 opgeroepen wordt.

F417 NTMSXP NoT MSX Printer
Komt overeen met de bij >SCREEN< mogelijke invoer voor MSX of niet-MSX printer.

F418 RAWPRT RAW PRinT
Is RAWPRT ongelijk aan 0 dan volgt de printuitvoer zonder voorafgaande opmaak (TAB, LPOS etc)

F41C/D CURLIN CURrent LINe
Aktueel regelnummer.

F663 VARTYP VAR TYPe kengetal

F676/7 BASSTA BASic STArtadres

F6AA AUTOFL AUTO FLag
1 betekent AUTO is ingeschakeld
0 betekent AUTO is uitgeschakeld.

F6AB/C Aktuele AUTO regel.

F6AD/E Stapgrootte voor AUTO

F5B3 ERRLIN ERRor LINe
Bevat de regel waar de laatste fout optrad.

F6C2 VARTAB VARiabelenTABel startadres

F6C4 VTBTAB Variabele TaB (arrays) TABel startadres

F7C4 TROFLG TROn FLaG
0=TROFF; 1=TRON

F85F MAXFIL MAXimum FILes aantal

F91F CGSLOT Character Generator SLOT
Slotnummer waarin de kopie van de tekenset staat.

F929/1 CGPNT Character Generator PoiNTer
Adres van de CG kopie

F922/3 NAMBAS NAMetabel aktueel BASisadres

F924/5 CAPBAS Character Generator aktueel BASisadres

F926/7 PATBAS sprite PATtern aktueel BASisadres

F928/9 ATRBAS sprite ATtRibute aktueel BASis adres.

FBB0 ENSTOP ENable STOP

Is ENSTOP=1 dan is een warme start via toetsenbord mogelijk. d.w.z. een onderbreking die direkt tot de normale invoermode leidt.

Schrijf eens een programma dat tegen elke onderbreking beschermd is (>ON STOP< etc.) en druk dan bij lopend programma gelijktijdig SHIFT+CODE+GRAF+CTRL en U bent weer in de invoermode!

Bescherm U daarvoor met >POKE &HFBB0<.

FBB1 BASROM BASic programma in ROM

Als BASROM<>0 dan betekent dit normaal gesproken dat een ROM BASIC programma loopt en dus niet onderbroken kan worden met CTRL-C en CTRL-STOP. Deze bescherming kan ook voor eigen BASIC programma's gebruikt worden met >POKE &HFBB1<.

FBCC CODSAV Bevat de kode van het teken waar de cursor op staat.

FC9E TIMER

Komt overeen met de variabele TIME

FCA9 CSRSW CuRSoR SHow cursor aan/uit

FCAA CSTYLE Cursor STYLE

Gehalveerde (insert) cursor aan/uit.

FCAB CAPST CAP STatus

Kleine letter/hoofdletter aan/uit

FCAF SCRMOD SCReen MODe aktuele screenmode

FCB0 OLDSCR OLD SCReen

Oude mode(tekst) indien grafiek ingeschakeld.

8. Basic intern

8.1 Inleiding

In het volgende hoofdstuk zullen we ons met de interne opslag van BASIC en de variabelen bezighouden. Het bezig zijn met dit thema geeft een middenweg tussen machinetaal en zuivere BASIC programmering.

Machinetaal, in zoverre dat de BASIC interpreter in machinetaal geschreven is en de interne structuren van BASIC programma's en variabelen door de interpreter begrepen en verwerkt kunnen worden.

BASIC natuurlijk door het feit dat het om de opbouw van de BASIC program ma's gaat. Voor eenvoudig programmeren is de kennis van de interne structuren niet beslist noodzakelijk. Zoals zo vaak kunnen, door het handig benutten van deze structuren vaak programma's geschreven worden die 'eigenlijk in 't geheel niet mogelijk zijn'. Daartoe behoren bijvoorbeeld:

- Het aanmaken van BASIC regels vanuit BASIC zelf. Daardoor zijn dan 'zich zelf schrijvende' programma's mogelijk.

- Programmeerhulpen, zoals DUMP (het uitvoeren van alle variabelen met hun waarde), XREF (betekenis van alle variabelen en de programmaregels waarin ze gebruikt werden), kunnen worden gerealiseerd.

- Een REM killer kan geprogrammeerd worden. Een dergelijk programma wist alle kommentaren in een programma en voegt, onder bepaalde voorwaarden, korte regels tot langere samen. Programma's worden daardoor vaak aanzienlijk sneller.

Deze voorbeelden moeten voldoende zijn om aan te tonen wat met de navolgende informatie gedaan kan worden.

8.2 Opslag van BASIC-regels

Omdat BASIC programma's met veranderlijke gegevens werken, moeten die in RAM opgeslagen worden. Het startadres van de BASIC RAM is bij 32K versies (en groter) &H8001, bij 16K versies &HC001. Principieel staat het startadres van de BASIC RAM in de adressen &HF676/77 als low en high byte.
De volgende BASIC regel voert het startadres uit:

```
PRINT HEX$(PEEK(&HF676)+256*PEEK(&HF677))
```

Om de interne opbouw van een BASIC programma te bekijken, hebben we een programma nodig, dat ons de inhoud van de geheugenplaatsen overzichtelijk weergeeft. Dit programma noemt men de monitor (zie hoofdstuk 5). Om de volgende gedachtengang te kunnen uitvoeren moet U beslist het monitorprogramma ingegeven hebben. Als eerste regel van de monitor geeft U in:

```
1 REM Data Becker boek.
```

Deze regel dient nu onderzocht te worden. Start nu de monitor en geef als startadres &H8000 en eindadres &H8017 in. U krijgt dan het volgende op Uw beeldscherm:

```
8000 00 18 80 01 00 8F 20 44 ........D
8008 61 74 61 20 42 65 63 6B ata Beck
8017 65 72 20 62 4F 65 6B 00 er boek
```

De betekenis van de afzonderlijke bytes is de volgende:

Adres	byte	betekenis
&H8000	&H00	nulbyte, kenteken basic begin
&H8001	&H18	
&H8002	&H80	De beide volgende bytes bepalen het geheugenadres, waarop de volgende regel begint. Omdat de low byte altijd als eerste staat, betekent dat in dit geval dat de volgende regel vanaf adres &H8018 opgeslagen wordt. (verbindingsadres)
&H8003	&H01	Na het verbindingsadres volgt in de 2 byte
&H8004	&H00	uitvoering het regelnummer, hier dus 1.
&H8005	&H8F	Hier begint nu de eerste programmaregel. &H8F is de token voor de REM-instruktie.
&H8006	&H20	Nu volgt de vastlegging na de REM, ze begint met een spatie (kode &H20)
&H8007	&H44	Letter D
&H8008	&H61	Letter a
&H8009	&H74	Letter t
ƒH800A	&H61	Letter a
&H800B	&H20	Spatie
&H800C	&H42	Letter B
&H800D	&H65	Letter e
&H800E	&H63	Letter c
&H800F	&H6B	Letter k
&H8010	&H65	Letter e
&H8011	&H72	Letter r
&H8012	&H20	spatie
&H8013	&H62	Letter b
&H8014	&H6F	Letter o
&H8015	&H65	Letter e
&H8016	&H6B	Letter k
&H8017	&H00	Dit nulbyte heeft als doel aan te geven dat hier de aktuele ingave van de regel beëindigd is.

Als U nu regel 1 wist en een andere regel probeert:

```
2 PRINT "Data Becker"
```

dan ziet U, na het ingeven van &H8000 en &H8017 voor de monitor, dat

de opbouw nagenoeg gelijk is.

&H8000 is het startbyte, daarna volgt het 2 byte verbonden adres.
Daarna het regelnummer, hier dus 2, en &H91, het token voor PRINT.
Vervolgens ziet U het aanhalingsteken (&H22) en de uit te voeren tekst (Data Becker). Na de tekst ziet U weer het aanhalingsteken en hde nulbyte.
De daarna volgende 2 bytes stellen het verbindingsadres voor de volgende regel voor. Het low byte staat op adres &H8014 en dat is juist het eerste verbindingsadres (zie boven). Aan de hand van deze adressen kan men zich door het BASIC programma worstelen. Om het principe volledig duidelijk te maken volgt hier een klein programma.

```
10 VA=&H8001: REM Startadres
20 ZN=PEEK(VA+2)+256*PEEK(VA+3):REM volgende verbindingsadres
30 VA=PEEK(VA)+256*PEEK(VA+1):REM volgende verbindingsadres
40 IF VA=0 THEN END
50 PRINT ZN:REM regelnummer naar scherm
60 GOTO 30:REM nogmaals verbindingsadres procedure
met nieuw adres
```

Indien bij Uw computer het BASIC niet op adres &H8001 begint, moet U regel 10 dienovereenkomstig wijzigen.
Als resultaat verkrijgt U op volgorde alle regelnummers (juist als bij LIST, uiteraard zonder inhoud).
Regel 30 leest de, op het verbindingsadres volgende, regelnummers.
In regel 40 wordt, met behulp van het oude verbindingsadres, dat natuurlijk steeds naar het volgende verwijst, het nieuwe gelezen.
Krijgt men als waarde een 0, dan betekent dit dat het einde van het programma is bereikt. Een programma wordt principieel met 3 nulbytes beëindigd. Daarbij wijst het eerste nulbyte het einde van de laatste programmaregel aan, de beide volgende stellen het verbindingsadres 0 voor.
Het einde van het programma wordt in regel 50 vastgelegd.
Hiermee hebben we de uitrusting van alle BASIC programma's leren kennen:

1 nulbyte	Als BASIC beginkenteken
2 bytes	eerste verbindingsadres
2 bytes	eerste regelnummer
.	
.	regelinhoud
.	
1 nulbyte	regeleindeteken
2bytes	tweede regelnummer
.	
.	regelinhoud
.	
1 nulbyte	regeleindeteken
2 nulbytes	einde van het programma

Bij de regelinhoud spelen de tokens een grote rol.

8.3 De tokens

Een token is een kode dat een BASIC instruktie voorstelt, d.w.z. dat de instrukties intern nooit woordelijk opgeslagen worden (dat zou veel te veel ruimte innemen). Om deze reden worden, na het beëindigen van de invoer, resp. wijzigingen, de diverse instrukties herkend en door de daarbij behorende kode (token) vervangen.
Er bestaat een eenvoudige truc om de toewijzing van tokens aan instrukties te weten te komen. Geeft U weer in:

```
1 REM hallo
```

Zoals we met behulp van de monitor vaststelden, staat het token voor REM direkt achter het regelnummer, dus op adres &H8005 (bij BASIC start op &H8000). >PRINT PEEK(&H8005)< geeft als resultaat 143 ofwel &H8F. Vervangen we nu dit token door een ander >POKE &H8005,&H91, dan krijgen we:

```
1 PRINT hallo
```
>POKE &H8005,&H84< leidt tot

```
1 DATA hallo
```
Dus is &H84 het token voor de >DATA< instruktie. Om niet alles moeizaam uit te proberen, volgt nu een lijst met tokens van alle BASIC instrukties.

DEC	HEX	INSTRUKTIE	DEC	HEX	INSTRUKTIE
129	81	END	130	82	FOR
131	83	NEXT	132	84	DATA
133	85	INPUT	134	86	DIM
135	87	READ	136	88	LET
137	89	GOTO	138	8A	RUN
139	8B	IF	140	8C	RESTORE
141	8D	GOSUB	142	8E	RETURN
143	8F	REM	144	90	STOP

DEC	HEX	INSTRUKTIE	DEC	HEX	INSTRUKTIE
145	91	PRINT	146	92	CLEAR
147	93	LIST	148	94	NEW
149	95	ON	150	96	WAIT
151	97	DEF	152	98	POKE
153	99	CONT	154	9A	CSAVE
155	9B	CLOAD	156	9C	OUT
157	9D	LPRINT	158	9E	LLIST
159	9F	CLS	160	A0	WIDTH
161	A1	ELSE	162	A2	TRON
163	A3	TROFF	164	A4	SWAP
165	A5	ERASE	166	A6	ERROR
167	A7	RESUME	168	A8	DELETE
169	A9	AUTO	170	AA	RENUM
171	AB	DEFSTR	172	AC	DEFINT
173	AD	DEFSNG	174	AE	DEFDBL
175	AF	LINE	176	B0	OPEN
177	B1	FIELD	178	B2	GET
179	B3	PUT	180	B4	CLOSE
181	B5	LOAD	182	B6	MERGE
183	B7	FILES	184	B8	LSET
185	B9	RSET	186	BA	SAVE
187	BB	LFILES	188	BC	CIRCLE
189	BD	COLOR	190	BE	DRAW
191	BF	PAINT	192	C0	BEEP
193	C1	PLAY	194	C2	PSET
195	C3	PRESET	196	C4	SOUND
197	C5	SCREEN	198	C6	VPOKE
199	C7	SPRITE	200	C8	VDP
201	C9	BASE	202	CA	CALL
203	CB	TIME	204	CC	KEY
205	CD	MAX	206	CE	MOTOR
207	CF	BLOAD	208	D0	BSAVE
209	D1	DSKO$	210	D2	SET
211	D3	NAME	212	D4	KILL

DEC	HEX	INSTRUKTIE	DEC	HEX	INSTRUKTIE
213	D5	IPL	214	D6	COPY
215	D7	CMD	216	D8	LOCATE
217	D9	TO	218	DA	THEN
219	DB	TAB(	220	DC	STEP
221	DD	USR	222	DE	FN
223	DF	SPC(	224	E0	NOT
225	E1	ERL	226	E2	ERR
227	E3	STRING$	228	E4	USING
229	E5	INSTR	230	E6	'(REM)
231	E7	VARPTR	232	E8	CSRLIN
233	E9	ATTR$	234	EA	DSKI$
235	EB	OFF	236	EC	INKEY$
237	ED	POINT	238	EE	>
239	EF	=	240	F0	<
241	F1	+	242	F2	-
243	F3	*	244	F4	/
245	F5	^	246	F6	AND
247	F7	OR	248	F8	XOR
249	F9	EQV	250	FA	IMP
251	FB	MOD	252	FC	Û

In deze lijst duiken enige instrukties op die U waarschijnlijk niet zult kennen. Dit zijn dan instrukties die U pas kunt gebruiken nadat U een diskdrive aansluit op Uw computer.

Zoals U ziet zijn alle tokens groter dan &H80= 128. Bijna alle getallen tot 255 zijn bezet, er ontbreken echter nog enige funkties, zoals bijv. >PEEK<.
Geeft U nu >POKE &H8005,255:POKE &H8006,&H97< en >LIST< in.

Inplaats van de >REM< instruktie, die oorspronkelijk op deze regel stond is nu de >PEEK< funktie verschenen, d.w.z. de funkties van het MSX BASIC worden door 2 tokens versluierd. Eerst het token met de waarde 255. Het geeft aan dat nu een funktie moet volgen. Hierna volgt het

eigenlijke token van de funktie. Hier volgt nu een lijst van de funktietoken. (vooraf moet altijd 255 staan)

DEC	HEX	INSTRUKTIE	DEC	HEX	INSTRUKTIE
129	81	LEFT$	153	99	SPACE$
130	82	RIGHT$	154	9A	OCT$
131	83	MID$	155	9B	HEX$
132	84	SGN	156	9C	LPOS
133	85	INT	157	9D	BIN$
134	86	ABS	158	9E	CINT
135	87	SQR	159	9F	CSNG
136	88	RND	160	A0	CDBL
137	89	SIN	161	A1	FIX
138	8A	LOG	162	A2	STICK
139	8B	EXP	163	A3	STRIG
140	8C	COS	164	A4	PDL
141	8D	TAN	165	A5	PAD
142	8E	ATN	166	A6	DSKF
143	8F	FRE	167	A7	FPOS
144	90	INP	168	A8	CVI
145	91	POS	169	A9	CVS
146	92	LEN	170	AA	CVD
147	93	STR$	171	AB	EOF
148	94	VAL	172	AC	LOC
149	95	ASC	173	AD	LOF
150	96	CHR$	174	AE	MKI$
151	97	PEEK	175	AF	MKS$
152	98	VPEEK	176	B0	MKD$

Zoals u ziet is bij de funktietokens nog ruimte. Met enige handigheid kan men, door het gebruik van de 'lege' token interessante dingen doen.

Listbeveiliging

We hebben een token gevonden, waarmee een eenvoudige listbescherming mogelijk is. Gaan we uit van de regel > 1 REM HALLO< en geef dan het volgende in:

```
POKE &H8005,255
POKE &H8006,52
```

Ga nu het programma listen.
Bij de start van het listen wordt automatisch een >CLS< uitgevoerd.
Als op regelmatige afstanden dergelijk gemanipuleerde >REM< regels in een programma voorkomen (en speciaal bij belangrijke plaatsen), is het zeer tijdrovend een samenhangende listing op het beeldscherm te verkrijgen. Deze regels mogen dan niet aangedaan worden omdat dit een syntax error tot gevolg heeft. Dergelijke regels kunnen oversprongen worden, bijv. met

```
RENUM
5 GOTO 20
```

Een programmakraker kan deze regels eenvoudig wissen. Dit geval behandelen we in de snuffelhoek (hoofdstuk 9).

Hier nog een toepassing:

Programma: DATA regelgenerator

De via het toetsenbord ingevoerde gegevens (hier woorden), moeten vast met >DATA< regels in het programma opgenomen worden. Zo zou bijv. een adresbestand in het programma geschreven kunnen worden, waarvan de gegevens niet (!) in de cassetterecorder, maar direkt in het programma opgeslagen zijn. De te veranderen >DATA< regel(s) moeten vooraf in het programma aanwezig zijn. Nemen we aan dat de >DATA< regelnummer 1000 heeft en 15 punten als provisorische gegevens bevat, dan funktioneert het volgende programma:

```
10 INPUT"woord";A$
20 L=LEN(A$)
30 IF L>15 THEN 10
40 VA=&H8001
50 ZN=PEEK(VA+2)+256*PEEK(VA+3)
```

```
60 IF ZN=1000 THEN 100
70 VA=PEEK(VA)+256*PEEK(VA+1)
80 IF VA=0 THEN PRINT"Regel 1000 ontbreekt":END
90 GOTO 50
100 AD=VA+4:REM Startadres der DATA regel 1000
110 IF PEEK(AD)<>&H84 THEN PRINT "Regel 1000 is geen data-regel":LIST 160
120 FOR I=1 TO L
130 IF PEEK(AD+I)<>46 THEN PRINT"te weinig punten in regel 1000":LIST 1000
140 POKE AD+I,ASC(MID$(A$,I,1))
150 NEXT
160 FOR I=L+1 TO 15
170 IF PEEK(AD+I)<>46 THEN 130
180 POKE AD+I,32:NEXT
1000 DATA ..................
```

Na afloop van het programma staat het door U ingegeven woord in >DATA< regel 1000. De regels tot 90 hebben we al besproken.
Regel 110 test het >DATA< token.
In de eerste lus (regels 120 tot 150) wordt het woord letter voor letter gepoked (regel 1140).
Vooraf wordt getest of op elke plaats nog een punt staat (regel 130).
De lus van de regels 160 tot 180 vult de rest van de regel met spaties.

Met de token zijn alle mogelijke bytes, groter dan &H80 besproken.
Bytes met waarden tussen 32 en 127 worden in het interne BASIC geheugen in ASCII kodes weergegeven, d.w.z. zij staan in de plaats van het betreffende teken. Speciale aandacht verdienen de kodes 0 tot 31.
Niet alle kodes vervullen een funktie, hier een samenvatting:

kode	funktie
11	kenteken voor octaalgetallen &O
12	kenteken voor hexadecimaalgetallen &H
17	cijfer 0
18	cijfer 1
19	cijfer 2
20	cijfer 3
21	cijfer 4
22	cijfer 5
23	cijfer 6
24	cijfer 7
25	cijfer 8
26	cijfer 9
27	getal 10
28	kenteken voor 2byte getallen (INT)
29	kenteken voor 4byte getallen (SGN)
31	kenteken voor 8byte getallen (DBL)

Al deze kodes zijn op een of andere manier met de opslag van getallen verbonden. Met de opslag van getallen en strings zullen we ons nu bezig gaan houden.

8.4 Getallenweergave

Er zijn 4 verschillende soorten van getallenweergave in de computer:

1	Integer	INT	- geheel getal
2	Single precision	SNG	- eenvoudige nauwkeurigheid
3	double precision	DBL	- dubbele nauwkeurigheid
4	strings	STR	- alfa-nummeriek

Integers

Eerst zullen we ons met de drie eerste soorten, d.w.z. het weergeven van getallen, bezighouden. De eerste soort, gehele getallen, zijn we al tegengekomen bij het opsplitsen in low byte en high byte. Bit nr. 7 van de high byte wordt als een voorteken gebruikt, nul betekent positief getal, 1=negatief getal.

De negatieve getallen worden komplementair voorgesteld. Hierdoor kunnen integer constanten of variabelen, voorgesteld in gehele getalwaarden van -32768 tot +32767 bevatten.

decimaal	binair	hex.
-32768	1000 0000 0000 0000	80 00
-32767	1000 0000 0000 0001	80 01
-32766	1000 0000 0000 0010	80 02
-32765	1000 0000 0000 0011	80 03
	.....	
2	1111 1111 1111 1110	FF FE
-1	1111 1111 1111 1111	FF FF
0	0000 0000 0000 0000	00 00
1	0000 0000 0000 0001	00 01
2	0000 0000 0000 0010	00 02
	.....	
32766	0111 1111 1111 1110	7F FE
32767	0111 1111 1111 1111	7F FF

Om intern de diverse voorstellingssoorten te kunnen onderscheiden is aan elke soort een type kengetal toegevoegd. Dit kengetal is gelijk aan het aantal bytes dat voor het opslaan van dat getal nodig is. Voor het opslaan van een integergetal zijn 2 bytes nodig, waardoor het typekengetal voor integerkonstanten of variabelen 2. Het kengetal voor de aktuele variabele is altijd op adres &HF663 opgeslagen.

Drijvende komma voorstelling

De drijvende komma getallen worden in het MSX operationele systeem op een, voor deze klasse computer, ongebruikelijke wijze opgeslagen. Het BCD format.

Bij het BCD (Binary Coded Decimal) format worden de afzonderlijke cijfers van een decimaal getal opgeslagen. Het voordeel is gelegen in een exact vastgelegd aantal plaatsen, dat verwerkt moet worden. Ook het waardebereik is bij het MSX systeem groter (10^-64 tot 10^62) dan bij de gebruikelijke opslagmethoden (10^-32 tot 10^31). Jammer genoeg kunnen getallen in het BCD format niet zo snel verwerkt worden. Om deze reden zijn er 2 vormen van nauwkeurigheid: SGN met 6 plaatsen en DBL met 14 plaatsen.

De weergave met eenvoudige nauwkeurigheid (SGN) zal in bijna alle gevallen met de vereiste nauwkeurigheid werken.

Hoe wordt nu een getal in BCD format opgeslagen?

Voordat we ons met de cijferopslag bezighouden, bespreken we eerst de exponentiële voorstelling van getallen, zoals die van zakrekenmachines bekend is. Moet een getal in de exponentiële schrijfwijze weergegeven worden, dan wordt eerst vastgesteld hoe vaak de basis van het decimaalsysteem, dus de tien, in het getal als faktor opgenomen is.
Vervolgens wordt het getal in 2 delen gesplitst. Het ene deel bevat de cijfers van het getal(mantisse), waarbij de komma altijd achter het eerste cijfer staat. Het tweede deel geeft nu aan, hoe vaak de tien in het oorspronkelijke getal past, of, met andere woorden, hoeveel plaatsen de komma in het eerste deel verschoven moet worden (exponent). Men schrijft dit als volgt:

27 = 2.7*10^2

3956 = 3.956*10^4

Met de duidelijke afspraak dat bij negatieve exponenten de komma in tegengestelde richting bewogen moet worden, kunnen nu alle getallen weergegeven worden:

0.21 = 2.1*10^-1

0.0051 = 5.1*10^-3

-9 = -9.0*10^0

Met deze afspraak is nu elk getal in een mantisse (cijferdeel voor de komma, slechts 1 plaats) en exponent te splitsen. De exponentiële schrijfwijze heeft het voordeel dat zeer grote en zeer kleine getallen met relatief weinig moeite opgeslagen kan worden:

0.00000000000000735 = 7.35*10^-15

6390000000000000 = 6.39*10^15

Bij het rekenen met deze getallen gelden vanzelfsprekend de gebruikelijke regels van de exponentieelberekeningen.

Optellen/aftrekken

Alleen getallen met gelijke exponenten kunnen opgeteld worden. Zijn de exponenten van de grondgetallen verschillend, dan wordt het getal met de kleinere exponent herschreven naar de grotere. Zijn de exponenten dan gelijk, dan kunnen eenvoudig de mantissen opgeteld resp. afgetrokken worden.
Voorbeeld:

```
      57  + 0.31
5.7*10*1  + 3.1*10^-1
5.7*10^1  + 0.031*10^1
  5.731   * 10^1
  57.31
```

Vermenigvuldigen/delen

Bij deze rekensoort worden eerst de mantissen vermenigvuldigd, resp. gedeeld, daarna de exponenten opgeteld, resp. afgetrokken.
Voorbeeld:

```
    0.13   * 20
1.3*10^-1  * 2.010^1
  1.3*2.0  10^1(-1+1)
    2.6    * 10^0
    2.6

     25    / 0.05
2.5*10^1   / 5.0*10^-2
   2.5/5   * 10^(1-(-2))
    0.5    * 10^3
      5    * 10^2
    500
```

Willen we nu deze handeling door de computer laten verrichten dan komt het er op aan de getallen in dit format op te slaan. Bij BCD formats wordt elk cijfer van de mantisse afzonderlijk opgeslagen.
Een cijfer in het decimale stelsel heeft waarden van 0 tot 9=&B1001.
We hebben dus 4 bits nodig voor de opslag van een cijfer. Met een byte

(8bits) kunnen derhalve 2 cijfers gekodeerd worden, waarbij de high nibble (bit 4 tot 7) het eerste, en de low nibble (bit 0 tot 3) het tweede cijfer voorstelt.
Voorbeeld:

Decimaal : 27
high nibble : 2 = &B10
low nibble : 7 = &B111
BCD format : &B0010 0111= &H27 =39

Wegens de bijzondere eigenschappen van het hexadecimaal systeem, dat 4 bits met een hexadecimaal cijfer overeenkomen, is de waarde van een hex-getal altijd het geïnterpreteerde decimaalgetal van de in BCD gekodeerde waarde.

De BCD waarde van 63 is &H63=&B0110 &B0011 = 99

Het aantal cijfers dat opgeslagen kan worden is theoretisch onbegrensd. Ze wordt bepaald door het aantal vast te leggen bytes. Bij de weergave van getallen met eenvoudige nauwkeurigheid (SGN) is het aantal bytes dat gebruikt wordt door de cijfers van de mantisse 3. Hierdoor hebben de getallen van dit type een nauwkeurigheid van 6 cijfers, namelijk 2 per byte.
Het getal 123794 wordt in BCD format als een opeenvolging van 3 bytes opgeslagen:
&H12,&H37,&H94

Wordt de dubbele nauwkeurigheid gekozen, dan staan 7 bytes ter beschikking, wat een aantal van 14 cijfers toelaat.
Het is nu duidelijk hoe de mantisse van een drijvende komma in de exponentiële vorm intern opgeslagen wordt.
Bij de kodering van de exponenten is een andere weg bewandeld. Hij wordt direkt door de waarde van een byte weergegeven. Overigens dienen wel gegevens over:
voorteken van de mantisse en
voorteken van de exponent
in dit byte ondergebracht worden. Bit 7 kenmerkt op de gebruikelijke wijze het voorteken van het getal (mantisse)
0 = positief
1 = negatief

De resterende 7 bits stellen de waarde van de exponent voor, waarbij &H41 aan de reële waarde werd toegevoegd:

waarde bit 6	waarde exponent
&H7F	&H3E = 62
&H7E	&H3D = 61
...	...
...	...
&H42	&H01 = 1
&H41	&H00 = 0
&H40	-&H01 = -1
...	...
...	...
&H02	-&H3F = -63
&H01	-&H40 = -64
&H00	betekent getal = 0

De exponent kan aldus waarden aannemen van -64 tot +62, een getallengebied dat voor elke berekening voldoende is.
Volgens afspraak wordt bij een getal met de waarde 0, de exponent op 0 geplaatst.

Laat ons deze vorm van weergeven eens testen met een BASIC programma.
Daarbij wordt de instruktie >VARPTR< gebruikt. Deze funktie geeft ons het adres in het geheugen, waar de waarde van de variabele, in de bovenomschreven vorm gekodeerd staat.
Voor het beheer van de variabelen volgt, direkt na het programma, een gebied waar elke gebruikte variabele met naam en waarde is opgeslagen.
In het adres, dat de >VARPTR< funktie aangeeft, staat de eerste byte van het gekodeerde getal, dat volgens afspraak de exponent is. Daarna volgen, naargelang het type, de 3 (SNG) resp. 7 (DBL) bytes die de cijfers voorstellen.

```
10 X!=43546!
20 AD=VARPTR(X!)
30 PRINT HEX$(PEEK(AD))
40 FOR I=AD+1 TO AD+3
```

```
50 PRINT RIGHT$("0"+HEX$(PEEK(I)),2);
60 NEXT
```

```
10 X#=.012342349#
20 AD=VARPTR(X#)
30 PRINT HEX$(PEEK(AD))
40 FOR I=AD+1 TO AD+6
50 PRINT RIGHT$("0"+HEX$(PEEK(I)),2);
60 NEXT
```

Geef nu verschillende getallen voor X in en let dan op de uitvoer van het programma.

Nadat we de getallen besproken hebben, kijken we nog eens naar een ander, volkomen verschillend, variabeletype: de stringvariabele.

8.5 Strings

In een string worden alfanummerieke gegevens, d.w.z. tekens zoals letters en/of cijfers opgeslagen. Aan elk teken is dan een kode toegewezen. De MSX tekenkode komt nagenoeg overeen met de kode's van 0 tot 127 van de zg. American Standard Code for Information Interchanche (ASCII).

Voor het opslaan van een teken is dan ook precies een byte nodig. Een string (Eng.string=snoer, ketting) is nu een ketting van opeenvolgende tekenkode's. Omdat een teken overeenkomt met een byte, wordt een string als een reeks opeenvolgende adressen in RAM opgeslagen. Om aan een bepaalde variabele een string toe te wijzen, zijn 2 informaties noodzakelijk die de zg. string descriptor vormen.

1 Het adres van de eerste geheugenplaats, die de eerste tekenkode van de string bevat.
2 De lengte van de string, dus het aantal bytes dat de tekensketen vormt.

Deze beide gegevens worden, samen met de variabelenaam, in het BASIC

variabelengebied opgeslagen. De tekensketen staat ergens anders, hetzij in het programma zelf, hetzij in een daarvoor gereserveerd stringgebied. De grootte van dit gebied kan door een CLEAR instruktie vastgelegd worden.
In BASIC kunnen we weer de >VARPTR< instruktie gebruiken om het adres van de stringdescriptor van een variabele te lezen.
De stringdescriptor bestaat uit 3 bytes:

byte 1 : Lengte van de string
byte 2 en 3 : Startadres van de string

De >VARPTR< funktie voert het adres van de eerste byte van de stringdescriptor uit.
Probeert U maar eens het volgende programma:

```
10 X$="TTTekenketEEEn"
20 AD=VARPTR(X$)
30 LA=PEEK(AD)
40 ST=PEEK(AD+1)+256*PEEK(AD+2)
50 FOR I=ST TO ST+LA-1
60 PRINT CHR$(PEEK(I));
70 NEXT
```

8.6 BASIC-Variabelentabel

Tenslotte bespreken we nog in het kort hoe de BASIC variabelen, met behulp van de reeds aangegeven variabelenblad, dat na het programma komt, wordt beheerd.
In het adres &HF6C2/3 staat het startadres van de variabelentabel.
Voor de variabelentabel geldt het volgende schema:

byte 1 : Typekengetal (gelijktijdig aantal bytes)
byte 2 : variabelennaam
byte 3 : ASCII kode weergave
vanaf
byte 4 : de waarde van de variabelen

Individueel geldt het volgende:

type INT

02 kenteken INT
·· variabelennaam
·· in ASCII kode low byte
high byte
type SGN

04 kenteken SGN
·· naam
··
4 bytes voor de waarde

type DBL

08 kenteken DBL
·· naam
··
8 bytes voor de waarde

type string

03 kenteken string
byte 1: lengte
byte 2 en 3: adres

Hiermee kunnen we nu een eenvoudig programma schrijven dat alle gebruikte variabelen uitvoert.

Dit programma kunt U aan Uw eigen programma's toevoegen en voor testdoeleinden oproepen. Een uitbreiding van het programma, dat ook de waarde van de variabelen uitvoert, zal geen problemen op mogen leveren.

```
1000 REM DUMP
1010 AD=PEEK(&HF6C2)+256*PEEK(&HF6C3):REM start variabelentabel
1020 AE=PEEK(&HF6C4)+256*PEEK(&HF6C5):REM einde variabelentabel
10030 TY=PEEK(AD):REM kenmerk
10040 REM naam tonen
```

```
10050 PRINT CHR$(PEEK(AD+1));CHR$(PEEK(AD+2));
10060 REM type tonen
10070 IF TY=2 THEN PRINT"%";ELSE IF TY=3 THEN PRINT"$";ELSE
IF TY=4 THEN PRINT"!":ELSE PRINT"#";
10080 PRINT
10090 AD=AD+3:REM start van waarde
10100 AD=AD+TY:REM start van volgende var.
10110 IF AD>AE THEN END ELSE 10030
```

9: De snuffelhoek

9.1 De DEEK instruktie

In enige BASIC 'dialekten' bestaat de instruktie >DEEK<. >DEEK< leest de 2byte waarde van twee opeenvolgende geheugenplaatsen. Zij komt overeen met de instrukties >PEEK(adres)+256*PEEK(adres+1).
De >DEEK< instruktie moet nu geïmplementeerd worden in MSX BASIC met behulp van de >USR< instruktie. Het adres van de low byte van de te lezen geheugenplaats moet overgedragen worden. Bij het begin van het programma wordt getest, of de overgedragen parameter van het juiste type is. Als dit niet het geval is, dan moet naar de 'type mismatch error' routine gesprongen worden.
We laden daartoe het E register met het nummer van de betreffende fout en naar &H406F (routine voor uitvoer van fouten) afgebogen.

```
CP2;        type 2
JR Z,OK;    Ja, dan Ok.
LD E,13;    Nee, dan
JP&H406F;   type mismatch error.
```

Uiteraard bestaat voor elke foutmelding ook een eigen inspringadres, die het laden van het E register verzorgt. Voor de 'Type mismatch error' is dit adres &H406D. Daarmee wordt het programma vereenvoudigd tot:

```
CP2;              type 2 ?
JP NZ,&H406D;     nee, dan 'type mismatch error'
```

Op deze wijze heeft de oplossing voor type-controle een nadeel. Geeft de gebruiker via het BASIC een getal van juiste grootte, maar van het verkeerde type, in, bijv. het getal 1000 kan zowel integer als SGN of DBL type opgeslagen worden en zou tot een type mismatch error leiden.
Het is echter mogelijk een dergelijk getal in een integer getal om te zetten. In BASIC vervult de >CINT< funktie deze rol.
Als we, bij het begin van het machinetaalprogramma, de >CINT< routine oproepen, dan wordt de verandering automatisch aangebracht. Een fout, zoals hierboven beschreven, treedt dan alleen nog op bij het invoeren van strings. Bovendien test de >CINT< routine of het ingegeven getal van de

juiste grootte is, anders wordt de melding 'overflow' gegeven.
Heeft u variabelen van een ander type nodig, dan gebruikt u de routine CSNG (&H2FB2) en CDBL (&H303A). Schenk ook aandacht aan de tabellen aan het einde van dit boek, waarin alle behandelde (en nog andere) systeemroutines aangehaald worden.

Nu voor de uitvoering van de >DEEK< instruktie.

```
10  ' CALL &H"F8A; CINT konverteren naar INT
20  ' LD HL, (&HF7F8); overgedragen parameterwaarde van
    adres
30  ' LD E, (HL); low byte
40  ' INC HL
50  ' LD D, (HL); high byte
```

In regel 20 wordt de over te dragen parameterwaarde, dus het adres, dat van de tweede byte af gelezen moet worden, door de overdrachtadressen &HF7F8 en &HF7F9 gelezen. In de regels 30 tot 50 wordt de 2bytewaarde, die op deze adressen staat, in de DE registers geladen.
Nu moeten we de verkregen waarde weer in BASIC terugplaatsen. Hiervoor moet de overdrachtswaarde in de adressen &HF7F8/9 geladen worden.
Bovendien moet de accu het typekengetal bevatten en HL moet met adressen &HF7F6 geladen worden.

```
60  ' LD (&HF7F8), DE; resultaat van 'DEEK'
70  ' LD HL, &HF7F6
80  ' RET
```

Assembleer dit programma en probeer de nieuwe funktie uit.

```
PRINT USR1 (2)
```

levert 370, hetzelfde resultaat als:

```
PRINT PEEK(2) +256*PEEK(3)
```

Dat betekent dus dat vanaf adres 2 van de 2bytewaarde 370 of &H207 staat.

```
F000            10                      ; DEEK instruktie
F000            20                      ; geeft 2 byte waarde vanaf
                                          overgedragen adres
F000 CD8A2F     30 CALL &H2F8A          ; CINT
F003 2AF8F7     40 LD HL, (&HF7F8       ; overgedragen parameter-
                                          waarde=adres
F006 5E         50 LD E, (HL)           ; low byte lezen
F007 23         60 INC HL
F008 56         70 LD D, (HL)           ; high byte lezen
F009 ED53F8F7   80 LD (&HF7F8), DE      ; resultaat van DEEK
F00D 21F6F7     90 LD HL,&HF7F6
F010 C9        100 RET
```

Programma: deek
Start: &HF000 Einde &HF 010
Lengte: &H11 bytes
Fouten: 0

```
10 REM DEEK Instruktie
20 CLEAR 200,&HEFFF
30 FOR I=&HF000 TO &HF010:READ A$
40 POKE I,VAL("&H"+A$):NEXT
50 DEFUSR1=&HF000
60 DATA CD,8A,2F,2A,F8,F7,5E,23
70 DATA 56,ED,53,F8,F7,21,F6,F7,C9
```

9. 2 De UPPER instruktie

Met behulp van de machinetaal is het mogelijk de, in enige BASIC dialecten bekende, >UPPER< instruktie te realiseren. De >UPPER< instruktie verandert alle kleine letters van een string in hoofdletters.

```
10 CLEAR 200,&HEFFF:MAXFILES=1
20 FOR I=&HF000 TO &HF021:READ A$
30 W=VAL("&H"+A$):S=S+W
40 POKE I,W:NEXT
50 IF S<>3814 THEN PRINT "fouten in DATAs":END
60 PRINT"Alles is klaar !"
```

```
70 DEFUSR1=&HF000
80 DATA FE,03,C2,6D,40,1A,B7,C8
90 DATA 47,62,6B,23,7E,23,66,6F
100 DATA 7E,FE,7B,30,06,FE,61,38
110 DATA 02,E6,DF,77,23,10,F1,3E
120 DATA 03,C9
```

```
F000            10                      ; UPPER INSTRUKTIE
F000            20      ORG &HF000
F000            30TYP   EQU &H406D      ; type mismatch error
F000 FE03       40      CP 3            ; stringvariabele ?
F002 C26D40     50      JP NZ, typerr   ; nee, dan type mismatch
                                          error
F005 1A         60      LD A, (DE)      ; lengte van de string
F006 B7         70      OR A            ; Z flag plaatsen als A0
F007 C8         80      RET Z           ; klaar indien lengte 0
F008 47         90      LD B,A          ; lengte als lusteller
F009 62        100      LD H,D          ; descriptoradres
F00A 6B        110      LD L,E          ; naar HL
F00B 23        120      INC HL
F00C 7E        130      LD A, (HL)      ; adres
F00D 23        140      INC HL          ; van
F00E 66        150      LD H, (HL)      ; stringbegin
F00F 6F        160      LD L,A          ; naar HL
F010 7E        170LUS   LD A,(HL)       ; indien ASCII kode
F011 FE7B      180      CP 123          ; >= als ASCII("z")+1
F013 30 FE     190      JR NC,OK        ; dan niet wijzigen
F015 FE61      200      CP 97           ; indien <ASCII("a")?
F017 38FE      210      JR C,OK         ; dan niet veranderen
F019 E6DF      220      AND &B11011111 ; veranderen van
    ****regel 190: OK=&HF01B Offset 6
    ****regel A10: OK=&HF01B Offset 2
F01B 77        230 OK   LD (HL),A       ; ASCII weer opslaan
F01C 23        240      INC HL          ; adres volgende kode
F01D 10F1      250      DJNZ LUS        ; tot stringeinde
F01F 3E03      260      LD A,3          ; typekengetal van string
F021           270                      ; DE bevat oud descriptor-
                                          adres
F021 C9        280      RET.
```

Programma: UPPER
Start: &HF000 einde: &HF021
Lengte 22 bytes
Fouten: 0
Variabelentabel:
TYP 406D LUS F010
OK F01B

9.3 De slot PEEK instruktie

Hier volgt nu de theorie over de slot PEEK instruktie van de monitor. Bij de aansluitende tekst gaat het over een uittreksel uit het "machinetaalboek voor de MSX" van DATA BECKER.

Een BASIC programma wordt, na de invoer door de programmeur, in een tussenkode vertaald. De instruktiewoorden worden niet als zodanig opgeslagen, maar door een verkorte kode (1 byte lang >128) vervangen, het zg. token. Herkent de interpreter het token, dan springt hij in de aan dat token toegewezen routine. Daar worden dan de evt. achter de instruktie staande parameters ingelezen en op juistheid getest. Hier zullen we ons nu wat nader in verdiepen.

Nemen we als voorbeeld eens de >POKE< instruktie. Na de >POKE< instruktie staan 2, door een komma gescheiden parameters, het adres en de waarde. Nemen we eens aan dat het token voor >POKE< (=152) herkend werd en dat het programma bij de >POKE< routine naar &H5423 afbuigt. Principieel wijst, bij de BASIC interpretatie, het HL register altijd naar de aktueel bewerkte plaats. HL dient dan als BASIC programma pointer. Er moet altijd zekerheid bestaan dat deze pointer niet verloren gaat, omdat in dat geval de interpretatie niet korrekt voortgezet wordt.
Voor het inlezen van bovengenoemde parameters, zijn er in de interpreter speciale routines, die we natuurlijk ook kunnen gebruiken.
De bij de >POKE< instruktie opgeroepen routine heet GETADR. Zij leest een integerparameter. Bij het oproepen van deze routine moet HL met de aktuele BASIC pointer geladen worden. Deze analyseert dan alle uitdrukkingen, zoals 1237, maar ook komplexe uitdrukkingen als

ABS(INT(VAL(RIGHT$(STRING$(4,"0")+HEX$(x),4)))), op dezelfde manier. De verkregen 2 bytewaarde wordt in het DE register teruggegeven.
De BASIC pointer HL wijst na het einde van de routine, in ons geval, op de volgende komma. De GETADR routine heeft het adres &H542F.
Vervolgens wordt getest of nu een komma volgt. Ook hiervoor bestaat een systeemroutine: de RST &H08.
Deze restart dient om te testen op een willekeurig teken.
De ASCII kode van het teken moet met DB direkt achter de RST &H08 instruktie staan. Voor de >POKE< instruktie dus:

```
RST &H08  ; test op ","
DB &H2C   ; kode voor ","
```

Wordt dit teken niet gevonden, dan wordt een syntax error gegeven, anders volgt een terugsprong.
De routine, die dan opgeroepen wordt om de 8bit waarde in te lezen, noemen we GETBYT. De 1byte waarde wordt aan de accu teruggegeven. Nadat deze routine is afgewerkt, wijst het HL register als BASIC pointer op het einde van de instruktie.
Met behulp van deze routines is het mogelijk de >USR< instruktie zodanig uit te breiden dat een willekeurig aantal parameters kunnen worden meegegeven. Het volgende programma gebruikt ze om een gemodificeerde >PEEK< funktie op te wekken, waarbij aan het adres, dat gelezen moet worden, het slotnummer kan worden toegevoegd.
Hieruit ontstaat de mogelijkheid uitbreidingen of overlapte ingebouwde modules uit te lezen.
De eigenlijke uitvoering wordt door de routine vanaf &H000C veroorzaakt (slot RD). Wordt in de accu het slotnummer, en in het HL register het adres overgedragen, dan geeft deze routine in de accu de waarde van het gelezen byte terug.
De BASIC pointer kan in een zelfgemaakt programma door een dubbele 'POP' uit de stack gelezen worden. Belangrijk is dat de stack, en speciaal het terugkeeradres, steeds juist geconstrueerd wordt.
Hier volgt dan de assemblerlisting:

```
F000            10                          ; Slot PEEK
F000            20                          ; format USR(adres)
F000            30          ORG &HF000
F000            40 ILLQUA   EQU&H475A
F000            50 CINT     EQU &H2F8A
F000            60 GETBYT   EQU &H521C
F000            70 SLOTRD   EQU &H000C
F000            80
F000 CD8A2F     90          CALL CINT
                                        ; converteren naar INT en
                                          waarde
F003 EB         100         EX DE,HL    ; laden naar HL
F004 C1         110         POP BC      ; terugspringadres
F005 E1         120         POP HL      ; BASIC-pointer
F006 E5         130         PUSH HL     ; stack herstellen
F007 C5         140         PUSH BC
F008 D5         150         PUSH DE     ; waarde adressen
F009 CF         160         RST &HO8    ; test op
F00A 2C         170         DB &H2C     ; ASCII (",")
F00B CD1C52     180         CALL GETBYT ; haalt slotnummer
F00EFE03        190         CP 3        ; slot nummer >=3?
F010 D25A47     200         JP NC,ILLQUA ; ja, dan illegal quantity
F013 E3         210         EX (SP),HL  ; te lezen adres verwisselen
                                          met BASIC-pointer
F014 CD0C00     220         CALL SLOTRD ; slot read
F017 CDCF$F     230         CALL &H$FCF ; accu in FAC laden
F01A E1         240         POP HL      ; BASIC-pointer
F01B C1         250         POP BC      ; terugspringen
F01C D1         260         POP DE      ; oude BASIC-pointer
F01D E5         270         PUSH HL     ; nieuwe BASIC-pointer
F01E C5         280         PUSH BC     ; terugspringen
F01F 21F6F7     290         LD HL,&HF7F6 ; startadres FAC
                                          bevat reeds
                                          type 2 (zie rout. vanaf
                                          &H4FCF)
F022            300
F022 C9         310         RET
```

Programma: sltrd
Start: &HF000 Einde: &HF022

Lengte: &H23 bytes
Fouten: 0
Variabelentabel: ILLQUA 475A CINT 2FBA
GETBYT 521C SLOTRD 000C

9.4 De CLEAR instruktie

Bij onze MSX computer funktioneerde de CLEAR instruktie niet feilloos:

```
10 CLEAR 200,&HF000
20 INPUT B$
30 FOR I=32 TO 110
40 FOR J=32 TO Ii
50 A$=A$+CHR$(J)
60 NEXT J
70 PRINT B$
80 A$=""
90 NEXT I
100 END
```

Dit programma moet, na het ingeven van een woord van minstens 2 tekens steeds het woord uitvoeren (regel 70). Zouden bij Uw computer moeilijkheden optreden (bijv. het woord 'Boom' wordt plotseling 'mmmm') dan dient U achter elke >CLEAR< instruktie een >MAXFILES< instruktie te plaatsen, dus:

```
10 CLEAR 200, &HF000: MAXFILES=1
```

Nu dient alles korrekt te verlopen.
De fout is gelegen in het niet korrekt plaatsen van de wijzer voor de BASIC stringvariabelen. Met >MAXFILES< wordt dit gecorrigeerd.

9.5 Listbeveiliging

Een eenvoudige listbeveiliging verkrijgt men door de blokkade van de listpatch.

```
POKE &HFF89,&HDD
POKE &HFF8A,&HE1
```

De kode POP IX betekent, dat de listroutine simpelweg uit de stack is gehaald. Bij de volgende RET instruktie wordt direkt in de interpreterlus teruggesprongen.
Met POKE &HFF89,HC9 wordt de listbescherming opgeheven.

Een andere listbeveiliging, die speciaal in verbinding met die van hoofdstuk 8 werkzaam is, is de test op de lengte van het programma. Wordt getracht de regel, die bij het listen steeds de CLS aanzet, dan verandert de lengte van het programma, hetgeen vastgesteld kan worden. Daaruit volgt de zelfvernietiging van het programma. U heeft daarvoor de volgende regel nodig:

```
1 A=FRE(0): IF A<>&H1111 THEN END
```

De korrekte waarde voor de lengte, die op de plaats van &H1111 gezet wordt, kan door een proefrun verkregen worden.: RUN ingeven en PRINT HEX$(A)

De verkregen waarde wordt dan in &H1111 geplaatst.
Verandert U nu iets in de volgende regels van het programma dan wordt dit vastgesteld en het programma direkt beëindigd.
Nog effectiever wordt de beveiliging als de END instruktie wordt vervangen door NEW (of zelfs door een RESET sprong)

9.6 Input zonder ?

Als de ROM in het RAM gekopieerd zou worden, was dit probleem opgelost. Er bestaat ook een mogelijkheid bij ingeschakelde ROM het vraagteken met een patch te overschrijven.

```
10 FOR I=&HFDE0 TO &HFDE3
20 READ A
30 POKE I,A
40 NEXT
50 DATA &H21,&HD2,&H23,&HE3
```

Assemblerlisting:

```
FDE0 210223     LD HL,&H23D2      ; input patch &H23D2 is
                                    nieuw sprongadres
FDE3 E3         EX (SP),HL        ; op stack
FDE4 C9         RET
```

De oude toestand kan met

```
POKE &HFDE0,&HC9
```

hersteld worden.

10. Programma's

10.1 De menugenerator

De menugenerator stelt U in staat programma's, waarin met veel menu's gewerkt wordt, eenvoudiger vorm te geven.
De eigenlijke menugenerator is in de regels 50 tot 480 en regels 10000 tot einde opgenomen.
De DATA regels aan het einde van het programma kunt U naar behoefte veranderen. Het uitroepteken (!) geeft de laatste DATA regel aan.
De eerste DATA regel (10270) moet altijd Hoofdmenu zijn, gevolgd door regel 10280 DATA 0. Vanaf deze regel kunnen Uw eigen menupunten ingegeven worden. De volgorde is:

Na DATA 1 staat het sub-menu voor het eerste punt van het hoofdmenu (hier: printen).
Na DATA 2 staat het sub-menu enz.
Is elk punt van het hoofdmenu aan een sub-menu toegewezen, dan volgt het sub-menu (van de tweede orde) van het eerste sub-menu. (hier: tot het format behorende DATA 6 en volgend).

Het bijzondere is dat U niet hoeft te letten op de lengte van de afzonderlijke menu's. De invoer op de hierboven beschreven wijze is voldoende.

Als U Uw eigen menu ingegeven heeft, dient U nog de daarbij behorende routines te schrijven. Daartoe dienen de regels 1000-9999.
De integratie van de menugenerator is, schematisch gezien, altijd hetzelfde (zie regels 1010-1040). Op de doelregels in >GOSUB< instrukties staat dan, ofwel de oproep voor het volgende menu (exact volgens schema), of een uitvoerende routine. Deze moet dan met RETURN afgesloten worden.
Het beheer van het programma berust op het principe van een wijzerveld MD (MP,I). MD(,) bevat informatie over:

0 - terug: nummer van het laatste menu
1 - next: nummer van het volgende menu
2 - omvang: aantal inputs in menu
3 - default: laatste, in dit menu gekozen, waarde.

Om de points telkens op te roepen, wordt de menupointer (MP) en het kengetal van de informatie (0-3) voor MD als wijzer gebruikt.
De menustring en de wijzervelden worden beiden in de regels 10200 tot 10260 samengesteld.

```
10 REM Menugenerator
20 CLEAR 200,&HEFFF:MAXFILES=0
30 KEY OFF:SCREEN 0:WIDTH 38
40 DEFINT A-Z
50 GOSUB 10000 : REM Init
60 GOSUB 1000 : REM Hoofdmenu
70 CLS: END
170 '
180 REM
190 '
200 REM kader
210 CLS:PRINT"Menugenerator  ";M$(MP)
220 PRINTSTRING$(38,"-")
230 LOCATE 0,21:PRINTSTRING$(38,"-");
240 PRINT" >RETURN< keuze maken"
250 NU=MD(MP,RU):IF NU>1 THEN NU=1
260 PRINT"of >BS < voor ";EX$(NU+1)
270 RETURN
290 '
300 REM menu opbouwen
310 GOSUB 200: REM kader
320 WA=MD(MP,DL):
330 LOCATE 0,4
340 FOR I=1 TO MD(MP,GR):IF I=WA THEN POKE &HFDA4,&HC3:REM
menupunt invers
350 PRINTI;:POKE IN,AU:PRINT" - ";:IF I=WA THEN POKE IN, AN
360 PRINT M$(I+MD(MP,NX)-1):POKE IN,AU
370 NEXT
380 PRINT:PRINT:PRINT"keuze :";:POKE IN,AN
390 PRINTWA;:POKE IN,AU
400 LOCATE POS(0)-3
```

```
410 EI$=INKEY$:IF EI$="" THEN 410
420 IF EI$=ZR$ THEN RETURN
430 IF EI$=CHR$(13) THEN MD(MP,DL)=WA:MP=MD(MP,NX)+WA-1:RETURN
440 IF EI$=CHRs(28) OR EI$=CHR$(31) OR EI$=CHRPRINT$(32)
THEN WA=WA+1+(WA=MD(MP,GR))*MD(MP,GR):GOTO 480
450 IF EI$=CHR$(29) OR EI$=CHR$(30) THEN WA=WA-1-(WA=1)*
MD(MP,GR):GOTO 480
460 N=VAL(EI$):IF (N<1) OR (N>MD(MP,GR)) THEN 410
470 WA=N
480 POKE IN,AN:PRINTWA;:POKE IN,AU:GOTO330
990 '
1000 REM hoofdmenu
1010 GOSUB 300:REM Menu tonen
1020 IF EI$=ZR$ THEN RETURN
1030 ON WA GOSUB 1050,1310,1400,1490,6000
1040 GOTO 1010
1050 REM
1060 GOSUB 300
1070 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1080 ON WA GOSUB 1100,1170,1240
1090 GOTO 1050
1100 REM
1110 GOSUB 300
1120 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1130 ON WA GOSUB 1140,2160,1150:RETURN
1140 REM
1150 REM
1160 RETURN
1170 REM
1180 GOSUB 300
1190 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1200 ON WA GOSUB 1210,2260,1220:RETURN
1210 REM
1220 REM
1230 RETURN
1240 REM
```

```
1250 GOSUB 300
1260 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1270 ON WA GOSUB 1280,2360,1290:RETURN
1280 REM
1290 REM
1300 RETURN
1310 REM
1320 GOSUB 300
1330 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1340 ON WA GOSUB 1360,3200,3300
1350 GOTO 1310
1360 REM
1370 GOSUB 300
1380 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1390 ON WA GOSUB 3150,3160,3170:RETURN
1400 REM
1410 GOSUB 300
1420 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1430 ON WA GOSUB 1450,4200,4300
1440 GOTO 1400
1450 REM
1460 GOSUB 300
1470 IF EI$=ZR$ THEN MP=MD(MP,RU):RETURN
1480 ON WA GOSUB 4150,4160,4170:RETURN
1490 ' enz.
1500 ' alle regels van 1000 -
1510 ' 10000 zijn voorbeeld;
1520 ' binnen deze regels
1530 ' moet het eigenlijke
1540 ' programma gemaakt zijn;
1550 ' Deze opbouw kan
1560 ' dienst doen, ook
1570 ' in eigen programma's,
1580 ' voor menuverwerking
10000 REM Init
10010 REM Inverse tekenset
```

```
10020 BA=BASE(2)+128*8
10030 FOR I=BA TO BA+128*8-1
10040 VPOKE I,VPEEK(I-128*8) XOR 255:NEXT
10050 REM Invers Patch laden
10060 FOR I=&HF370 TO &HF37A:READ A$
10070 POKE I,VAL("&H"+A$):NEXT
10080 DATA E1,F1,FE,20,38,02,F6,80,F5,E5,C9
10090 POKE &HFDA5,&H70:POKE &HFDA6,&HF3
10100 IN=&HFDA4:AN=&HC3:AU=&HC9:Rem vers. voor inverse aan/uit
10110 ZR$=CHR$(8)
10120 FOR I=0 TO 2:READ EX$(I):NEXT
10130 DATA einde programma
10140 DATA hoofdmenu
10150 DATA laatste menu
10160 DIM M$(100),MC(100,3)
10170 RU=0:NX=1:REM md(i,u) point naar vorig menu,
md(i,nx) op het volgende
10180 GR=2:DL=3:REM md(i,gr) grootte menu, md(i,dl) laatst gekozen
punt (standaard)
10190 N=1:I=-1:REM teller
10200 I=I+1:READ M$(I):REM Menupunt lezen
10210 MD(I,RU)=N:MD(I,DL)=1:REM laatste menu
10220 IF M$(I)="!" THEN I=I-1:MD(MD(I,RU),GR)=I-MD(MD(I,RU),NX)+1:
RETURN:REM einde menu
10230 IF LEN(M$(I))>2 THEN 10200
10240 N=VAL(M$(I)):MD(N,NX)=I:I=I-1
10250 IF MD(I,RUCK)>=0 THEN MD(MD(I,RUCK),GR)=I-MD(MD(I,RU),NX)+1
10260 GOTO 10200
10270 DATA hoofdmenu
10280 DATA 0
10290 DATA printen
10300 DATA tonen
10310 DATA edit
10320 DATA zoeken
10330 DATA dienst
10340 DATA 1
```

```
10350 DATA formatteren
10360 DATA pag.
10370 DATA normaal
10380 DATA 2
10390 DATA geformatteerd
10400 DATA 40 tekens
10410 DATA 32 tekens
10420 DATA 3
10430 DATA blok
10440 DATA wissen
10450 DATA 4
10460 DATA zoekwoord
10470 DATA aant. pagina's
10480 DATA 5
10490 DATA floppy
10500 DATA cassette
10510 DATA 6
10520 DATA standaardformat
10540 DATA zomaar printen
10550 DATA !
```

10.2 Minitextomat

Een van de belangrijkste toepassingen van de computer is de tekstverwerking. Om U een indruk te geven van de mogelijkheden van de MSX computer op het terrein van de personal computer, hebben we een tekstprogramma voor de MSX geschreven.
De bediening van het programma is in principe gelijk aan het werken met het BASIC beeldscherm (bij het programmeren), d.w.z. >INS<, >DEL< en >BS< werken bij BASIC altijd slechts op een regel, in het tekstprogramma op een gehele alinea.
Een alinea wordt door het gelijktijdig indrukken van >SHIFT< en >RETURN< afgesloten. Een kleine pijl geeft het alinea-einde aan.
Moeten spaties in de tekst verschijnen dan moeten die eveneens met >SHIFT/ RETURN< afgesloten worden. Deze toetsencombinatie wist tevens het resterende deel van een regel.

De cursortoetsen en >RETURN< (zonder >SHIFT<) dienen om vrij in de tekst te bewegen. Met >CTRL< + cursor springt U met 1 scherm tegelijk door de tekst.

Aan het einde van de tekst moet altijd >GRAPH+p< ingegeven worden. (moet apart op 1 regel staan). Dit teken, dat op de cursor lijkt, beëindigt de tekstuitvoer bij het printen.

Met >ESC< verlaat U het programma. Als U bij het herstarten van het programma bij de wisfunktie 'nee' ingeeft, blijft de oude tekst intact. Voorts kunnen via de funktietoetsen de routines printen/load/save opgeroepen worden.

Met save/load kunnen teksten opgeslagen/geladen worden. Bij het printen wordt het format, dat de paginaopbouw bepaalt, met LR, AS, OB, AZ vastgelegd (regels 2510-2520). Desgewenst kunnen de waarden veranderd worden.

Aansluitend aan de listing vindt U de assemblerlisting van de in het tekstprogramma gebruikte machinetaalroutines.

```
10 ' Minitextomat
20 CLEAR 200,&HE6FF:REM geheugen reseveren
30 GOSUB 2000:REM Init
40 INPUT "geheugen wissen?";A$
50 IF (ASC(A$)OR2^5)=ASC("j") THEN X=USR7(1):CLS ELSE X=USR1(TB)
60 A$=""
70 LOCATE 0,0,1
80 '
90 A$=INKEY$:IF A$="" THEN 90
100 GOSUB 120
110 GOTO 80
120 REM
130 AF=0:REM Scrollflag
140 IF A$>=" " THEN 290:REM standaard invoer
150 A=ASC(A$)
160 IF A>26 THEN ON A-26 GOTO 230,390,430,510,580:REM Escape en
Cursor
```

```
170 IF A=13 THEN 650:REM Return
180 IF A=11 THEN LOCATE 0,0:RETURN:REM Home
190 IF A=12 THEN X=USR1(TB):ZU=0:LOCATE 0,0,1:RETURN:REM Shift Home
200 IF A=8 THEN A$=CHR$(29):GOSUB 430:X=USR3(TB+(ZU+CSRL IN)*SM+POS
(0)+1):X=USR1(TB+ZU*SM):RETURN:REM Back Space
210 IF A=18 THEN POKE &HFCAA,1+(PEEK(&HFCAA)=1):LOCATE,1:RETURN:REM
Insert toets
220 RETURN: REM herhaal invoer
230 REM einde
240 CLS:LOCATE 0,0,0
250 ON ERROR GOTO 0:REM On Error uitschakelen
260 POKE &HFCAA,0:REM Insert Flag resetten
280 END
290 REM
300 IF ASC(A$)=127 THEN X=USR3(TB+(ZU+CSRLIN)*SM+POS(0)+1):X=USR1(
TB+ZU*SM):RETURN: REM delete
310 IF POS(0)=SM-1 AND CSRLIN=23-ZO THEN AF=-1
320 IF PEEK(&HFCAA)=1 THEN GOSUB 750
330 POKE TB+(ZU+CSRLIN)*SM+POS(0),ASC(A$)
340 PRINTA$;
350 IF POS(0)<>0 THEN RETURN
360 IF ZU+CSRLIN+1>=ZM THEN BEEP:AF=1:ZU=ZU-1
370 IF AF THEN ZU=ZU+1:X=USR1(TB+ZU*SM)
380 RETURN
390 REM Cursor rechts
400 IF POS(0)=SM-1 AND CSRLIN=23-ZO THEN AF=-1:A$=CHR$(13)+CHR$(10)
410 PRINTA$;
420 GOTO 350
430 REM Cursor links
440 IF POS(0)=0 AND CSRLIN=0 THEN LOCATE SM-1:GOTO 460
450 PRINTA$;:RETURN
460 REM
470 IF ZU=0 THEN LOCATE 0:BEEP:RETURN
480 ZU=ZU-1
490 X=USR1(TB+ZU*SM)
```

```
500 RETURN
510 REM Cursor naar boven
520 IF (PEEK(&HFBEB)AND2)=0 THEN 550
530 IF CSRLIN=0 THEN 460
540 PRINTA$;:RETURN
550 REM
560 ZU=ZU-24+ZO:IF ZU<0 THEN ZU=0
570 X=USR1(TB+ZU*SM):RETURN
580 REM
590 IF (PEEK(&HFBEB)AND2)=0 THEN 620:REM CTRL ?
600 IF CSRLIN=23-ZO THEN AF=-1:PRINTCHR$(10);:GOTO 360
610 PRINTA$;:RETURN
620 REM
630 J=ZU+24-ZO:IF J<ZM-24+ZO THEN ZU=J ELSE ZU=ZM-24+ZO
640 X=USR1(TB+ZU*SM):RETURN
650 REM
660 IF (PEEK(&HFBEB)AND1)=0 THEN 690
670 IF CSRLIN=23-ZO THEN AF=-1:PRINTA$;CHR$(10);:GOTO 360:REM Scrolling?
680 PRINTA$;CHR$(10);:RETURN
690 REM Shift RETURN
700 I=TB+(ZU+CSRLIN)*SM
710 FOR J=I+POS(0) TO I+SM-1:POKE J,32:NEXT
720 POKE I+POS(0),208
730 GOSUB 670:X=USR1(TB+ZU*SM):REM Return
740 RETURN
750 REM INSERT
760 X=USR2(TB+(ZU+CSRLIN)*SM+POS(0-1)
770 X=USR1(TB+ZU*SM)
780 RETURN
1000 REM
1010 CLS
1020 Y$=STRING$(OB,10)
1030 AA=TB
1040 ZZ=0
1050 X$=STRING$(LR,32)
1060 REM
```

```
1070 LPRINT Y$;
1080 REM
1090 Z$=""
1100 VP=VARPTR(Z$)
1110 POKE VP,AS:REM lengte
1120 POKE VP+1,AA-INT(AA/256)*256
1130 POKE VP+2,INT(AA/256)-256*(AA<0)
1140 AA=AA+AS
1150 PO=INSTR(z$,E$)
1160 IF PO<>0 OR AA>&HF1FF THEN
LPRINT STRING$(PZ-OB-AZ,10):
POKE &HF3DE,257-ZO:X=USR1(TB):RETURN 80
1170 PO=INSTR(Z$,CR$)
1180 IF PO=0 THEN 1240
1190 REM Shift Return
1200 POKE VP,PO-1
1210 AA=AA-AS+PO
1220 IF PEEK(AA)<>32 THEN 1310 ELSE AA=AA+1:GOTO 1220:REM
1230 GOTO 1310
1240 REM
1250 IF PEEK(AA)=32 THEN AA=AA+1:GOTO 1310
1260 I=0
1270 AA=AA-1:I=I+1
1280 IF PEEK(AA)<>32 THEN 1270
1290 POKE VP,PEEK(VP)-I
1300 AA=AA+1
1310 LPRINTX$;Z$
1320 ZZ=ZZ+1
1330 FI ZZ=AZ THEN LRPINT STRING$(PZ-OB-AZ,10):GOTO 1060
1340 GOTO 1080
1350 REM opslaan
1360 X=USR4(1):REM Keys wissen
1370 POKE &HF3DE,0:POKE &HF3B1,25
1380 LOCATE 0,22,1
1390 INPUT"Naam :";N$
1400 BSAVE "CAS:"+N$,TB,'HF1FF
```

```
1410 LOCATE 0,0,1
1420 POKE &HF3DE,257-ZO:POKE &HF3B1,23
1430 X=USRS(1):REM Keys tonen
1440 RETURN 80
1450 REM laden
1460 X=USR4(1):REM Keys wissen
1470 POKE &HF3DE,0:POKE &HF3B1,25
1480 LOCATE 0,22,1
1490 INPUT"Naam :";N$
1500 BLOAD "CAS:"+N$
1510 LOCATE 0,0,1
1520 POKE &HF3DE,257-ZO:POKE &HF3B1,23
1530 X=USR5(1)
1540 RETURN 80
2000 REM Init
2010 SCREEN 0
2020 COLOR 14,4,4
2030 DEFINT A-Z
2040 TB=&HE700
2050 SM=36:WIDTH SM
2060 ZM=INT((&HF1FF-TB(/SM)
2070 ZO=3
2080 POKE &HF3B1,23
2090 ON STOP GOSUB 230
2100 STOP ON
2110 ON ERROR GOTO 230
2120 REM
2130 FOR I=&HF300 TO &HF326:READ A$:POKE I,VAL("&H"+A$): NEXT
2140 DATA 23,23,5E,23,56,3A,B0,F3
2150 DATA 4F,06,00,CD,32,0C,21,01,01,F5
2160 DATA E5,D5,CD,F2,0B,D1,C5,CD
2170 DATA 45,07,C1,E1,23,F1,3D,20
2180 DATA EE,CD,ED,09,C9
2190 DEFUSR1=&HF300
2200 REM
2210 FOR I=&HF330 TO &HF371:READ A$:POKE I,VAL("&H"+A$):NEXT
```

```
2220 DATA 23,23,7E,23,66,6F,E5,11
2230 DATA FF,F1,EB,B7,ED,52,E3,C1,E5
2240 DATA 3E,D0,ED,B1,D1,E5,B7,ED
2250 DATA 52,E3,7E,FE,20,28,1A,E5
2260 DATA 21,FF,F1,3A,B0,F3,5F,16
2270 DATA 00,E5,19,D1,EB,03,ED,B8,47,23,3E,20
2280 DATA 77,23,10,FC,E1,C1,54,5D
2290 DATA 1B,EB,ED,B8,C9
2300 DEFUSR2=&HF330
2310 REM
2320 FOR I=&HF2B0 TO &HF2F0:READ A$:POKE I,VAL("&H"+A$):NEXT
2330 DATA 23,23,7E,23,66,6F,E5,11
2340 DATA FF,F1,EB,B7,ED,52,E3,C1,E5
2350 DATA 3E,D0,ED,B1,D1,B7,ED,52
2360 DATA 44,4D,62,6B,1B,ED,B0,2B
2370 DATA 36,20,E5,3A,B0,F3,47,7E
2380 DATA 23,FE,20,20,11,10,FB,EB,21
2390 DATA FF,F1,B7,ED,52,44,4D,E1
2400 DATA 00,EB,ED,B0,C9,E1,C9
2410 DEFUSR3=&HF2B0
2420 REM
2430 FOR I=&HF280 TO &HF28D:READ A$:POKE I,VAL("&H"+A$):NEXT
2440 DATA 21,00,E7,36,20,11,01,E7
2450 DATA 01,FF,1A,ED,B0,C9
2460 DEFUSR7=&HF280
2470 ON KEY GOSUB 1000,1350,1450
2480 KEY 1,"print":KEY (1) ON
2490 KEY 2,"save":KEY (2) ON
2500 KEY 3,"load":KEY (3) ON
2510 LR=10:AS=60
2520 OB=5:AZ=60:PZ=72
2530 KEY ON:CLS:LOCATE 0,0,1:POKE &HF3DE,256-ZO+1
2540 E$=CHR$(219):CR$(208)
2550 DEFUSR4=&HB15
2560 DEFUSR5=&HB2B
2570 RETURN
```

```
F000            10
F000            20 EINDE  EQU
F000            30                    ; overdracht RAM adres
F000 23         40        INC HL
F001 23         50        INC HL
F002 7E         60        LD A,(HL)
F003 23         70        INC HL
F004 66         80        LD H,(HL)
F005 6F         90        LD L,A
F006 E5         100       PUSH HL     ; aktueel RAM adres
F007 11FFF1     110       LD DE,EINDE ; teksteinde
F00A EB         120       EX DE,HL
F00B B7         130       OR A        ; carry wissen
F00C ED52       140       SBC HL,DE   ; aantal tot teksteinde
F00E E3         150       EX (SP),HL
F00F C1         160       POP BC      ; aantal
F010 E5         170       PUSH        ; RAM adres
F011 3ED0       180       LD A,200    ; shift return
F013 EDB1       190       CPIR        ; zoeken
F015 D1         200       POP DE      ; RAM adres
F016 E5         210       PUSH HL     ; adres SHIFT?
                                      RETURN&1=doeladr.
F017 B7         220       OR A        ; carry = 0
F018 ED52       230       SBC HL,DE   ; aantal te verplaatsten
                                        bytes
F01A E3         240       EX (SP),HL
F01B 7E         250       LD A,(HL)
F01C FE20       260       CP 32       ; spatie?
F01E 28FE       270       JR Z,OK     ; Ja, dan is er nog plaats
F020 E5         280       PUSH HL     ; adres einddoel
F021 21FF1      290       LD HL,EINDE
F024 3AB0F3     300       LD          ; width
                          A,(&HF3B0)
F027 5F         310       LD E,A
F028 1600       320       LD D,0
F02A E5         330       PUSH HL     ; teksteinde
F02B 19         340       ADD HL,DE   ; plus width
F02C D1         350       POP DE
F02D EB         360       EX DE,HL
F02E 03         370       INC BC
```

```
F02F EDB8      380          LDDR          ; totale resttekst verschuiven
F031 47        390          LD B,A
F032 3E20      400          LD A,&H20
F034 77        410 NEXT     LD (HL),A     ; vrijkomende regels wissen
F035 23        420          INC HL
F036 10FC      430          DJNZ NEXT
F038 E1        440          POP HL
*****regel 270: OK=$HF039 Offset 19*****
F039 C1        450 OK       POP BC
F03A 54        460          LD D,H
F03B 5D        470          LD E,L
F03C 1B        480          DEC DE
F03D EB        490          EX DE,HL
F03E EDB8      500          LDDR          ; regels tot SHIFT/RETURN met 1 opschuiven.
F040 C9        510          RET
```

Programma: insert
Start &HF000 Einde &HF040
Lengte &H41 bytes
Fouten 0
Variabelentabel
EINDE FIFF NEXT F034
OK F039

```
F000           10                         ; delete routine
F000           20                         ; overdracht RAM adres
F000           30 EINDE     EQU &HFIFF
F000 23        40           INC HL
F001 23        50           INC HL
F002 7E        60           LD A,(HL)
F003 23        70           INC HL
F004 66        80           LD H,(HL)
F005 6F        90           LD L,A
F006 E5        100          PUSH HL       ; RAM adres
F007 11FFF1    110          LD DE,EINDE
```

```
F00A EB          120          EX DE,HL
F00B B7          130          OR A
F00C ED52        140          SBC HL, DE      ; aantal tot teksteinde
F00E E3          150          EX (SP),HL
F00F C1          160          POP BC          ; aantal
F010 E5          170          PUSH HL         ; RAM adres
F011 3EDO        180          LD A,208        ; Shift/return
F013 EDB1        190          CPIR            ; zoeken
F015 D1          200          POP DE
F016 B7          210          OR A
F017 ED52        220          SBC HL,DE       ; aantal te verplaatsen by-
                                              tes
F019 44          230          LD B,H
F01A 4D          240          LD C,L
F01B 62          250          LD H,D
F01C 6B          260          LD L,E
F01D 1B          270          DEC DE          ; -1 = doeladres
F01E EDB0        280          LDIR
F020 2B          290          DEC HL          ; oud adres van shift re-
                                              turn
F021 3620        300          LD (HL),&H20    ; wissen
F023 E5          310          PUSH HL
F024 3AB0F3      320          LD              ; width
                              A,(&HF3B0)
F027 47          330          LD B,A
F028 7E          340TEST      LD A,(HL)
F029 23          350          INC HL
F02A FE20        360          CP 32           : spatie?
F02C 20FE        370          JR NZ,TERUG     ; nee? dan einde
F02E 10F8        380          DJNZ TEST       ; gehele regel testen
F030 EB          390          EX DE,HL        ; hele regel met spaties
                                              wissen
F031 21FFF1      400          LD HL,EINDE

F034 B7          410          OR A
F035 ED52        420          SBC HL,DE
F037 44          430          LD B,H
F038 4D          440          LD C,L          ; aantal
F039 E1          450          POP HL          ; adres oude shift return
F03A 2B          460          DEC HL
```

```
F03B EB       470          EX DE,HL
F03C EDB0     480          LDIR          ; evt. vrijgekomen regels
                                         wissen
F03E C9       490          RET

*****regel 370: TERUG= $HF03F Offset 11**** *
F03F E1       500TERUG     POP HL
F040 C9       510          RET
```

Programma: delete
Start &HF000 Einde &HF040
Lengte &H41 bytes
Fouten 0
Variabelentabel:
EINDE FIFF TEST F028
TERUG F03F

```
F000          10
F000          20                          ; Tekstindicatie routine
                                          ; overdracht van de eer-
                                          ste, te indiceren letter
F000          30 EINDE     EQU &HF1FF
F000 23       40           INC HL
F001 23       50           INC HL
F002 5E       60           LD E,(HL)
F003 23       70           INC HL
F004 56       80           LD D,(HL)
F005 3AB0F3   685          LD A,          ; width
                           (&HF3B0)
F008 4F       90           LD C,A
F009 0600     100          LD B,0         ; BC=kolommenteller
F00B CD320C   110          CALL &H0C32    ; tekens p. regel p. scherm
                                          in accu
F00E 210101   120          LD &H0101      ; kolom/regel
F011 F5       130 NEXT     PUSH AF
F012 E5       140          PUSH HL
F013 D5       150          PUSH DE
F014 CDF20B   160          CALL &H0BF2    ; VRAM-adres uit ko-
                                          lom/regel berekenen
F017 D1       170          POP DE         RAM-adres
```

```
F018 C5      180        PUSH BC          ; Tekens per regel
F019 CD4507  190        CALL &H0745      bloklaadroutine RAM
                                         VRAM
F01C C1      200        POP BC           ; aantal per regel
F01D E1      210        POP HL           ; cursorpositie
F01E 23      220        INC HL           ; naar volgende regel
F01F F1      230        POP AF           ; regelteller
F020 3D      240        DEC A            ; verlagen
F021 20EE    250        JR NZ,NEXT       ; indien geen nul, dan ver-
                                         der
F023 CDED09  260        CALL &H09ED      ; cursor weer indiceren
F026 C9      270        RET              ; terug naar BASIC
```

Programma: indic
Start: &HF000 Einde: &HF026
Lengte: &H27 bytes
Fouten: 0
Variabelentabel:
EINDE F1FF NEXT F011

```
F000         10                          ; tekstgebied wissen
F000 2100E7  20         LD HL,&HE700     ; begin tekstgebied
F003 3620    30         LD (HL),32       ; wissen
F005 1101E7  40         LD DE,&HE701     ; doeladres
F008 01FF1A  50         LD               ; aantal = &HF200-
                        BC,&H1AFF        &HE701
F00B EDB0    60         LDIR             ; gebied wissen
F00D C9      70         RET
```

Programma: wissen
Start: &HF000 Einde: &HF00D
Lengte: &He bytes
Fouten: 0

OMREKENINGSTABEL DECIMAAL-HEXADECIMAAL-BINAIR

dec	hex	binair	dec	hex	binair
0	&H00	&B00000000	26	&H1A	&B00011010
1	&H01	&B00000001	27	&H1B	&B00011011
2	&H02	&B00000010	28	&H1C	&B00011100
3	&H03	&B00000011	29	&H1D	&B00011101
4	&H04	&B00000100	30	&H1E	&B00011110
5	&H05	&B00000101	31	&H1F	&B00011111
6	&H06	&B00000110	32	&H20	&B00100000
7	&H07	&B00000111	33	&H21	&B00100001
8	&H08	&B00001000	34	&H22	&B00100010
9	&H09	&B00001001	35	&H23	&B00100011
10	&H0A	&B00001010	36	&H24	&B00100100
11	&H0B	&B00001011	37	&H25	&B00100101
12	&H0C	&B00001100	38	&H26	&B00100110
13	&H0D	&B00001101	39	&H27	&B00100111
14	&H0E	&B00001110	40	&H28	&B00101000
15	&H0F	&B00001111	41	&H29	&B00101001
16	&H10	&B00010000	42	&H2A	&B00101010
17	&H11	&B00010001	43	&H2B	&B00101011
18	&H12	&B00010010	44	&H2C	&B00101100
19	&H13	&B00010011	45	&H2D	&B00101101
20	&H14	&B00010100	46	&H2E	&B00101110
21	&H15	&B00010101	47	&H2F	&B00101111
22	&H16	&B00010110	48	&H30	&B00110000
23	&H17	&B00010111	49	&H31	&B00110001
24	&H18	&B00011000	50	&H32	&B00110010
25	&H19	&B00011001	51	&H33	&B00110011
52	&H34	&B00110100	78	&H4E	&B01001110
53	&H35	&B00110101	79	&H4F	&B01001111
54	&H36	&B00110110	80	&H50	&B01010000
55	&H37	&B00110111	81	&H51	&B01010001
56	&H38	&B00111000	82	&H52	&B01010010
57	&H39	&B00111001	83	&H53	&B01010011

dec	hex	binair	dec	hex	binair
58	&H3A	&B00111010	84	&H54	&B01010100
59	&H3B	&B00111011	85	&H55	&B01010101
60	&H3C	&B00111100	86	&H56	&B01010110
61	&H3D	&B00111101	87	&H57	&B01010111
62	&H3E	&B00111110	88	&H58	&B01011000
63	&H3F	&B00111111	89	&H59	&B01011001
64	&H40	&B01000000	90	&H5A	&B01011010
65	&H41	&B01000001	91	&H5B	&B01011011
66	&H42	&B01000010	92	&H5C	&B01011100
67	&H43	&B01000011	93	&H5D	&B01011101
68	&H44	&B01000100	94	&H5E	&B01011110
69	&H45	&B01000101	95	&H5F	&B01011111
70	&H46	&B01000110	96	&H60	&B01100000
71	&H47	&B01000111	97	&H61	&B01100001
72	&H48	&B01001000	98	&H62	&B01100010
73	&H49	&B01001001	99	&H63	&B01100011
74	&H4A	&B01001010	100	&H64	&B01100100
75	&H4B	&B01001011	101	&H65	&B01100101
76	&H4C	&B01001100	102	&H66	&B01100110
77	&H4D	&B01001101	103	&H67	&B01100111
104	&H68	&B01101000	130	&H82	&B10000010
105	&H69	&B01101001	131	&H83	&B10000011
106	&H6A	&B01101010	132	&H84	&B10000100
107	&H6B	&B01101011	133	&H85	&B10000101
108	&H6C	&B01101100	134	&H86	&B10000110
109	&H6D	&B01101101	135	&H87	&B10000111
110	&H6E	&B01101110	136	&H88	&B10001000
111	&H6F	&B01101111	137	&H89	&B10001001
112	&H70	&B01110000	138	&H8A	&B10001010
113	&H71	&B01110001	139	&H8B	&B10001011
114	&H72	&B01110010	140	&H8C	&B10001100
115	&H73	&B01110011	141	&H8D	&B10001101

dec	hex	binair	dec	hex	binair
116	&H74	&B01110100	142	&H8E	&B10001110
117	&H75	&B01110101	143	&H8F	&B10001111
118	&H76	&B01110110	144	&H90	&B10010000
119	&H77	&B01110111	145	&H91	&B10010001
120	&H78	&B01111000	146	&H92	&B10010010
121	&H79	&B01111001	147	&H93	&B10010011
122	&H7A	&B01111010	148	&H94	&B10010100
123	&H7B	&B01111011	149	&H95	&B10010101
124	&H7C	&B01111100	150	&H96	&B10010110
125	&H7D	&B01111101	151	&H97	&B10010111
126	&H7E	&B01111110	152	&H98	&B10011000
127	&H7F	&B01111111	153	&H99	&B10011001
128	&H80	&B10000000	154	&H9A	&B10011010
129	&H81	&B10000001	155	&H9B	&B10011011
156	&H9C	&B10011100	182	&HB6	&B10110110
157	&H9D	&B10011101	183	&HB7	&B10110111
158	&H9E	&B10011110	184	&HB8	&B10111000
159	&H9F	&B10011111	185	&HB9	&B10111001
160	&HA0	&B10100000	186	&HBA	&B10111010
161	&HA1	&B10100001	187	&HBB	&B10111011
162	&HA2	&B10100010	188	&HBC	&B10111100
163	&HA3	&B10100011	189	&HBD	&B10111101
164	&HA4	&B10100100	190	&HBE	&B10111110
165	&HA5	&B10100101	191	&HBF	&B10111111
166	&HA6	&B10100110	192	&HC0	&B11000000
167	&HA7	&B10100111	193	&HC1	&B11000001
168	&HA8	&B10101000	194	&HC2	&B11000010
169	&HA9	&B10101001	195	&HC3	&B11000011
170	&HAA	&B10101010	196	&HC4	&B11000100
171	&HAB	&B10101011	197	&HC5	&B11000101
172	&HAC	&B10101100	198	&HC6	&B11000110
173	&HAD	&B10101101	199	&HC7	&B11000111

dec	hex	binair	dec	hex	binair
174	&HAE	&B10101110	200	&HC8	&B11001000
175	&HAF	&B10101111	201	&HC9	&B11001001
176	&HB0	&B10110000	202	&HCA	&B11001010
177	&HB1	&B10110001	203	&HCB	&B11001011
178	&HB2	&B10110010	204	&HCC	&B11001100
179	&HB3	&B10110011	205	&HCD	&B11001101
180	&HB4	&B10110100	206	&HCE	&B11001110
181	&HB5	&B10110101	207	&HCF	&B11001111
208	&HD0	&B11010000	232	&HE8	&B11101000
209	&HD1	&B11010001	233	&HE9	&B11101001
210	&HD2	&B11010010	234	&HEA	&B11101010
211	&HD3	&B11010011	235	&HEB	&B11101011
212	&HD4	&B11010100	236	&HEC	&B11101100
213	&HD5	&B11010101	237	&HED	&B11101101
214	&HD6	&B11010110	238	&HEE	&B11101110
215	&HD7	&B11010111	239	&HEF	&B11101111
216	&HD8	&B11011000	240	&HF0	&B11110000
217	&HD9	&B11011001	241	&HF1	&B11110001
218	&HDA	&B11011010	242	&HF2	&B11110010
219	&HDB	&B11011011	243	&HF3	&B11110011
220	&HDC	&B11011100	244	&HF4	&B11110100
221	&HDD	&B11011101	245	&HF5	&B11110101
222	&HDE	&B11011110	246	&HF6	&B11110110
223	&HDF	&B11011111	247	&HF7	&B11110111
224	&HE0	&B11100000	248	&HF8	&B11111000
225	&HE1	&B11100001	249	&HF9	&B11111001
226	&HE2	&B11100010	250	&HFA	&B11111010
227	&HE3	&B11100011	251	&HFB	&B11111011
228	&HE4	&B11100100	252	&HFC	&B11111100
229	&HE5	&B11100101	253	&HFD	&B11111101
230	&HE6	&B11100110	254	&HFE	&B11111110
231	&HE7	&B11100111	255	&HFF	&B11111111

Register

Dit boek laat u zien wat u allemaal met een MSX kunt doen! Een greep uit de inhoud: een tekensetgenerator, windows, tex/grafiek hardcopy, joystick-programmering, terminalprogramma, systeemroutines, peeks en pokes, tokens, listbeveiliging, genereren van data-regels enzovoort. Uiteraard zult u in dit boek de nodige voorbeeldprogramma's niet missen!

ISBN 90 229 3371 7

MSX Bibliotheek 5

DATA BECKER
NEDERLANDS