MSX2 LEERBOEK

ALLE EXTRA'S VAN MSX2

DEEL 4

WESSEL AKKERMANS

MSX2 LEERBOEK MSX2 DEEL 4
WESSEL AKKERMANS

MSX Leerboek
deel 4
Alle extra's van MSX2

MSX2 LEERBOEK

ALLE EXTRA'S VAN MSX2

DEEL 4

WESSEL AKKERMANS

uitgeverij STARK-TEXEL b.v.
Postbus 302 1794 ZG Oosterend Nh tel. 02220-18661 fax -18495

CIP-gegevens Koninklijke Bibliotheek, Den Haag

Akkermans, Wessel

MSX leerboek / Wessel Akkermans.
– Oosterend : uitgeverij Stark-Texel b.v.
Dl. 4: Alle extra's van MSX2.
ISBN 90 6398 737 4
SISO 365.3 UDC 681.3 NUGI 851
Trefw.: MSX (computer).

eerste druk april 1989

ISBN 90 6398 737 4

INHOUD

VOORWOORD

Bij de opzet voor dit boek, het vierde deel uit de leerboeken serie, heb ik lang nagedacht over de inhoud. Immers, na dit boek komt er niets meer. De vier delen samen wil je toch graag zo kompleet mogelijk hebben.

In eerste instantie is dit deel bedoeld om de mogelijkheden van de MSX2 computers uit de doeken te doen. Alle hoofdstukken tot en met hoofdstuk 8 zullen dan ook over MSX2 gaan. Zou ik dit boek echter beperken tot MSX2 alleen, dan zouden een aantal onderwerpen, die ook in de vorige delen niet aan de orde kwamen, tussen de wal en het schip geraken. Vandaar, dat ik besloot om in dit deel ook iets meer over *schijven* te vertellen.

Datakommunikatie heeft ook voor MSX'ers een grote vlucht genomen. Er zijn seriële interfaces te koop, die in een expansion slot van de MSX computer (zowel MSX1 als MSX2) kan worden gestoken, waarna het MSX BASIC met een aantal BASIC statements wordt uitgebreid.

Zowel schijven als seriële interfaces komen voor op zowel MSX1 als op MSX2 computers. Deze twee onderwerpen vallen dus eigenlijk een beetje buiten het kader van dit boek. Toch meen ik er goed aan te doen deze onderwerpen te behandelen, om zodoende een zo kompleet mogelijk geheel te krijgen.

Het nieuwe beeldschermgeheugen en alle statements, die daarmee te maken hebben, worden in de hoofdstukken 1 en 2 behandeld. De zeer uitgebreide kleurenmogelijkheden komen in hoofdstuk 3 aan de orde. Doordat MSX2 een hogere schermresolutie en meer kleurmogelijkheden heeft dan MSX1, zijn de teken-statements zoals CIRCLE, LINE, PAINT, enz. uitgebreid. Deze uitbreidingen worden in hoofdstuk 4 behandeld. De video display processor van de MSX2 computer wordt in hoofdstuk 5 onder de loep genomen. Hoofdstuk 6 behandelt alle

statements die u al van de MSX1 kende, doch die nu een enigszins gewijzigde funktie hebben of een uitbreiding hebben ondergaan. Hoofdstuk 7 gaat over de klok-chip en de daarbij behorende statements. Een heel interessant onderwerp is de RAM-disk. Het aanmaken en gebruiken van een RAM-disk wordt in hoofdstuk 8 behandeld.

De laatste twee hoofdstukken bevatten onderwerpen die niet strikt zijn voorbehouden aan MSX2. Hoofdstuk 9 gaat dieper in op het formaat van schijven en methodes om van die diepergaande kennis gebruik te maken. U zult onder meer zien hoe u zelf een inhoudsopgave van een schijf kunt maken, waarin veel meer informatie staat dan u onder MSXDOS krijgt. Hoofdstuk 10 behandelt de seriële interface. Deze interface wordt op bijna geen enkele MSX computer standaard meegeleverd. Er zijn echter interface modules te koop als optie. Deze interfaces bieden u de mogelijkheid zelf kommunikatieprogramma's te schrijven (er zijn ook uitstekende kommunikatieprogramma's te koop). De BASIC statements die nodig zijn om zelf een kommunikatie-programma te schrijven, samen met een korte theoretische inleiding over datakommunikatie worden beschreven in hoofdstuk 10.

Het boek wordt besloten met een groot voorbeeldprogamma (appendix A). Met dit programma kunt u zichzelf volgens het tienvingersysteem leren typen. Het programma bevat een groot aantal voorbeelden van de grafische mogelijkheden van de MSX2.

HOOFDSTUK 1

NIEUWE BEELDSCHERMFORMATEN

Wat bij MSX2 computers het meest in het oog springt zijn de uitgebreidere grafische mogelijkheden ten opzichte van MSX1 computers. Ten eerste is de resolutie hoger en ten tweede zijn er veel meer kleuren mogelijk. Wie wat langer met een MSX2 computer werkt zal ontdekken, dat het nu ook mogelijk is ieder beeldpuntje een eigen kleur te geven, zonder de MSX1 beperking dat er per acht beeldpuntjes maar twee verschillende kleuren mogen worden gebruikt. Ook het inkleuren van vlakken werkt nu beter dan op MSX1 computers. Nu zou u kunnen verwachten, dat al die nieuwe en uitgebreidere mogelijkheden het systeem langzamer maken. Niets is minder waar. De MSX2 beeldschermen werken veel sneller dan de oude MSX1 schermen. Dit danken we aan het feit, dat er in de MSX2 een nieuwe video controller chip is ingebouwd, die niet alleen meer kan, maar dat ook nog sneller doet.

De eerste vijf hoofdstukken in dit boek zijn aan de BASIC statements gewijd, waarmee de mogelijkheden van die nieuwe video chip worden gebruikt. In dit eerste hoofdstuk zullen we de verschillende scherminstellingen (SCREEN 0 tot en met 8) behandelen.

1.1 Scherminstellingen

Het instellen van de verschillende formaten beeldschermen wordt met het reeds uit MSX1 bekende SCREEN statement gedaan. Aan het SCREEN statement kunnen, zoals u zich zult herinneren, een aantal parameters worden toegevoegd. Onder MSX2 is dat aantal parameters met een uitgebreid, zoals in het formaat van het SCREEN statement, dat hierna volgt, is te zien:

```
SCREEN [<mode>][,[<sprite>][,[<klik>][,[<baud>]
       [,[<printertype>][,[<interlace>]]]]]]
```

Dat het formaat alleen maar is uitgebreid met een parameter, wil nog niet zeggen dat er verder niets is gewijzigd aan het SCREEN statement. Onder MSX1 kan voor *<mode>* worden gekozen uit de waarden 0 tot en met 3. Onder MSX2 kan echter worden gekozen uit de modes 0 tot en met 8.

***<mode>*:**

SCREEN mode 0 stelt het tekstscherm in. Wanneer deze schermmode op een MSX2 computer wordt ingesteld, dan is het maximum aantal tekens per regel 80. Om een langere regellengte dan 40 tekens te kiezen dient het WIDTH-statement te worden gebruikt. Het formaat van dat statement is:

```
WIDTH <regellengte>
```

Voor regellengte mag een getal worden gekozen van 1 tot 80, hetgeen overeenkomt met 1 tot en met 80 tekens per regel. Wanneer u een getal van 40 of kleiner invult, dan krijgt u de grote tekens, zoals ze ook op een MSX1 computer worden weergegeven. Kiest u een waarde die hoger is dan 40, dan worden smallere letters op het scherm geschreven, waarvan er maximaal 80 op een regel passen. Indien een regel niet geheel gevuld is (dat wil zeggen minder dan 40 of minder dan 80 tekens), dan wordt de regel gecentreerd op het beeldscherm afgedrukt.

Voor het overige zijn de scherminstellingen 0 tot en met 3 geheel gelijk aan dezelfde instellingen van MSX1 computers. MSX2 computers hebben echter een aantal nieuwe scherminstellingen. Deze nieuwe instellingen zijn alle grafische scherminstellingen. De bijzonderheden van die nieuwe instellingen zullen we nu een voor een behandelen.

SCREEN 4

Hiermee wordt een grafisch scherm ingesteld, dat qua resolutie geheel overeenkomt met het grafisch scherm dat met SCREEN 2 wordt ingesteld. Het enige verschil is, dat er nu sprites van meerdere kleuren kunnen worden aangestuurd, waarvan er dan ook nog acht tegelijk op dezelfde lijn mogen staan (tegenover vier in SCREEN 2). Deze nieuwe sprite mogelijkheden gelden overigens voor alle nieuwe scherm instellingen (SCREEN 4 tot en met 8).

```
1000 SCREEN 4
1010 FOR I=1 TO 256
1020 LINE (I,0)-(I,50),I MOD 16
1030 NEXT I
1040 GOTO 1040
```

SCREEN 5

Met dit statement wordt een grafisch scherm van 256 beeldpuntjes horizontaal bij 212 beeldpuntjes vertikaal ingesteld. Voor wat betreft de sprites gelden voor dit scherm dezelfde regels als voor SCREEN 4. Elk beeldpuntje kan afzonderlijk worden gekleurd, waarbij keuze uit 16 verschillende kleuren kan worden gemaakt. De beperking uit SCREEN modes 2 en 4, waarbij per 8 horizontale beeldpuntjes slechts twee verschillende kleuren mogen voorkomen, is hier niet meer van toepassing. Een kompleet scherm neemt in het video geheugen 32 kBytes ruimte in beslag. In een MSX2 computer met 64 kBytes video gheugen kunnen derhalve twee schermen worden opgeslagen. MSX2 computers met een 128 kBytes videogeheugen kunnen zelfs vier schermen opslaan. Later zullen we terugkomen op het video geheugen, waarin meerdere schermen kunnen worden opgeslagen.

```
1000 SCREEN 5
1010 FOR I=1 TO 256
1020 LINE (I,0)-(I,50),I MOD 16
1030 NEXT I
1040 GOTO 1040
```

SCREEN 6

Met SCREEN 6 wordt een grafisch scherm van 512 horizontale beeldpuntjes bij 212 vertikale beeldpuntjes gedefinieerd. Ieder beeldpuntje kan afzonderlijk worden gekleurd. Daarbij is er keuze uit vier kleuren. Een kompleet scherm neemt 32 kBytes ruimte in het video geheugen in beslag. Er zijn wel twee keer zoveel beeldpuntjes als onder scherminstelling 5, doch de beeldpuntjes kunnen minder verschillende kleuren hebben, vandaar dat de grootte van het scherm in het video geheugen gelijk is aan een scherm uit scherminstelling 5. Ook nu geldt, dat er in een video geheugen van 64 kBytes twee schermen kunnen worden opgeslagen en in een MSX2 computer met 128 kBytes videogeheugen vier schermen.

```
1000 SCREEN 6
1002 COLOR=(0,7,7,7)
1004 COLOR=(1,7,0,0)
1006 COLOR=(2,0,7,0)
1008 COLOR=(3,0,0,7)
1010 FOR I=1 TO 4
1020 LINE (I*10,10)-(I*10+10,50),I-1,BF
1030 NEXT I
1040 GOTO 1040
```

De volgende twee scherminstellingen werken alleen op MSX2 computers met 128 kBytes video geheugen.

SCREEN 7

Hiermee wordt een grafisch scherm van 512 horizontale bij 212 vertikale beeldpuntjes ingesteld. Ieder beeldpuntje kan afzonderlijk worden gekleurd, waarbij er keuze is uit 16 verschillende kleuren. Een kompleet beeldscherm neemt 64 kBytes geheugen in beslag. Er kunnen derhalve twee verschillende schermen in het video geheugen worden opgeslagen.

```
1000 SCREEN 7
1010 FOR I=0 TO 511
1020 LINE (I,10)-(I,50),I MOD 16
1030 NEXT I
1040 GOTO 1040
```

SCREEN 8

Met dit statement wordt een scherminstelling bereikt van 256 horizontale bij 212 vertikale beeldpuntjes. Elk beeldpuntje kan afzonderlijk worden gekleurd. Daarbij is er keuze uit 256 verschillende kleuren. Een kompleet beeldscherm vergt 64 kBytes geheugenruimte. Er kunnen derhalve twee verschillende schermen in het video geheugen worden opgeslagen.

```
1000 SCREEN 8
1010 FOR I=0 TO 255
1020 LINE (I,10)-(I,50),I
1030 NEXT I
1040 GOTO 1040
```

<sprite>:

Hier kunt u de soort en grootte van de te gebruiken sprites invullen. Er is keuze uit één van de volgende waarden:

0 - kleine sprites van 8 bij 8 beeldpuntjes
1 - kleine sprites, vergroot tot 16 bij 16 beeldpuntjes
2 - grote sprites van 16 bij 16 beeldpuntjes
3 - grote sprites, vergroot tot 32 bij 32 beeldpuntjes

Het statement SCREEN 5,2 zal tot gevolg hebben dat eventueel gebruikte sprites grote sprites zijn van 16 bij 16 beeldpuntjes. Sprites zullen nader worden bekeken in hoofdstuk 3. Vooral de onder MSX2 beschikbare nieuwe kleurmogelijkheden zullen daar worden toegelicht.

<klik>:

Hiermee wordt het klikken van de toetsen aan of uit gezet. Er is geen verschil met MSX1. Een waarde 0 zet het klikken uit, 1 zet het klikken weer aan.

<baud>:

Hiermee wordt de lees- en schrijfsnelheid naar de datarecorder ingesteld. Er is geen verschil met MSX1. De waarde 1 stelt 1200 baud in, de waarde 2 stelt 2400 baud in.

<printertype>:

Hiermee wordt het soort printer dat op de computer is aangesloten aan het systeem kenbaar gemaakt. Indien een waarde 0 wordt opgegeven, dan geeft dat aan, dat er een MSX-printer aan het systeem is aangesloten. De waarde 1 geeft aan, dat het een andere soort printer is.

<interlace>:

Dit is een nieuwe parameter, die alleen onder MSX2 kan worden opgegeven. Indien niets wordt ingevuld, dan wordt de waarde 0 aangenomen. Dit wil zeggen dat het beeld normaal wordt weergegeven, net zo als op een MSX1 computer. Er zijn echter meer mogelijkheden:

0 - Normaal

1 - Interlaced

2 - Normaal, even/oneven schermpagina door elkaar.

3 - Interlaced, even/oneven schermpagina door elkaar.

Iedere vijftigste sekonde wordt er een beeld naar het scherm geschreven. Dit schrijven gebeurt door middel van 212 horizontale lijnen. Doordat dit 50 keer per sekonde wordt gedaan, en doordat de fluorescerende laag op het beeldscherm enige tijd nagloeit, is het steeds opnieuw schrijven van de lijnen niet te zien. Wat we wel zien zijn de lijnen zelf. Met "interlaced" kunnen we die lijnen laten verdwijnen. Interlaced wil zeggen, dat de lijnen, waaruit het beeld op het scherm wordt opgebouwd elke keer dat ze worden geschreven een klein beetje verschoven zijn ten opzichte van de vorige keer dat ze werden geschreven. Hierdoor krijgt het beeld een egaler aanzien.

Kiezen we voor parameter twee of drie, dan kunnen we twee beeldschermen uit het beeldschermgeheugen min of meer tegelijkertijd op het beeldscherm afdrukken. Dit werkt alleen, wanneer is gekozen voor een *oneven* pagina. Een pagina (en dus ook een oneven pagina) kan worden gekozen met behulp van het SET PAGE statement. Het formaat van dat statement is:

```
SET PAGE <weergave pagina>,<aktieve pagina>
```

Zoals eerder in dit hoofdstuk al werd opgemerkt, kunnen afhankelijk van de schermmode en de grootte van het videogeheugen meerdere beeldschermen in het videogeheugen worden opgeslagen. Het is dan mogelijk, om de ene pagina op het beeldscherm af te drukken en ondertussen een andere (niet zichtbare pagina) te beschrijven met nieuwe informatie. In het SET PAGE statement geeft de eerste parameter aan welke pagina uit het videogeheugen naar het beeldscherm moet worden afgedrukt. De tweede parameter geeft aan naar welke pagina in het

videogeheugen nieuwe informatie moet worden geschreven. Zou u bijvoorbeeld pagina 0 als weergave pagina specificeren en pagina 1 als aktieve pagina, en u geeft een CIRCLE opdracht, dan ziet u niet dat er een cirkel wordt getekend. Zodra u echter pagina 1 als weergave pagina specificeert, dan ziet u dat de cirkel al is getekend.

Zolang er nog geen verandering in de weergave- en aktieve schermpagina's is aangebracht met een SET PAGE statement, geldt dat beide parameters pagina 0 aanwijzen. Met andere woorden, u schrijft naar dezelfde pagina die u ook ziet.

In het volgende programma wordt gekozen voor een normaal scherm, waarbij de even en oneven pagina's door elkaar worden afgedrukt. Het wordt niet echt door elkaar afgedrukt. De ene keer wordt pagina 1 afgedrukt en de volgende keer pagina 0, dan weer pagina 1 enzovoort. De pagina's wisselen elkaar af met een snelheid van 50 keer per sekonde. Hierdoor lijkt het voor het menselijk oog alsof ze door elkaar worden afgedrukt. Dit door elkaar afdrukken werkt alleen wanneer wordt gekozen voor een oneven nummer (zoals in regel 1050 is te zien). Alleen in dat geval wordt de even pagina met het nummer van de oneven pagina minus 1 afwisselend met de oneven pagina afgedrukt.

```
1000 SCREEN 5,,,,,2
1010 CLS
1020 OPEN "grp:" AS #1
1030 CIRCLE (128,106),100
1040 PAINT (100,100),8,1
1050 SETPAGE 1,1
1060 CLS
1070 LINE (20,20)-(236,192),4,BF
1080 GOTO 1080
```

HOOFDSTUK 2

MANIPULEREN MET HET VIDEOGEHEUGEN

In het vorige hoofdstuk werd al gemeld dat het videogeheugen onder MSX2 veel groter is geworden en dat er meerdere beeldschermen tegelijk in kunnen worden opgeslagen. Het is onder meer mogelijk, om de ene schermpagina op het beeldscherm af te drukken, terwijl er naar een ander scherm wordt geschreven. In dit hoofdstuk zullen we wat nieuwe mogelijkheden van het COPY statement behandelen.

Met deze nieuwe uitbreidingen kan het COPY statement ondermeer stukken van het videogeheugen kopiëren naar een andere plaats in dezelfde of in een andere schermpagina. Ook kan een stuk video geheugen naar of uit een array worden gekopieerd en naar of uit een schijvenbestand. Ten slotte is het nog mogelijk om een array naar een schijvenbestand te schrijven of terug te lezen.

Bij het teruglezen van een stukje videogeheugen uit een schijvenbestand of uit een array kan de volgorde van de opgeslagen bitjes worden gemanipuleerd. Wat oorspronkelijk de linkerbovenhoek was kan nu de rechter bovenhoek of rechter onderhoek worden. Hiermee kan het stuk videogeheugen bijvoorbeeld worden gespiegeld.

Alle COPY statements in dit hoofdstuk werken alleen voor de schermmodes SCREEN 5 tot en met 8. Hier volgt nog een korte opsomming van de voor het COPY statement belangrijke gegevens over de beeldschermpagina's in de verschillende schermmodes.

Schermmode:	**Aantal pixels:**		**Paginanummers:**
	Hor(X)	**Ver(Y)**	
SCREEN 5	256	212	0, 1, (2, 3)
SCREEN 6	512	212	0, 1, (2, 3)
SCREEN 7	512	212	0, 1
SCREEN 8	256	212	0, 1

Vanaf nu zullen we er vanuit gaan, dat u een MSX2 computer met een 128 kBytes videogeheugen hebt. Hebt u een computer met 64 kBytes videogeheugen, dan kunt u de SCREEN modes 7 en 8 niet gebruiken en kunt u alleen paginanummers 0 en 1 gebruiken.

2.1 Van videogeheugen naar videogeheugen

Met het eerste te behandelen nieuwe COPY statement kunnen we de informatie van een beeldschermpagina (of een gedeelte daaruit) uit het videogeheugen kopiëren naar een andere beeldschermpagina, of naar dezelfde pagina. Het statement waarmee dit wordt gedaan heeft het volgende formaat:

```
COPY (x1,y1)-(x2,y2)[,<src>] TO (x3,y3)[,<dest>[,<log>]]
```

x1 en *y1* geven de pixel koördinaat aan van een diagonaal. *x2* en *y2* geven de pixel koördinaat van het andere einde van de diagonaal aan. De diagonaal verbindt de linker boven hoek met de rechter onderhoek van de rechthoek op het beeldscherm die moet worden gekopieerd. Deze rechthoek zal worden gekopieerd naar een plaats op het beeldscherm waarvan de linkerbovenhoek op de koördinaten *x3* en *y3* ligt. Het volgende programma laat hiervan een voorbeeldje zien. Het woordje "COPY" wordt links bovenaan het beeldscherm afgedrukt. Vervolgens wordt een rechthoekje gedefinieerd met de koördinaten *x1,y1* en *x2,y2*. Het woordje COPY valt precies binnen dit rechthoekje. Nu wordt het COPY statement uitgevoerd, waarbij het rechthoekje wordt gekopieerd naar de koördinaten X3,Y3. Deze laatste koördinaten zijn ongeveer midden op het beeldscherm gekozen. U zult bij uitvoering dan ook zien, dat het woordje COPY midden op het scherm verschijnt.

```
1000 SCREEN 5
1010 OPEN "GRP:" AS #1
1020 PRESET(10,10)
1030 PRINT #1,"COPY"
1040 FOR I=1 TO 500: NEXT I
1050 COPY (10,10)-(41,17) TO (100,100)
1060 GOTO 1060
```

<src>:

Met deze parameter kan worden aangegeven vanuit welke beeldscherm pagina moet worden gekopieerd. Voor SCREEN 5 en 6 kan worden gekozen voor paginanummers 0 tot en met 3. Voor SCREEN 7 en 8 kan worden gekozen voor paginanummers 0 of 1. Indien deze parameter wordt weggelaten, dan zal worden gekopieerd vanuit de aktieve pagina (zie ook het SET PAGE statement uit het vorige hoofdstuk).

<dest>:

Met deze parameter kan worden aangegeven naar welke beeldscherm pagina de kopie moet worden toe geschreven. Zie *<src>* voor bijzonderheden per SCREEN mode. Weglaten van deze parameter zal tot gevolg hebben dat er naar de aktieve pagina zal worden geschreven.

<log>:

Dit is een logische operator. Er is keuze uit een van de volgende mogelijkheden:

PSET
Het te kopiëren pixel wordt ongewijzigd naar een andere plaats op het beeldscherm gekopieerd. Ongewijzigd in dit verband wil zeggen dat het de oorspronkelijke kleur behoudt.

PRESET
Onder SCREEN 5 en 7 krijgt het gekopieerde pixel de kleur 15 minus het oorspronkelijke kleurnummer. Voorbeeld:
Kleur van het te kopiëren pixel: 8
Kleur van het gekopieerde pixel: 15 – 8 = 7

Onder SCREEN 6 krijgt het gekopieerde pixel de kleur met het nummer 3 minus het kleurnummer van de kleur van het oorspronkelijke pixel. Voorbeeld:
Kleur van het te kopiëren pixel: 3
Kleur van het gekopieerde pixel: 3 – 3 = 0

Onder SCREEN 8 krijgt het gekopieerde pixel de kleur met het nummer 255 minus het kleurnummer van de kleur van het oorspronkelijke pixel. Voorbeeld:
Kleur van het te kopiëren pixel: 44
Kleur van het gekopieerde pixel: 255 – 44 = 211

AND
Op de kleurnummers van de kleur van het te kopiëren pixel en het pixel waar naartoe wordt gekopieerd wordt de logische operatie AND verricht. Het resultaat levert het kleurnummer van het gekopieerde pixel op. Voorbeeld:
Kleur van het te kopiëren pixel: 8
Kleur van het pixel waarnaartoe moet worden gekopieerd: 3

9 binair = 1001
3 binair = 0011 AND
Resultaat 0001 = 1 (decimaal)

OR
Op de kleurnummers van de kleur van het te kopiëren pixel en het pixel waar naartoe wordt gekopieerd wordt de operatie OR verricht. Het resultaat is het kleurnummer van het gekopieerde pixel. Voorbeeld:

9 binair = 1001
3 binair = 0011 OR
Resultaat 1011 = 11 (decimaal)

XOR
Op de kleurnummers van de kleur van het te kopiëren pixel en het pixel waar naartoe wordt gekopieerd wordt de logische operatie XOR verricht. Het resultaat levert het kleurnummer van het gekopieerde pixel op. Voorbeeld:

9 binair = 1001
3 binair = 0011 XOR
Resultaat 1010 = 10 (decimaal)

TPSET, TPRESET, TAND, TOR en TXOR
Alle zojuist behandelde logische bewerkingen kunnen ook worden voorafgegaan door de letter T. Ze hebben dezelfde werking als hiervoor beschreven, alleen zullen pixels met de kleur 0 (Transparant) niet mee worden gekopieerd.

2.2 Van videogeheugen naar een array

Het is ook mogelijk om een deel van het beeldschermgeheugen in een array op te slaan. Later kan die array dan weer worden teruggekopieerd naar het beeldschermgeheugen. Hiervoor zijn twee formaten van het COPY statement beschikbaar. Nu zult u zich misschien afvragen wat het voordeel kan zijn van het kopiëren naar een array en vervolgens van die array naar het videogeheugen. Je kunt immers met een enkel statement direkt van videogeheugen naar videogeheugen kopiëren. Welnu, een belangrijke reden is, dat er tijdens het terugkopiëren van de array naar het beeldschermgeheugen met de te kopiëren bitjes kan worden gemanipuleerd. Zoals we straks in detail zullen zien, kan de kopie gespiegeld worden, zowel horizontaal als vertikaal.

Het statement waarmee een stuk beeldschermgeheugen naar een array wordt gekopieerd heeft het volgende formaat:

```
COPY (x1,y1)-(x2,y2)[,<SRC>] TO <array>
```

De betekenis van *x1, y1, x2, y2* en *<SRC>* is gelijk aan de betekenis van dezelfde parameters uit het vorige COPY statement, zoals beschreven in de vorige paragraaf.

<array>

Hier moet u de naam invullen van de numerieke array, waar het stuk video geheugen naartoe moet worden gekopieerd. Daarbij dient alleen de variabele naam, zonder de dimensie, te worden opgegeven. Zou u bijvoorbeeld een array A(5000) hebben gedefinieerd, dan dient in het COPY statement alleen de letter A te worden opgegeven.

De array moet worden gedimensioneerd, voordat het COPY statement wordt uitgevoerd. U dient zelf de dimensie te bepalen. Dat kan met de volgende formule:

Dimensie = `INT((P*ABS(X2-X1)+1)*(ABS(Y2-Y1)+1)+7)/8+4`

Voor SCREEN 5, 7 en 8 moet voor P de waarde 4 worden ingevuld.
Voor SCREEN 6 moet voor P de waarde 2 worden ingevuld.

Is een stuk van het videogeheugen eenmaal opgeslagen in een array, dan kunnen we het daar weer uit kopiëren naar het video geheugen met het volgende COPY statement:

```
COPY <array>[,<richting>] TO (x3,y3)[,<DEST>[,<LOG>]]
```

Alle parameters, behalve *<richting>*, werden al eerder uitgelegd. Met de parameter *<richting>* hebben we de volgende mogelijkheden:

0 = Kopieer van linksboven naar rechtsonder
1 = Kopieer van rechtsboven naar linksonder
2 = Kopieer van linksonder naar rechtsboven
3 = Kopieer van rechtsonder naar linksboven

Wat de gevolgen van deze parameter zijn kunt u zien met behulp van het volgende programma. In de FOR NEXT lus van regel 1110 worden alle richtingen, zoals hiervoor beschreven, een keer uitgevoerd.

Let u verder op regel 1030, waarin de grootte van de array wordt berekend. Hierbij is voor P de waarde 4 ingevuld. Wanneer u dit programma in SCREEN 6 zou willen testen, dan zult u de 4 moeten vervangen door een 2.

```
1000 X1=0: Y1=0
1010 X2=50: Y2=20
1020 X3=100: Y3=100
1030 N=INT((4*ABS(X2-X1)+1)*(ABS(Y2-Y1)+1)+7)/8+4
1040 DIM A(N)
1050 SCREEN 5
1060 OPEN "GRP:" AS #1
1070 PRESET(10,10)
1080 PRINT #1,"COPY"
1090 FOR I=1 TO 500: NEXT I
1100 COPY (X1,Y1)-(X2,Y2) TO A
1110 FOR R=0 TO 3
1120 COPY A,R TO (X3,Y3)
1130 FOR I=1 TO 500: NEXT I
1140 NEXT R
1150 GOTO 1150
```

2.3 Van videogeheugen naar schijvenbestand

Met een enigszins gewijzigd COPY statement is het mogelijk om een (stuk) videogeheugen naar een bestand op schijf te schrijven en later dit bestand weer van schijf te lezen en naar het video geheugen te schrijven. U zult zien, dat het formaat van deze COPY statements bijna gelijk is aan de statements waarmee naar een array kan worden geschreven. In plaats van een array wordt nu een bestandsnaam gespecificeerd.

Wat is nu het voordeel van het schrijven van het videogeheugen naar schijf? Er zijn meerdere voordelen. Ten eerste kan men een groot aantal beeldschermen naar schijf schrijven en die later op ieder willekeurig moment teruglezen. Op die manier is snel een nieuw scherm te voorschijn te toveren. Het gaat natuurlijk niet zo snel als naar en van een array; de array ligt immers in het geheugen zèlf en toegang tot het geheugen is altijd veel sneller dan toegang tot een schijf. Wanneer echter de computer wordt uitgeschakeld, gaat de geheugeninhoud verloren, terwijl een schijvenbestand gewoon op schijf blijft staan.

Een tweede voordeel is, dat het grootste beeldscherm van 64 kBytes (uit SCREEN 7 of SCREEN 8) wel in zijn geheel naar een schijvenbestand kan worden geschreven, maar niet naar een array. De array maakt deel uit van het BASIC-programma en voor dat programma, inklusief de ruimte voor variabelen en arrays, is slechts 32 kBytes vrije geheugenruimte beschikbaar. Bij het kopiëren naar een array zullen we dus altijd beperkt zijn tot een gedeelte van het video geheugen, terwijl bij schijven het hele beeldscherm kan worden weggeschreven. Het formaat van het COPY statement waarmee het videogeheugen naar schijf wordt geschreven is:

```
COPY (x1,y1)-(x2,y2)[,<SRC>] TO "[<dev>:]<fnam>"
```

Hierin zijn *x1*, *y1*, *x2*, *y2* en *SRC* al bij de voorgaande COPY statements behandeld. Het verschil zit hem in het tweede deel van het statement. "fnam" wil zeggen bestandsnaam, inklusief een eventuele bestandsnaam-uitbreiding die van de naam is gescheiden door een punt. Het is toegestaan om naar een bestand op een andere dan de aktieve schijf te schrijven. In dat geval moet de aanduiding van die andere schijf in parameter *<dev>* worden opgegeven. Stel, dat we onder SCREEN 8 het beeldscherm uit geheugenpagina 1 naar een bestand

met de naam PLAATJE.VID op schijf C willen schrijven, dan zal het statement er als volgt uit zien:

```
COPY (0,0)-(255,211),1 TO "C:PLAATJE.VID"
```

Met het volgende statement kunnen we later beeldschermen vanuit schijvenbestanden terugladen:

```
COPY "[<dev>:]<fnam>"[,<dir>] TO (X3,Y3)[,<dest>[,<log>]]
```

Dit statement moet u nu onmiddellijk kunnen begrijpen. Alle parameters zijn al in vorige statements behandeld. Willen we het gekopieerde beeldscherm van het vorige voorbeeld onder SCREEN 8 naar pagina 0 in het videogeheugen schrijven, dan ziet het statement er als volgt uit:

```
COPY "C:PLAATJE.VID" TO (0,0),0
```

We krijgen nu een exakte kopie van het oorspronkelijk gekopieerde beeldscherm, nu echter in een andere beeldschermpagina. De volgende programma's laten het kopiëren van en naar schijvenbestanden nog eens zien:

```
1000 X1=0: Y1=0
1010 X2=50: Y2=20
1020 X3=100: Y3=100
1030 SCREEN 5
1040 OPEN "GRP:" AS #1
1050 PRESET(10,10)
1060 PRINT #1,"COPY"
1070 FOR I=1 TO 500: NEXT I
1080 COPY (X1,Y1)-(X2,Y2) TO "A:VIDEO.SCN"
1090 FOR R=0 TO 3
1100 COPY "A:VIDEO.SCN",R TO (X3,Y3)
1110 FOR I=1 TO 500: NEXT I
1120 NEXT R
1130 GOTO 1130
```

```
1000 X1=0: Y1=0
1010 X2=50: Y2=20
1020 X3=100: Y3=100
1030 SCREEN 5
1040 SET PAGE 0,0
1050 OPEN "GRP:" AS #1
1060 PRESET(10,10)
1070 PRINT #1,"COPY"
1080 FOR I=1 TO 500: NEXT I
1090 COPY (X1,Y1)-(X2,Y2) TO "A:VIDEO.SCN"
1100 FOR R=0 TO 3
1110 COPY "A:VIDEO.SCN",R TO (X3,Y3),1
1120 FOR I=1 TO 500: NEXT I
1130 NEXT R
1140 SET PAGE 1,0
1150 GOTO 1150
```

2.4 Van array naar schijvenbestand en terug

Het laatste nieuwe kopieer-statement is erg eenvoudig. Het stelt ons in staat om een met beeldscherminformatie gevulde numerieke array te kopiëren naar een bestand op schijf. Natuurlijk is er ook een statement dat precies in tegengestelde richting werkt: van een schijvenbestand naar een numerieke array. Het is wel zaak om te zorgen, dat de array bij het teruglezen van een schijvenbestand groot genoeg is om het hele bestand te bevatten. Ik mag u nog eens op paragraaf 2.2 wijzen, waar werd uitgelegd hoe de benodigde grootte van een array kan worden bepaald. Het formaat van de statements is:

COPY <*array*> TO "[<*dev*>:]<*fnam*>"

COPY "[<*dev*>:]<*fnam*>" TO <*array*>

Alle parameters zijn al in de vorige paragrafen behandeld en zullen daarom hier niet nog eens worden uitgelegd.

HOOFDSTUK 3

EEN SCHERM VOL KLEUREN

De nieuwe en uitgebreide BASIC statements, waarmee het beeldscherm van de MSX2 kan worden bestuurd zijn, zoals uit de vorige hoofdstukken bleek, zeer talrijk en geven de programmeur een bijna eindeloze hoeveelheid mogelijkheden. Zoals echter nu zal blijken, zijn er nog heel veel andere nieuwe mogelijkheden. In dit hoofdstuk zullen we zien welke kleurmogelijkheden er onder MSX2 zijn bijgekomen.

Achtereenvolgens zullen we zien, hoe kleuren worden gemaakt, welke kleurmogelijkheden we per schermmode hebben en welke kleurmogelijkheden er voor sprites zijn bijgekomen.

3.1 Rood, groen en blauw mengen

Iedere kleurentelevisie, en zo ook onze kleurenmonitor, kent in principe slechts drie kleuren: rood, groen en blauw. Door die kleuren met een bepaalde intensiteit naar het beeldscherm te stralen, ontstaan kleurmengsels die door ons oog als bijvoorbeeld geel, bruin en paars worden gezien. Wanneer alledrie de kleuren, rood, groen en blauw helemaal niet worden uitgestraald, zien wij de "kleur" van het beeldscherm; zwart dus. Worden nu de kleuren rood, groen en blauw op maximale intensiteit naar het beeldscherm gestraald, dan zien wij de "kleur" wit.

Zou je nu de kleur rood maximaal uitstralen en de kleuren groen en blauw helemaal niet uitstralen, dan krijg je de kleur rood. Straal je de kleuren rood en blauw maximaal uit en groen niet, dan ontstaat de kleur paars. Voor de kleuren groen en rood op maximaal en blauw op nul ontstaat de kleur donker geel. Met groen en blauw op maximum straling en geen rode straling ontstaat een blauwgroene kleur.

Als er geen andere mogelijkheden waren dan niets uitstralen of maximaal uitstralen, dan zouden we alleen de kleuren zwart, rood, groen,

blauw, paars, donkergeel, blauwgroen en wit kunnen maken. Uit onze ervaring met MSX1 weten we echter, dat er veel meer kleuren mogelijk zijn. Immers, we hadden daar de mogelijkheid om één van de volgende kleuren te kiezen:

0	transparant	**8**	rood
1	zwart	**9**	lichtrood
2	groen	**10**	donkergeel
3	lichtgroen	**11**	lichtgeel
4	donkerblauw	**12**	donkergroen
5	blauw	**13**	paars
6	donkerrood	**14**	grijs
7	lichtblauw	**15**	wit

Hoe is het nu mogelijk om meer kleuren te maken? Door tussen 'helemaal niets' en 'maximaal uitstralen' van de kleuren rood, groen en blauw een aantal intensiteiten te definiëren. In onze MSX computers (zowel MSX1 als MSX2) zijn alledrie de kleuren rood, groen en blauw in acht intensiteiten (van 0 tot en met 7) uit te stralen, variërend van niets tot maximaal. We kunnen dus een klein beetje rood kombineren met een beetje groen en een beetje blauw. Zo zal de kleur donkerblauw ontstaan door 1 deel rood en 1 deel groen te mengen met 7 delen blauw. Lichtblauw zal ontstaan uit 2 delen rood, 6 delen groen en 7 delen blauw, enz.

U ziet, er zijn met acht intensiteiten per kleur rood, groen en blauw heel veel kombinaties te maken, om precies te zijn: 8 keer 8 keer 8 is 512 verschillende kleuren. Onder MSX1 werd uit die 512 verschillende kleuren door de fabrikant een keuze gemaakt, die niet door ons was te beïnvloeden (zie de voorgaande kleurentabel). Onder MSX2 hebben wij echter de mogelijkheid gekregen de standaard door de fabrikant gedefinieerde kleuren te beïnvloeden. Laten we eerst op de volgende pagina eens zien hoe de kleuren standaard zijn opgebouwd

kleur nummer	kleur naam	intensiteit rood	groen	blauw
0	transparant	0	0	0
1	zwart	0	0	0
2	groen	1	6	1
3	lichtgroen	3	7	3
4	donkerblauw	1	1	7
5	blauw	2	3	7
6	donkerrood	5	1	1
7	lichtblauw	2	6	7
8	rood	7	1	1
9	lichtrood	7	3	3
10	donkergeel	6	6	1
11	lichtgeel	6	6	3
12	donker groen	1	4	1
13	paars	6	2	5
14	grijs	5	5	5
15	wit	7	7	7

Willen we nu deze standaard kleurdefinities veranderen, dan kunnen we dat doen met het nieuwe MSX2 statement:

```
COLOR=(<kleur>,<rood>,<groen>,<blauw>)
```

Voor *<kleur>* dient u het kleurennummer van de te wijzigen kleur in te vullen (zie tabel hiervoor). Voor *<rood>*, *<groen>* en *<blauw>* kunt u de gewenste intensiteiten invullen, die mogen variëren van 0 tot en met 7. Zo kunt u kleurnummer 2 bijvoorbeeld veranderen in een soort oud-roze door het volgende statement te geven:

```
COLOR=(2,7,5,4)
```

Wanneer u nu kleurnummer 2 gebruikt zal dat de kleur "oud roze"" opleveren. Dit gaat ook op voor de oude MSX1 schermmodes, maar alleen wanneer deze schermmodes op een MSX2 computer worden gebruikt. Met het volgende programma kunt u het voorgaande uittesten.

```
1000 SCREEN 5
1010 OPEN "grp:" AS #1
1020 COLOR 1,15,4:CLS
1030 FOR I=0 TO 15
1040 COLOR 1: PRESET (10,I*8+4):PRINT#1,I;
1050 COLOR I: PRESET (50,I*8+4): PRINT #1,"******"
```

```
1060 NEXT I
1070 COLOR 1
1080 PRESET (10,148): PRINT #1,"Redefine (Y/N)?";
1090 I$=INKEY$:IF I$="" THEN 1090
1100 IF I$="N" OR I$="n" THEN END
1110 IF I$<>"Y" AND I$<>"y" THEN 1090
1120 PRESET (10,148): PRINT #1,"                ";
1130 FOR I=1 TO 15
1140 COLOR I: PRESET(100,I*8+4):PRINT #1,"R=";
1150 I$=INKEY$:IF I$="" THEN GOTO 1150 ELSE R=VAL(I$):PRINT#1,R
1160 COLOR I: PRESET(150,I*8+4):PRINT #1,"G=";
1170 I$=INKEY$:IF I$="" THEN GOTO 1170 ELSE G=VAL(I$):PRINT#1,G
1180 COLOR I: PRESET(200,I*8+4):PRINT #1,"B=";
1190 I$=INKEY$:IF I$="" THEN GOTO 1190 ELSE B=VAL(I$):PRINT#1,B
1200 COLOR=(I,R,G,B)
1210 COLOR I: PRESET (50,I*8+4): PRINT #1,"*****"
1220 NEXT I
1230 COLOR 1,15
1240 GOTO 1020
```

Het programma laat van boven naar beneden op het beeldscherm alle kleurnummers (0 tot en met 15) zien, met daarachter de op dat moment geldende kleur. Nu kunt u de kleuren her-definiëren, door de vraag "redefine?" met **Y** of y te beantwoorden. U ziet dan dat er achter de kleuren op het scherm de tekst "R =" verschijnt. Hierop kunt u een kleurintensiteit voor de kleur rood intikken. Die intensiteit kan variëren van 0 tot 7. Heeft u een intensiteit ingetikt, dan verschijnt de tekst "G =" en kunt u de intensiteit voor groen intikken. Tenslotte wordt dit nog eens herhaald voor blauw.

Na het intikken van de intensiteit voor blauw wordt het betreffende kleurvak op het beeldscherm in de nieuw gedefinieerde kleur afgedrukt, zodat u direkt het resultaat kunt zien. Op deze manier kunnen alle 15 kleurnummers van een nieuwe kleur worden voorzien en kunt u onmiddellijk zien hoe de nieuw gedefinieerde kleuren eruit zien.

In de listing op regels 1050 en 1210 ziet u dat er "*****" wordt afgedrukt. Deze tekens zijn in de listing opgenomen omdat mijn tekstverwerker niet in staat is de oorspronkelijke "zwarte blokjes", die ik in de listing had staan, af te drukken. U dient voor een goed resultaat de "*"-tekens te vervangen door zwarte blokjes. Dit bereikt u door de GRAPH-toets samen met de letter P in te drukken.

Zoals uit voorgaand programma blijkt, kunnen wij nu aan ieder kleurnummer een kleur toekennen, die we kunnen kiezen uit 512 verschillende kleuren. Om die reden wordt onder MSX2 ook niet meer van een kleurnummer gesproken, maar van een *palette-nummer*. Beschouw zo'n palette maar als het palet van een schilder, waarop hij kleuren mengt. De palette nummers en de toekenning van kleuren daaraan geldt voor alle schermmodes, behalve schermmode 8. Voor schermmode 6 bestaat er een beperking, omdat er in die mode slechts vier palettes beschikbaar zijn, de nummers 0, 1, 2 en 3. Deze vier palettes kunnen echter wel op de hiervoor beschreven manier van een kleur worden voorzien.

Een andere bijzonderheid van SCREEN 6 is, dat de randkleur een waarde mag hebben van 0 tot 3 of 16 tot 31. Wordt een waarde van 16 tot 31 gebruikt, dan wordt de randkleur uit twee palette kleuren samengesteld, en wel volgens de hierna gegeven tabel:

randkleur	palettekleur even pixels	palette kleur oneven pixels
16	0	0
17	0	1
18	0	2
19	0	3
20	1	0
21	1	1
22	1	2
23	1	3
24	2	0
25	2	1
26	2	2
27	2	3
28	3	0
29	3	1
30	3	2
31	3	3

U ziet, dat randkleur 22 resulteert in de kleuren 1 voor de even pixels en 2 voor de oneven pixels. Door de hoge resolutie van SCREEN 6 mengen beide kleuren zich voor het menselijk oog tot een egaal vlak met een kleur die afhangt van de beide palette kleuren. Het volgende programma maakt gebruik van SCREEN 6 en laat alle nu besproken mogelijkheden zien:

```
1000 SCREEN 6
1010 COLOR=(1,7,2,2)
1020 COLOR=(2,2,7,2)
1030 COLOR=(3,2,2,7)
1035 FOR R=16 TO 30
1040 COLOR 1,2,R
1043 LINE (100,50)-(200,150),1,BF
1044 LINE (200,50)-(300,150),0,BF
1045 LINE (300,50)-(400,150),3,BF
1046 FOR I=1 TO 150: NEXT I
1047 NEXT R
1050 GOTO 1050
```

Voor schermmode 8 geldt een geheel andere manier voor het aangeven van kleuren. In deze mode zijn op een scherm 256 verschillende kleuren tegelijkertijd mogelijk. Die kleuren hebben kleurnummers van 0 tot en met 255. Voor zowel de voorgrondkleur, de achtergrondkleur als de randkleur kan een kleurnummer – variërend van 0 tot en met 255 – worden opgegeven. Deze kleurnummers vormen een direkte kodering van de kleurintensiteiten van de kleuren rood, groen en blauw. Daarbij geldt, dat de kleuren rood en groen ieder mogen variëren van 0 tot 7, terwijl de kleur blauw mag variëren van 0 tot 3. Om het uiteindelijke kleurnummer te bepalen kan gebruik worden gemaakt van de volgende formule:

kleurnummer = 4 * rood + 32 * groen + blauw

In deze formule staan de woorden rood, groen en blauw voor de gewenste intensiteit van elk van deze kleuren. Indien u bijvoorbeeld het kleurnummer wilt bepalen voor een kleur die uit de intensiteiten 7, 3 en 3 (voor respektievelijk de kleuren rood, groen en blauw) bestaat, dan zal de formule als volgt werken:

kleurnummer = 4*7 + 32*3 + 3 = 28 + 96 + 3 = 127

Het volgende programmavoorbeeld laat alle mogelijke kleuren op het scherm afdrukken:

```
1000 SCREEN 8
1010 FOR I=0 TO 255
1020 LINE (I,0)-(I,211),I
1030 NEXT I
1040 GOTO 1040
```

Wanneer u het kleurenpalet heeft gewijzigd en u wilt de standaard kleuren palet weer aktiveren, dan kunt u dat doen met het statement:

```
COLOR  of  COLOR=NEW
```

Een voorbeeld van de toepassing van dit statement ziet u in het volgende programma:

```
1000 SCREEN 0
1010 COLOR 15,4,4
1020 COLOR=(15,7,0,0)
1030 LOCATE 1,1:PRINT "ROOD"
1040 FOR I=1 TO 200: NEXT I
1050 COLOR=NEW
1060 LOCATE 1,1:PRINT "WIT "
1070 FOR I=1 TO 200: NEXT I
1080 GOTO 1020
```

Volgens de standaard kleurenpalet is kleur 15 wit. In regel 1020 veranderen we die kleur echter met een COLOR= statement in rood. Vervolgens schrijven we het woord "rood" op het scherm, en zien dat de voorgrondkleur (15) inderdaad rood is geworden. Met regel 1050 wordt de oorspronkelijke kleurenpalet weer ingesteld. Vandaar dat het met regel 1060 afgedrukte woord "wit" ook weer in een witte kleur wordt afgedrukt.

Het kleuren palet wordt in het videogeheugen opgeslagen. Ook wanneer er wijzigingen in worden aangebracht, dan komen die wijzigingen in het video geheugen te staan. Wanneer we nu weten waar die palet informatie precies staat, dan zouden we een palet vanuit het video geheugen naar schijf kunnen schrijven, om deze er later (misschien in een ander programma) weer vanaf te kunnen lezen en in het video geheugen te zetten. De kleuren palet adressen in het video geheugen zijn bekend, maar variëren per ingestelde schermmode.

De startadressen variëren, maar alle paletten hebben dezelfde lengte, en wel 32 bytes (komt overeen met 20 hexadecimaal). Hier volgt een overzicht van alle startadressen, gegeven in hexadecimale kode.

SCREEN	PAGE 0	PAGE 1	PAGE 2	PAGE 3
0 (40)	&H0400	-	-	-
0 (80)	&H0F00	-	-	-
1	&H2020	-	-	-
2	&H2020	-	-	-
3	&H2020	-	-	-
4	&H2020	-	-	-
5	&H7680	&HF680	&H17680	&H1F680
6	&H7680	&HF680	&H17680	&H1F680
7	&HFA80	&H1FA80	-	-
8	&HFA80	&H1FA80	-	-

Het naar schijf schrijven van een kleuren palet gebeurt met het BSAVE statement. Willen we bijvoorbeeld het kleuren palet van schermmode 1 naar schijf schrijven, dan geven we het volgende kommando:

```
BSAVE "PALETTE1.BIN",&H2020,&H2040,S
```

Vergeet vooral die laatste letter S niet. Die letter zorgt er namelijk voor, dat het schermgeheugen wordt geselekteerd in plaats van het werkgeheugen.
Het bestand kan weer van schijf worden teruggelezen met het statement:

```
BLOAD "PALETTE1.BIN",S
```

Wilt u vervolgens de kleuren uit de zojuist ingelezen kleurenpalet aktiveren, dan dient u het volgende statement uit te voeren:

```
COLOR=RESTORE
```

Hiermee zijn alle nieuwe COLOR statements en uitbreidingen van de bestaande COLOR statements aan de orde geweest. Er valt echter nog wel meer over kleuren te zeggen en dat betreft dan de kleuren van sprites.

3.2 Veelkleurige sprites

Onder MSX1 kon iedere sprite slechts één kleur hebben. Die kleur kan met het PUT SPRITE statement aan de sprite worden toegekend. Toch is het ook onder MSX1 mogelijk om een sprite voor het menselijk oog uit meerdere kleuren te laten bestaan. Dit is te verwezelijken door meerdere sprites die elk een andere kleur hebben elkaar te laten overlappen. Hoewel dit geen nieuwe mogelijkheid is, wordt daar over het algemeen weinig aandacht aan besteed, waardoor de bestaande mogelijkheden vaak niet volledig worden gebruikt.

Een voorbeeld van een sprite, die uit drie elkaar overlappende sprites bestaat, wordt in het volgende programma gegeven:

```
1000 'sprite 1
1010 DATA 7,31,48,96,103,199,204,204
1020 DATA 204,204,199,99,48,24,15,3
1030 DATA 224,248,12,6,230,227,51,51
1040 DATA 51,51,227,198,12,24,240,192
1050 'sprite 2
1060 DATA 0,0,15,31,24,56,48,48
1070 DATA 48,48,56,28,15,7,0,0
1080 DATA 0,0,240,248,24,28,12,12
1090 DATA 12,12,28,56,240,224,0,0
1100 'sprite 3
1110 DATA 0,0,0,0,0,0,3,3
1120 DATA 3,3,0,0,0,0,0,0
1130 DATA 0,0,0,0,0,0,192,192
1140 DATA 192,192,0,0,0,0,0,0
1150 SCREEN 5,3: S$="": RESTORE 1010
1160 FOR I=1 TO 32
1170 READ S: S$=S$+CHR$(S)
1180 NEXT I
1190 SPRITE$(0)=S$
1200 S$="": RESTORE 1060
1210 FOR I=1 TO 32
1220 READ S: S$=S$+CHR$(S)
1230 NEXT I
1240 SPRITE$(1)=S$
1250 S$="": RESTORE 1110
1260 FOR I=1 TO 32
1270 READ S: S$=S$+CHR$(S)
```

```
1280 NEXT I
1290 SPRITE$(2)=S$
1295 X=100: Y=100
1300 PUT SPRITE 30,(X,Y),8,0
1310 PUT SPRITE 29,(X,Y),7,1
1320 PUT SPRITE 28,(X,Y),11,2
1330 I$=INKEY$:IF I$="" THEN GOTO 1330
1340 IF I$=CHR$(29) THEN X=X-1: IF X=<0 THEN X=240
1350 IF I$=CHR$(28) THEN X=X+1: IF X=>240 THEN X=0
1360 IF I$=CHR$(30) THEN Y=Y-1: IF Y=<0 THEN Y=200
1370 IF I$=CHR$(31) THEN Y=Y+1: IF Y=>200 THEN Y=0
1380 GOTO 1300
```

Na het definiëren van de sprites in de regels 1000 tot en met 1290, worden met regel 1295 de koördinaten bepaald waarop de drie sprites moeten worden geplaatst. In de regels 1300, 1310 en 1320 ziet u dat de drie sprites precies over elkaar heen op het scherm worden geplaatst. De eerste sprite krijgt kleurnummer 8, de tweede kleurnummer 7 en de derde nummer 11. Met de regels 1330 tot en met 1380 worden de koördinaten gewijzigd, afhankelijk van de ingedrukte kursorbesturingstoetsen. Daarna worden de sprites weer exakt over elkaar heen afgebeeld. U zult zien, dat er voor het oog een enkele sprite van drie kleuren ontstaat.

Onder schermmodes SCREEN 4 tot en met SCREEN 8 is het mogelijk om sprites van kleur te veranderen, zonder de andere parameters, zoals die in het PUT SPRITE statement voorkomen, te wijzigen. Daartoe kan het volgende statement worden gebruikt:

COLOR (<*transp*>) =<*kleur*>

Achter dit COLOR statement moet het transparantnummer, waarop de te kleuren sprite staat, worden opgegeven. Achter het gelijkteken mag de kleur worden opgegeven. Deze kleur mag uit de volgende elementen bestaan:

0 - 15 Een nummer uit het kleuren palet, variërend van 0 tot 15.

+32 Hierbij mag een waarde 32 worden opgeteld. Dit houdt in, dat een botsing van deze sprite met een andere sprite niet meer kan worden gedetekteerd, zelfs al is het statement ON SPRITE GOSUB uitgevoerd.

+64 Net als +32, echter de sprite wordt bovendien onzichtbaar, behalve het deel dat op dezelfde hoogte ligt als een sprite in een vóórliggend transparant. Indien de onzichtbare sprite een andere sprite overlapt, dan wordt op de elkaar overlappende pixels een logische AND-bewerking van de kleuren losgelaten. Zou de onzichtbare sprite de kleur 13 hebben en de overlappende sprite de kleur 7, dan zou de resulterende kleur van de elkaar overlappende pixels zijn: 13 AND 7 = 1101 AND 0111 = 0101 = 5.

Deze extra kleurmogelijkheden gelden ook voor het kleurnummer in het PUT SPRITE statement, wanneer er onder schermmodes 4 tot en met 8 wordt gewerkt.

Een belangrijke vernieuwing van de kleurmogelijkheden van sprites biedt het statement:

```
COLOR SPRITE$(<transp>)=<string>
```

Een sprite bestaat uit 8 of 16 horizontale lijnen, die kunnen worden verdubbeld wanneer er van vergrote sprites sprake is (dit wordt bepaald met het SCREEN statement). Met COLOR SPRITE$ kan aan iedere afzonderlijke horizontale lijn een aparte kleur worden toegekend. Daartoe dient in de *<string>* per horizontale lijn een kode te worden opgenomen. Die kode komt overeen met het palet nummer van de gewenste kleur. Stel dat we een kleine sprite (van acht horizontale lijnen) willen inkleuren met achtereenvolgens de kleuren 1, 2, 3, 4, 5, 6, 7 en 8. We dienen dan een string samen te stellen als volgt:

```
CHR$(1)+CHR$(2)+CHR$(3)+.....+CHR$(8)
```

Het is ook mogelijk om deze string aan een variabele toe te kennen, waarna die stringvariabele in het COLOR SPRITE$ statement kan worden opgenomen. Een manier is, deze string te laten voorafgaan door een LET statement. Een voorbeeld hiervan ziet u in het volgende programma op regel 1080. Wanneer u dit programma intikt en uitvoert, zult u zien dat er een vlaggetje op het scherm verschijnt, waarvan de eerste drie horizontale lijnen rood zijn (kleur nummer 8), de volgende twee lijnen wit (kleurnummer 15) en de laatste drie lijnen blauw (kleurnummer 4).

```
1000 DATA 255,255,255,255,255,255,255,255
1010 S$=""
1020 FOR I=1 TO 8
1030 READ S: S$=S$+CHR$(S)
1040 NEXT I
1050 COLOR 1,10,10
1060 SCREEN 5,1
1070 SPRITE$(0)=S$
1080 C$=CHR$(8)+CHR$(8)+CHR$(8)+CHR$(15)+CHR$(15)+CHR$(4
)+CHR$(4)+CHR$(4)
1090 PUT SPRITE 0,(I,100),15,0
1100 COLOR SPRITE$(0)=C$
1110 I$=INKEY$: IF I$="" THEN GOTO 1110
1120 IF I$=CHR$(28) THEN I=I+1: IF I>240 THEN I=0
1130 IF I$=CHR$(29) THEN I=I-1: IF I<0 THEN I=240
1140 GOTO 1090
```

Een andere manier om de kleuren aan een sprite toe te kennen is: de kleurnummers in een DATA statement zetten. De DATA regel wordt dan uitgelezen en de gelezen waarde wordt als stringwaarde aan een stringvariabele toegekend. Dit is in het volgende programma toegepast. Dit programma is in feite een uitbreiding van het programma dat aan het begin van deze paragraaf werd gegeven. Er worden weer drie elkaar overlappende sprites gebruikt, maar nu hebben de individuele horizontale lijnen van iedere sprite een andere kleur. U zult zien, dat er op deze manier een zeer groot aantal kleuren mogelijk is.

```
1000 'sprite 1
1010 DATA 7,31,48,96,103,199,204,204
1020 DATA 204,204,199,99,48,24,15,3
1030 DATA 224,248,12,6,230,227,51,51
1040 DATA 51,51,227,198,12,24,240,192
1050 'sprite 2
1060 COLOR 1,15
1070 DATA 0,0,15,31,24,56,48,48
1080 DATA 48,48,56,28,15,7,0,0
1090 DATA 0,0,240,248,24,28,12,12
1100 DATA 12,12,28,56,240,224,0,0
1110 'sprite 3
1120 DATA 0,0,0,0,0,0,3,3
1130 DATA 3,3,0,0,0,0,0,0
```

```
1140 DATA 0,0,0,0,0,0,192,192
1150 DATA 192,192,0,0,0,0,0,0
1160 SCREEN 5,3: S$="": RESTORE 1010
1170 FOR I=1 TO 32
1180 READ S: S$=S$+CHR$(S)
1190 NEXT I
1200 SPRITE$(0)=S$
1210 S$="": RESTORE 1070
1220 FOR I=1 TO 32
1230 READ S: S$=S$+CHR$(S)
1240 NEXT I
1250 SPRITE$(1)=S$
1260 S$="": RESTORE 1120
1270 FOR I=1 TO 32
1280 READ S: S$=S$+CHR$(S)
1290 NEXT I
1300 SPRITE$(2)=S$
1310 RESTORE 1510
1320 C0$="":FORI=1TO16:READC:C0$=C0$+CHR$(C):NEXTI
1330 RESTORE 1530
1340 C1$="":FORI=1TO16:READC:C1$=C1$+CHR$(C):NEXTI
1350 RESTORE 1550
1360 C2$="":FORI=1TO16:READC:C2$=C2$+CHR$(C):NEXTI
1370 X=100: Y=100
1380 PUT SPRITE 30,(X,Y),,0
1390 COLOR SPRITE$(30)=C0$
1400 PUT SPRITE 29,(X,Y),,1
1410 COLOR SPRITE$(29)=C1$
1420 PUT SPRITE 28,(X,Y),,2
1430 COLOR SPRITE$(28)=C2$
1440 I$=INKEY$:IF I$="" THEN GOTO 1440
1450 IF I$=CHR$(29) THEN X=X-1: IF X=<0 THEN X=240
1460 IF I$=CHR$(28) THEN X=X+1: IF X=>240 THEN X=0
1470 IF I$=CHR$(30) THEN Y=Y-1: IF Y=<0 THEN Y=200
1480 IF I$=CHR$(31) THEN Y=Y+1: IF Y=>200 THEN Y=0
1490 GOTO 1380
1500 'kleuren sprite 0
1510 DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,7,8
1520 'kleuren sprite 1
1530 DATA 0,0,8,9,2,3,4,5,6,7,8,9,10,11,0,0
1540 'kleuren sprite 2
1550 DATA 0,0,0,0,0,0,10,11,12,13,0,0,0,0,0,0
```

Bij het toekennen van kleur met behulp van het COLOR SPRITE$ statement kan bij het kleurnummer, net als dat hiervoor al werd uitgelegd, een waarde worden opgeteld. Ook nu kan een waarde van 32 of 64 worden bijgeteld. De betekenis is net als hiervoor werd toegelicht. Er kan echter ook nog een waarde 128 worden bijgeteld.

Indien de waarde 128 wordt bijgeteld, dan heeft dat de volgende betekenis. De beeldlijn, waaraan deze waarde is toegevoegd, zal 32 pixels verder naar links worden afgebeeld, tenzij de horizontale koördinaat met een negatief getal werd aangegeven.

HOOFDSTUK 4

TEKENEN MET MSX2

De uit MSX1 bekende teken-instrukties hebben allemaal een uitbreiding of op zijn minst een veranderde funktie gekregen. In dit hoofdstuk zullen we zien wat die wijzigingen precies inhouden en welke mogelijkheden we met die gewijzigd statements hebben gekregen. Het gaat daarbij om de volgende statements:

```
DRAW
PSET
PRESET
LINE
CIRCLE
PAINT
POINT
```

U ziet, nieuwe statements zijn er niet bij. Het zal echter duidelijk zijn, dat een hogere schermresolutie en een grotere varieteit aan kleuren op zijn minst gewijzigde formaten nodig maakt. Hier volgt een kort overzicht van de belangrijkste wijzigingen per statement.

DRAW	Aanpassing van de kleuren voor SCREEN 6 en 8. Gewijzigde koördinaten voor SCREEN 5, 6, 7 en 8.
PSET	Als DRAW, maar tevens uitgebreid met een logische bewerking op de kleuren.
PRESET	Als PSET
LINE	Als PSET
CIRCLE	Als DRAW
PAINT	Als DRAW
POINT	Gewijzigde koördinaten voor SCREEN 5, 6, 7 en 8. Bovendien zal deze funktie onder SCREEN 6 en 8 de daaronder te gebruiken kleurnummers opleveren.

Een belangrijke verbetering van het PAINT statement is bovendien, dat nu een vlak in iedere gewenste kleur kan worden geverfd, ongeacht de kleur van de begrenzende lijn. Bij de behandeling van PAINT zullen we dit nader toelichten.

4.1 Gewijzigde teken-statements

De enige wijzigingen in het DRAW-statement zijn, dat onder schermmodes 6 en 7 de horizontale koördinaat maximaal 511 mag zijn en dat onder schermmodes 5, 6, 7 en 8 de vertikale koördinaat maximaal 211 mag zijn. Voor de kleur geldt, dat onder schermmodes 2, 3, 4, 5 en 7 een kleurnummer tussen 0 en 15 mag worden gekozen, onder schermmode 6 een kleurnummer tussen 0 en 3 en onder schermmode 8 een kleurnummer van 0 tot en met 255.

Het volledige formaat van het DRAW statement is:

DRAW <*string*> *of* DRAW <*stringvariabele*>

De string of stringvariabele wordt opgebouwd uit een of meer subkommando's. Met deze subkommando's kan een bepaalde richting worden aangegeven, kan een koördinaat waarnaartoe een lijn moet worden getekend worden aangegeven en kan een kleur worden opgegeven, waarin de lijn moet worden getekend. Verder kan nog worden opgegeven een schaal en een hoek waaronder moet worden getekend.

Voor een uitgebreide behandeling van het DRAW statement verwijs ik naar MSX leerboek deel 3. We zullen alles hier niet nog eens herhalen. Zie voor een uitgebreide behandeling van de kleuren het vorige hoofdstuk. Daar werden alle kleurmogelijkheden uitvoerig besproken. We gaan er daarom nu van uit, dat de kleurmogelijkheden bekend zijn.

Voor de statements PSET en PRESET gelden dezelfde uitbreidingen als voor het DRAW statement voor zover het de koördinaten en de kleuren betreft. Aan de statements PSET en PRESET is echter een parameter toegevoegd. Met deze extra parameter kan een logische bewerking worden opgegeven op de kleuren van het te kleuren pixel. De formaten van beide statements zijn:

PSET [STEP](<*HOR*>,<*VER*>)[,[<*KLEUR*>],<*LOG*>]

PRESET [STEP](<*HOR*>,<*VER*>)[,[<*KLEUR*>],<*LOG*>]

<LOG>:

Dit is een logische operator. Er is keuze uit één van de volgende mogelijkheden:

<log> = PSET

Het te schrijven pixel ondergaat geen logische bewerking. Het pixel wordt gewoon in de opgegeven kleur geschreven.

<log> = PRESET

Onder SCREEN 5 en 7 krijgt het pixel de kleur 15 minus het opgegeven kleurnummer. Voorbeeld:
Opgegeven kleur: 8
Resulterende kleur: 15 – 8 = 7

Onder SCREEN 6 krijgt het pixel de kleur met het nummer 3 minus het opgegeven kleurnummer. Voorbeeld:
Opgegeven kleur: 3
Resulterende kleur: 3 – 3 = 0

Onder SCREEN 8 krijgt het pixel de kleur met het nummer 255 minus het opgegeven kleurnummer. Voorbeeld:
Opgegeven kleur: 44
Resulterende kleur: 255 – 44 = 211

<log> = AND

Op de kleurnummers van de opgegeven kleur en van de kleur van het pixel waar naartoe wordt geschreven wordt de logische operatie AND verricht. Het resultaat levert het kleurnummer van het te schrijven pixel op. Voorbeeld:
Opgegeven kleur: 9
Kleur van het pixel waarnaartoe moet worden geschreven: 3

9 binair =	1001	
3 binair =	0011	AND
Resultaat	0001	= 1 (decimaal)

<log> = OR

Op de kleurnummers van de opgegeven kleur en de kleur van het pixel waar naartoe wordt geschreven wordt de logische operatie OR verricht. Het resultaat levert het kleurnummer van het geschreven pixel op.

Voorbeeld:

9 binair = 1001
3 binair = 0011 OR
Resultaat 1011 = 11 (decimaal)

<log> = XOR

Op de kleurnummers van de opgegeven kleur en de kleur van het pixel waar naartoe wordt geschreven wordt de logische operatie XOR verricht. Het resultaat levert het kleurnummer van het geschreven pixel op. Voorbeeld:

9 binair = 1001
3 binair = 0011 XOR
Resultaat 1010 = 10 (decimaal)

<log> = TPSET, TPRESET, TAND, TOR of TXOR

Alle zojuist behandelde logische bewerkingen kunnen ook worden voorafgegaan door de letter T. Ze hebben dezelfde werking als hiervoor beschreven, alleen zullen pixels met de kleur 0 (Transparant) niet worden geschreven.

Het volgende statement dat we zullen behandelen is het LINE statement. De wijzigingen in dit statement, ten opzichte van hetzelfde statement onder MSX1, zijn dezelfde als de wijzigingen die voor het PSET statement gelden. Het formaat van het LINE statement is:

`LINE [[STEP](x1,y1)]-[STEP](x2,y2)[,[<k>][,[B[F]][,<log>]]]`

x1 en *x2* mogen voor SCREEN 2, 3, 4, 5 en 8 een waarde hebben van 0 tot en met 255. 0 is het meest linkse pixel op het scherm en 255 het meest rechtse. Voor SCREEN 6 en 7 gelden de waarden 0 tot en met 511.

y1 en *y2* mogen voor SCREEN 2, 3 en 4 een waarde hebben van 0 tot en met 191. 0 is het bovenste pixel en 191 is het onderste. Voor SCREEN 5, 6, 7 en 8 mogen de waarden 0 tot en met 211 worden opgegeven.

Voor <*k*> en voor <*log*> gelden dezelfde regels als voor de kleuren en de logische bewerking zoals die bij het PSET statement werden beschreven (zie aldaar). Het volgende programma laat zien hoe een zogenaamde "bar-chart" er uit gaat zien, wanneer van de nieuwe MSX2 mogelijkheden gebruik wordt gemaakt.

```
5 OPEN "grp:" AS #1
10 Y=150:COLOR 1,15: SCREEN 7
11 AR=4:AB=7:MW=200
12 GOSUB 500
15 FOR X=110 TO 360 STEP 50
16 K=(X-10)/25-1
17 H=RND(-TIME)*100+20
30 GOSUB 1000
35 NEXT X
40 GOTO 40
500 '****************************
510 '* achtergrond tekenen.     *
520 '* aantal rijen:  AR        *
530 '* max. waarde :  MW        *
535 '* aantal bars :  AB        *
540 '****************************
550 '
560 LINE (40,10)-(40,110)
570 FOR Y=10 TO 110 STEP 20
580 PSET (10,Y):PRINT #1,INT(MW-(Y-10)/100*MW)
590 LINE (40,Y)-(340,Y)
600 NEXT Y
610 LINE (340,10)-(340,110)
620 FOR Y=10 TO 110 STEP 20
630 LINE (340,Y)-(340+AR*20,Y+AR*10)
640 NEXT Y
650 LINE (40,110)-(40+AR*20,110+AR*10)
660 LINE -STEP(300,0)
665 LINE -STEP(0,-100)
670 RETURN
1000 '**************************
1010 '* Maak BAR.               *
1020 '* positie :  X,Y          *
1030 '* Hoogte  :  H            *
1040 '* Kleur    :  K (3,5,7,9, *
1045 '*                11,13,15)*
1050 '**************************
```

```
1060 '
1070 LINE (X-8,Y)-(X+8,Y-H),K,BF
1080 LINE (X-8,Y)-(X-18,Y-5),K
1090 LINE -STEP(0,-H),K
1100 LINE -STEP(16,0),K
1110 LINE -STEP(10,4),K
1120 PAINT(X-9,Y-6),K-1,K
1130 RETURN
```

Met de programmaregels 5 tot en met 40 worden de variabelen opgevraagd, die de lengte en waarden van de staven bepalen. Bovendien wordt vanuit dit programma de subroutine die op regel 500 start aangeroepen. Met de regels 500 tot en met 670 wordt de achtergrond voor de grafiek gemaakt. Ook wordt vanuit deze subroutine de subroutine die op regel 1000 start aangeroepen. Deze routine tekent een staaf in de grafiek.

Het programma werkt onder SCREEN 7. Dit wil zeggen dat gebruik wordt gemaakt van de maximale resolutie (512 bij 212 beeldpuntjes) met 16 verschillende kleuren. Tikt u het programma maar eens in. U zult versteld staan van de beeldkwaliteit. De subroutines van regel 500 en 1000 kunt u in uw eigen programma's toepassen. In uw programma hoeft u alleen maar voor het vullen van de variabelen te zorgen, de subroutines doen zelf de rest.

Het volgende statement dat we zullen behandelen is het CIRCLE statement. Het formaat van dit statement is:

CIRCLE [STEP] (*x,y*),*r*,[[*k*][,[*bh*][,[*eh*][,*af*]]]]

Voor de koördinaten van het middelpunt *x* en *y* gelden dezelfde regels als voor de koördinaten van het PSET statement. Ook de toegestane kleuren zijn precies gelijk aan de kleuren zoals opgegeven bij het PSET statement. Dit zijn de enige wijzigingen ten opzichte van MSX1. Wel zult u er rekening mee moeten houden dat een cirkel, getekend op een scherm van lage resolutie, een andere afplattingsfaktor nodig heeft dan een cirkel die op een hoge resolutie scherm (SCREEN 6 of 7) wordt geschreven. Een afplattingsfaktor die onder bijvoorbeeld SCREEN 5 een perfekte cirkel geeft, zal onder SCREEN 6 een ei opleveren. Hiermee bedoel ik, dat de cirkel vertikaal veel langer zal zijn dan dat deze horizontaal breed is.

Het volgende programma laat een voorbeeld zien van het CIRCLE statement. Het programma maakt een zogenaamde *"pie-chart"* op het hoge resolutie scherm (SCREEN 7).

```
5 COLOR 1,3,3
10 OPEN "grp:" AS #1:SCREEN 7
20 R=80:U=0:A=.7
30 RESTORE
40 FOR I=0 TO 300 STEP 60
50 IF I>240 THEN U=1 ELSE U=0
60 X=256:Y=100
70 BH=I:EH=I+60:K=I/60+4
80 READ T$
90 GOSUB 1000
100 NEXT I
110 GOTO 110
120 DATA guldens,kwartjes,dubbeltjes,stuivers,centen,rij
ksdaalders
1000 '*******************************
1010 '* cirkelsegment tekenen:      *
1020 '* middelpunt   - X en Y       *
1030 '* straal       - R           *
1040 '* beginhoek    - BH (graden) *
1050 '* eindhoek     - EH (graden) *
1060 '* kleur        - K           *
1070 '* afplatting   - A           *
1080 '* uitgelicht   - U           *
1090 '* tekst        - T$          *
1100 '*******************************
1110 '
1120 IF BH=0 THEN BH=.1
1130 X1=X:Y1=Y
1140 M=6.283185#-(BH+(EH-BH)/2)/360*6.283185#
1150 IF U=1 THEN X1=X+8*COS(M):Y1=(Y+8*SIN(M))
1160 X2=X1+R/2*COS(M):Y2=Y1+R/2*SIN(M)
1170 X3=X1+R*1.2*COS(M):Y3=Y1+R*1.2*(SIN(M)*A)-4
1180 BH=BH/360*6.283185#
1190 EH=EH/360*6.283185#
1200 CIRCLE (X1,Y1),R,K,-BH,-EH,A
1210 PAINT (X2,Y2),K
1220 IF X3<X THEN X3=X3-8*LEN(T$)
1230 PSET (X3,Y3)
1240 PRINT #1,T$
1250 RETURN
```

De regels 5 tot en met 120 bevatten de variabelen, die bepalen uit hoeveel segmenten de cirkel zal bestaan en hoe groot de verschillende segmenten moeten worden. Ook wordt aangegeven welk segment uit de cirkel moet worden gelicht. Tenslotte wordt nog opgegeven welke tekst bij ieder segment moet komen te staan. De subroutine vanaf regel 1000 tekent de "pie-chart" overeenkomstig de opgegeven variabelen.

Dit voorbeeldprogramma laat, evenals het vorige, heel duidelijk zien, dat horizontaal naast elkaar gelegen pixels elke gewenste kleur mogen hebben. Onder de oude MSX1 schermen mochten per 8 naast elkaar gelegen pixels slechts twee verschillende kleuren worden getekend. Probeerde men toch meer dan twee kleuren naast elkaar te tekenen, dan werden de voorgaande pixels *(modulo 8)* allemaal in de laatste tekenkleur gekleurd. De nieuwe schermmodes van MSX2 (SCREEN 5, 6, 7 en 8) hebben deze beperking niet meer.

De zojuist genoemde verbetering is ook van het grootste belang voor de werking van het PAINT statement, zoals we zodadelijk zullen zien. Het formaat van het PAINT statement is:

`PAINT [STEP] (`*x,y*`) [,[<`*verfkl*`>][,<`*randkl*`>]]`

Voor de koördinaten *x* en *y* en de kleuren *<verfkl>* en *<randkl>* gelden dezelfde regels als voor het PSET statement. Met PAINT kunnen we een op het scherm getekende afbeelding vullen met de verfkleur. De omhulling van de getekende afbeelding moet in de randkleur zijn getekend. Is dat niet het geval, dan zal het hele beeldscherm worden geverfd met de verfkleur.

Het volgende programma laat het gebruik van PAINT zien. Er worden eerst een aantal cirkels getekend in 15 verschillende kleuren. De binnenste cirkel wordt in de kleur zwart (1) getekend. Aan deze cirkels kunt u zien, dat ieder pixel iedere gewenste kleur mag hebben. Nadat alle cirkels zijn getekend blijft er een vlak in het midden open. Dat vlak wordt door de zwarte cirkel omlijnd. Met het PAINT statement kunnen we nu dat vlak in iedere gewenste kleur verven. In het programma wordt gekozen voor kleurnummer 8, terwijl tevens wordt vermeld dat de rand van het vlak kleurnummer 1 heeft. Hierdoor weet het PAINT statement precies tot hoever het vlak moet worden ingekleurd.

```
1000 SCREEN 7: COLOR 4,3,2
1010 FOR I=31 TO 17 STEP -1
1020 CIRCLE (100,100),I,I-16,,,0.65
1030 NEXT I
1040 PAINT (100,100),8,1
1050 GOTO 1050
```

We kozen hier voor een afplattingsfaktor van 0.65. Onder SCREEN 7 levert dit een vrijwel ronde cirkel op. Indien u het programma zou wijzigen, zodat bijvoorbeeld SCREEN 5 wordt gebruikt, dan dient u tevens de afplattingsfaktor te wijzigen en wel in 1.35, om weer een ronde cirkel te krijgen.

Tenslotte is er nog een funktie, die onder MSX2 een wijziging heeft ondergaan ten opzichte van MSX1: de funktie POINT. Het formaat van deze funktie is:

```
K = POINT (x,y)
```

Deze funktie levert het kleurnummer op van het pixel dat op de koördinaten *x* en *y* ligt. *x* en *y* mogen dezelfde waarden hebben die ook al werden genoemd voor de koördinaten van het statement PSET. Het resultaat van de funktie is een kleurnummer. Welke mogelijke kleurnummers worden verkregen, hangt af van de gebruikte schermmode. Onder SCREEN 6 kan het resultaat variëren van 0 tot en met 3. Onder SCREEN 8 kan het resultaat variëren van 0 tot 255. Onder alle andere schermmodes kan het resultaat variëren van 0 tot 15.

HOOFDSTUK 5

DE VIDEOPROCESSOR EN HET VIDEOGEHEUGEN

Tot nu toe hebben we, waarschijnlijk zonder het te beseffen, voor alle voorbeeldprogramma's gebruik gemaakt van het video geheugen. Dit geheugen wordt gekontroleerd en bestuurd door een aparte processor, de videoprocessor. In MSX2 computers is een nieuw type videoprocessor toegepast. Deze nieuwe videoprocessor heeft veel meer mogelijkheden dan de oude MSX1 videoprocessor. Alles wat met de MSX1 videoprocessor mogelijk was, is ook met de videoprocessor uit de MSX2 mogelijk, maar, zoals we in de voorgaande hoofdstukken hebben gezien, er zijn daarnaast nog een hele reeks nieuwe mogelijkheden beschikbaar.

In dit hoofdstuk zullen we zien hoe de videoprocessor werkt en hoe deze het videogeheugen benut. Daarbij zullen we de volgende BASIC statements en funkties tegen komen:

BASE - Opvragen tabeladres uit videogeheugen
VPEEK - Leest een byte uit het videogeheugen
VPOKE - Schrijft een byte in het videogeheugen
VDP - Leest/schrijft een register van de video display processor

5.1 De organisatie van het videogeheugen

In hoofdstuk 2 zagen we reeds, dat het videogeheugen van de MSX2 computer sterk is uitgebreid tot maximaal 128 kBytes. In dit hoofdstuk zullen we van die maximale grootte uitgaan. Dat betekent dat er afhankelijk van de gebruikte schermmode vier of twee schermpagina's in het videogeheugen kunnen worden opgeslagen.

Alle nieuwe schermmodes zijn zogenaamde *bit-mapped* schermen. Dit wil zeggen, dat indien er tekst naar zo'n scherm wordt geschreven, niet

de ASCII kode wordt opgeslagen, maar de vorm van de letters. Die vorm wordt bepaald door de kleur van de pixels. Ieder afzonderlijk pixel op het scherm kan afzonderlijk worden gekleurd. De informatie over de scherminhoud wordt in het video geheugen opgenomen in de vorm van een aantal tabellen. Die tabellen hebben startadressen, die afhankelijk zijn van de schermmode.

Met de systeemvariabele BASE kunnen de startadressen van de meeste tabellen worden gelezen. Voor de oude MSX1 tabellen (0 tot en met 19) geldt bovendien, dat de startadressen van de tabellen kunnen worden gewijzigd. De adressen van de nieuwe MSX2 tabellen kunnen alleen worden gelezen. Het formaat voor het gebruik van de systeemvariabele BASE voor het lezen van een tabeladres is:

```
PRINT BASE(<base-nr>)
of
LET X=BASE(<base-nr>)
```

Het *<base-nr>* mag variëren van 0 tot en met 44. Merk op dat niet alle base nummers aan een tabel zijn gerelateerd. Aan BASE(41) is bijvoorbeeld geen tabel toegekend. Gebruik van deze systeemvariabele zal de waarde 0 opleveren. Wil men een tabel adres wijzigen, dan is het formaat van de te gebruiken systeemvariabele:

```
BASE(<base-nr>) = <adres>
```

In dit geval mag *<base-nr>* slechts een waarde van 0 tot en met 19 hebben. Een bijzonderheid is, dat wanneer de start adressen van BASE(10) tot en met BASE(14) worden gewijzigd, dat dan automatisch de startadressen van BASE(20) tot en met BASE(24) worden meegewijzigd.

Met het volgende programma worden alle BASE adressen uitgelezen en op het beeldscherm afgedrukt.

```
1000 S=8
1010 SCREEN S
1020 DIM S$(44)
1030 FOR I=S*5 TO S*5+4
1040 S$(I)=RIGHT$("0000"+HEX$(BASE(I)),4)
1050 NEXT I
```

```
1060 SCREEN 0: WIDTH 80
1070 FOR I=S*5 TO S*5+4
1080 PRINT S$(I)
1090 NEXT I
```

Van een aantal tabellen kunnen de startadressen niet met de systeemvariabele BASE worden opgevraagd. Dit zijn bijvoorbeeld de palettetabellen. Deze tabellen hebben echter wel een vaste plaats in het videogeheugen (vast binnen een schermmode). We kunnen ze dus toch lezen. Bij de uitleg van die tabellen komen we daarop terug. We beginnen nu met de uitleg van de tabellen van de eerste schermmode en gaan vervolgens verder met de tabellen uit de daaropvolgende schermmodes.

5.2 Videogeheugentabellen onder SCREEN 0

Onder SCREEN 0 hebben we te maken met achtereenvolgens de scherm-info tabel, de matrix tabel, de palette tabel en onder WIDTH 80 de kleur tabel.

SCREEN 0 (WIDTH 40):	BASE-nr:	Start:	Lengte:
scherminfo tabel	BASE(0)	&H0000	960
-	BASE(1)	&H0800	-
matrix tabel	BASE(2)	&H0800	2048
-	BASE(3)	&H0000	-
-	BASE(4)	&H0000	-
palette tabel	-	&H0400	32

SCREEN 0 (WIDTH 80):	BASE-nr:	Start:	Lengte:
scherminfo tabel	BASE(0)	&H0000	1920
kleur tabel	BASE(1)	&H0800	240
matrix tabel	BASE(2)	&H1000	2048
-	BASE(3)	&H0000	-
-	BASE(4)	&H0000	-
palette tabel	-	&H0F00	32

De scherminfo tabel:

Voor iedere teken positie op het scherm is er een entry van 1 byte in de scherminfo tabel. Bij een regellengte van 40 tekens per regel bevat de tabel 40 keer 24 is 960 bytes. Wanneer de regellengte 80 tekens per regel is, zal de tabel 80 keer 24 is 1920 bytes lang zijn. Iedere entry bevat de ASCII kode van het teken dat op de bijbehorende tekenpositie op het scherm moet worden afgedrukt. Indien er op de bij de entry behorende tekenpositie geen teken is afgedrukt, zal de entry 0 zijn. Indien er een spatie is afgedrukt, zal de entry de ASCII kode voor het spatie-teken bevatten.

Een voorbeeldje:
Onder SCREEN 0, WIDTH 80, staat er een hoofdletter A op de eerste positie van de tweede regel. De ASCII kode voor de letter A is decimaal 65. De eerste positie van de eerste regel is positie 0. De regel bevat 80 tekenposities, van 0 tot en met 79. De eerste positie van de tweede regel is derhalve positie 80. Dit houdt in, dat entry nummer 80 van de scherminfo tabel de waarde 65 (ASCII kode voor de hoofdletter A) bevat.

De kode in de scherminfo tabel wordt gebruikt om de matrix tabel te adresseren.

De matrix tabel:

In de matrix tabel is de vorm van de tekens, die op het scherm kunnen worden afgedrukt, opgenomen. Er kunnen 256 verschillende tekens worden afgedrukt. Alle tekens worden in principe opgebouwd uit een matrix van 8 horizontale bij 8 vertikale beeldpuntjes. De tabel bevat derhalve 256 entries van 8 bytes. Iedere byte bestaat uit 8 bits. Ieder bit is gekoppeld aan een beeldpuntje. Op het scherm worden alleen de linker 6 beeldpuntjes van ieder horizontaal lijntje afgedrukt. Hierdoor kunnen er tot 40 of 80 tekens per regel worden afgedrukt (bij respektievelijk 256 beelpuntjes per lijn en 512 beeldpuntjes per lijn voor WIDTH 40 en 80). Van de 6 horizontale beeldpuntjes kunnen alleen de linker 5 worden gebruikt voor de letterdefinitie. Het zesde beeldpuntje dient als scheiding tussen twee letters.

Het volgende programma drukt een matrix vergroot af op het beeldscherm. Met de FOR/NEXT lus van de regels 20 tot en met 40 wordt

de matrix van de in regel 30 opgegeven letter (in het voorbeeld de letter y) uit de tabel gelezen en in de gedimensioneerde variabele A geplaatst. Deze variabele wordt in de FOR/NEXT lus van regels 50 tot en met 70 als binaire waarde op het scherm afgedrukt. Het resultaat zal zijn, dat de matrix van 8 bij 8 wordt afgedrukt met een 0 indien er niets moet worden afgedrukt en een 1 indien er een pixel moet worden geschreven.

```
10 SCREEN 0: WIDTH 80
20 FOR I=0 TO 7
30 A(I)=VPEEK(&H1000+(ASC("y")*8+I))
40 NEXT I
50 FOR I=0 TO 7
60 PRINT RIGHT$("00000000"+BIN$(A(I)),8)
70 NEXT I
```

Het resultaat van dit programma zal er als volgt uit zien:

```
00000000
00000000
10001000
10001000
10011000
01101000
00001000
01110000
```

Door nu de matrix tabel te wijzigen, kunnen we de vorm van een af te drukken teken wijzigen. Dit wordt in het volgende programma gedemonstreerd.

```
10 SCREEN 0: WIDTH 80
20 FOR I=0 TO 7
30 READ A(I):VPOKE &H1000+ASC("A")*8+I,A(I)
40 NEXT I
50 FOR I=0 TO 7
60 READ A(I):VPOKE &H1000+ASC("T")*8+I,A(I)
70 NEXT I
80 PRINT "TAAK"
90 END
100 DATA 28,36,36,68,124,132,132,0
110 DATA 124,16,32,32,64,64,192,0
```

In de DATA regels zijn de definities van twee nieuwe tekens opgenomen. Met de FOR/NEXT lus van regels 20 tot en met 40 wordt de definitie uit DATA regel 100 toegekend aan de hoofdletter A. Met de daaropvolgende FOR/NEXT lus wordt de definitie van DATA regel 110 toegekend aan de hoofdletter T. Vervolgens wordt het woordje TAAK afgedrukt. U zult zien, dat de letters A en T nu kursief worden afgedrukt.

Na uitvoering van dit programma is de matrix tabel gewijzigd. Alles wat u nu afdrukt op het scherm staat bloot aan die gewijzigde matrix tabel. Wanneer u nu bijvoorbeeld het programma 'list', dan zult u zien dat alle letters A en T in de listing kursief worden afgedrukt. De manier om weer de orginele letters A en T terug te krijgen is, een SCREEN kommando te geven. Door uitvoering van het SCREEN kommando wordt de matrix tabel opnieuw vanuit de ROM in het video RAM geladen.

In voorgaande voorbeelden werd gebruik gemaakt van de funktie VPEEK en het statement VPOKE. Met de funktie VPEEK kan een byte uit het videogeheugen worden gelezen. Het formaat van deze funktie is:

```
A = VPEEK(<adres>)
```

Het *<adres>* dat mag worden opgegeven hangt af van de op dat moment geldende schermmode. Bij SCREEN 0 tot en met 4 mag het adres variëren van 0 tot en met 16383. Bij SCREEN 5 tot en met 8 mag het adres variëren van 0 tot en met 65535. Nu kan het video geheugen 128 kBytes groot zijn. We kunnen echter slechts tot maximaal 64 kBytes adresseren. Zoals u zich echter zult herinneren is het videogeheugen onder schermmodes 5 tot en met 8 opgedeeld in vier (SCREEN 5 en 6) of twee (SCREEN 7 en 8) pagina's. Het werkelijke adres wordt dan ook door de computer berekend met behulp van de formules:

SCREEN 5 en 6: adres = paginanr * &H8000 + *<adres>*
SCREEN 7 en 8: adres = paginanr * &H10000 + *<adres>*

Het pagina nummer wordt, zoals u zich zult herinneren, ingesteld met het statement SET PAGE. VPEEK werkt altijd op de aktieve pagina. Dat hoeft dus niet altijd de op het beeldscherm afgedrukte pagina te zijn.

Het statement VPOKE heeft het volgende formaat:

VPOKE <*adres*>,<*inhoud*>

Voor het <*adres*> gelden dezelfde regels als voor het adres in de VPEEK funktie. <*inhoud*> mag een waarde van 0 tot en met 255 hebben.

De palette tabel:

De palette tabel komt in alle schermmodes voor. Deze tabel kan echter niet met behulp van BASE( ) worden opgevraagd. In de schermmode tabel aan het begin van deze paragraaf wordt echter het adres van de palette tabel binnen een schermpagina gegeven. Deze adressen zijn vast en worden door alle MSX2 computers gebruikt.

De palette tabel bestaat uit 16 entries. Iedere entry is twee bytes lang. Het entry nummer komt overeen met het kleurnummer. Entry nummer 1 wordt door de computer gebruikt om te bepalen wat de kleur van kleurnummer 1 moet zijn. In de entry zijn de hoeveelheden rood, groen en blauw gekodeerd. Deze kodering werkt als volgt:

	1e byte		2e byte	
Entry *n*	rood	blauw		groen

Rood, blauw en groen kunnen een waarde van 0 tot en met 7 hebben, waarmee de hoeveelheid licht (de intensiteit) van iedere kleur wordt aangegeven (zie het COLOR statement uit hoofdstuk 3). Gewapend met de kennis over het beginadres van de palette tabel en de indeling van die tabel zijn we in staat om de palette tabel met behulp van VPEEK en VPOKE te benaderen, zoals in het volgende programma wordt gedaan.

```
1000 DIM BA(16): DIM BB(16)
1010 SCREEN 7
1020 FOR I=0 TO 31 STEP 2
1030 BA(I/2)=VPEEK(&HFA80+I)
1040 BB(I/2)=VPEEK(&HFA81+I)
1050 NEXT I
```

```
1060 SCREEN 0: WIDTH 80
1070 PRINT "nr:  red  blue grn
1080 FOR I=0 TO 15
1090 PRINT USING "##";I;:PRINT " - ";
1100 BA$=RIGHT$("00000000"+BIN$(BA(I)),8)
1110 PRINT LEFT$(BA$,4);
1120 PRINT " ";
1130 PRINT RIGHT$(BA$,4);
1140 PRINT " ";
1150 PRINT RIGHT$("00000000"+BIN$(BB(I)),4)
1160 NEXT I
```

De kleurtabel:

Deze tabel, die alleen bij WIDTH 41 of hoger voorkomt, bestaat uit bitjes, die 0 of 1 kunnen zijn. Ieder bitje is gerelateerd aan een tekenpositie op het scherm. Het eerste bitje hoort bij de eerste schermpositie (helemaal linksbovenaan). Het tweede bitje behoort bij de schermpositie rechts van de eerste schermpositie. Elk volgend bit heeft betrekking op elke volgende schermpositie.

Normaal hebben alle bitjes de waarde 0. Het is echter mogelijk om één of meer bitjes 1 te maken. Dit kunnen we met het VPOKE statement doen. Stel dat we de bitjes 4 tot en met 11 de waarde 1 willen geven. We geven dan het volgende statement:

```
VPOKE &H0800,&B00001111
VPOKE &H0801,&B11110000
```

We adresseren het videogeheugen met het startadres van de kleurtabel en zetten daar de binaire waarde 00001111 neer. Hierdoor zijn de bitjes 0 tot en met 3 nog steeds nul, maar krijgen de bitjes 4 tot en met 7 de waarde 1. Daarna adresseren we het volgende adres uit de kleurtabel (&H0801) en zetten we daarop de binaire waarde 11110000 neer. Er gebeurt nu nog steeds niets. We hebben alleen de bitjes uit de kleurtabel, die betrekking hebben op de schermposities 4 tot en met 11, op 1 gezet.

De video display processor heeft twee registers, waarmee de op 1 gezette bitjes uit de kleurtabel kunnen worden beïnvloed. Dit wil zeggen, dat de tekens op de daarbij behorende schermposities worden beïnvloed. Die beïnvloeding is tweeerlei. Ten eerste wordt de kleur

beïnvloed en ten tweede wordt de tijd dat de kleur wordt beïnvloed ingesteld. De twee registers van de video display processor die kunnen worden gewijzigd zijn VDP(13) en VDP(14). In VDP(13) kunnen we de voor en achtergrondkleur instellen van tekens op posities waarvoor de kleurtabel een 1 aangeeft. Voor diezelfde tekens kunnen we met VDP(14) aangeven hoe lang die tekens de gewijzigde kleurinstelling moeten krijgen. Het formaat van VDP(13) is als volgt:

```
VDP(13) = 16 * <voorgrondkleur> + <achtergrondkleur>
```

Het formaat van VDP(14) is als volgt:

```
VDP(14) = 16 * <periode 1> + <periode 2>
```

<periode 1> is een waarde, die aangeeft hoeveel keer 0,2 sekonden de gewijzigde kleur aktief moet zijn. *<periode 2>* geeft aan hoeveel keer 0,2 sekonden de normale kleur aktief moet zijn. Wilt u beide kleuren steeds 1 sekonde aktief hebben, dan zou u het volgende statement moeten programmeren:

```
VDP(14) = 16 * 5 + 5
```

Het volgende programma laat zien hoe men een stukje tekst op het scherm kan laten knipperen. Als u dit programma goed bestudeert en de voorgaande uitleg daarbij gebruikt, zult u ontdekken dat u iedere gewenste positie van het scherm kunt laten knipperen. Bovendien hoeft het knipperen niet tot een enkele plaats te worden beperkt. Men kan net zoveel posities van het scherm laten knipperen als men wil. Heeft u eenmaal één of meer plaatsen op het scherm op knipperen ingesteld, dan is de enige manier om dat knipperen weer op te heffen het vullen van de kleurtabel met allemaal nullen.

```
10 SCREEN 0: WIDTH 80: COLOR 1,15
20 VDP(13)=16*8+15
30 VDP(14)=16*2+2
40 PRINT "Het woord KNIPPER knippert"
50 VPOKE BASE(1)+1,&B00111111
60 VPOKE BASE(1)+2,&B10000000
70 END
```

5.3 Videogeheugentabellen onder SCREEN 1

Onder SCREEN 1 hebben we te maken met dezelfde tabellen als onder SCREEN 0, al zijn er wel enkele verschillen. Bovendien hebben we onder SCREEN 1 te maken met twee nieuwe tabellen, de sprite info tabel en de sprite$ tabel.

SCREEN 1	BASE-nr:	Start:	Lengte:
scherminfo tabel	BASE(5)	&H1800	768
kleur tabel	BASE(6)	&H2000	32
matrix tabel	BASE(7)	&H0000	2048
sprite-info tabel	BASE(8)	&H1B00	128
sprite$ tabel	BASE(9)	&H3800	2048
palette tabel	-	&H2020	32

De scherminfo tabel:

Net als onder SCREEN 0, echter met dien verstande dat de lengte van de tabel anders is. Onder SCREEN 1 is de regellengte 32 tekens. Bij 24 regels kunnen er dus in totaal 24 keer 32 is 768 tekens op het scherm worden geschreven. De kodes voor die 768 tekens staan in de scherm-info tabel, die dus ook 768 bytes lang moet zijn.

De kleurtabel:

De kleurtabel is onder SCREEN 1 slechts 32 bytes lang. Ieder byte heeft betrekking op een serie van 8 tekenposities op het scherm. Ieder byte in de kleurtabel bevat informatie over de voorgrond- en achtergrondkleur. Deze kleuren hebben betrekking op een groepje van 8 tekens (schermposities). De kleur van de eerste 8 tekens op de eerste regel van het scherm wordt bepaald door de inhoud van het eerste byte uit de kleurtabel. Het volgende groepje van 8 tekens op het scherm wordt gekleurd volgens het tweede byte uit de kleurtabel, enz.

Willen we de kleur van het tweede groepje van 8 tekens op de eerste regel van het beeldscherm veranderen, dan dienen we de gewenste kleuren met behulp van VPOKE in de kleurtabel te schrijven. Het formaat van het VPOKE statement waarmee we dit kunnen doen is:

VPOKE BASE(6)+<*entry nummer*>,16*<*voorgrond*>+<*achtergrond*>

Willen we de letters rood kleuren tegen een groene achtergrond, dan zullen we het statement VPOKE BASE(6)+1,16*8+3 moeten programmeren, zoals in het volgende programma is gedaan.

```
10 SCREEN 1: WIDTH 32
20 FOR I=0 TO 255
30 IF I<32 THEN PRINT CHR$(1)+CHR$(64+I); ELSE PRINT CH
R$(I);
40 NEXT I
50 VPOKE BASE(6)+1,16*8+3
60 END
```

Met BASE(6) wordt het startadres van de kleurtabel verkregen. Daarbij tellen we 1 op, om de tweede entry uit de kleurtabel te adresseren. De voorgrond- en achtergrondkleur worden naar de geadresseerde positie binnen de kleurtabel geschreven met regel 50 van het programma.

De matrix tabel:

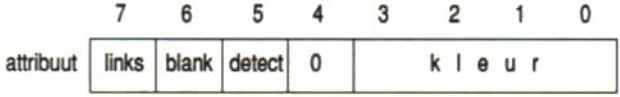
Deze tabel is gelijk aan de overeenkomstige tabel uit SCREEN 0. Zie aldaar.

De sprite info tabel:

Deze tabel bestaat uit 32 entries van elk 4 bytes. Voor iedere transparant, waarop een sprite kan worden afgebeeld, is er dus een entry in de tabel. De entries hebben allemaal de volgende indeling:

	byte 1	byte 2	byte 3	byte 4
Entry *n*	Y	X	spritenummer	attributen

Y en X zijn de koördinaten, waarop de betreffende sprite op het scherm wordt afgedrukt. In de attributen wordt onder meer de kleur van de sprite aangegeven. Het attributenbyte (byte 4) ziet er als volgt uit:



De kleur wordt gekodeerd in vier bitjes en kan dus variëren van 0 tot en met 15.

Indien bit 7 de waarde 1 heeft, wordt het sprite 32 beeldpunten meer naar links op het beeldscherm afgedrukt. Delen van de sprite die voorbij de linker beeldschermrand komen, verdwijnen, zonder dat ze aan de rechter kant van het scherm weer verschijnen.

Indien bit 6 de waarde 1 heeft, wordt de sprite niet zichtbaar afgebeeld. Passeert deze sprite een voorliggende sprite, dan worden de overlappende delen wel zichtbaar gemaakt. Passeert deze sprite een achterliggende sprite, dan blijft deze onzichtbaar en wordt er geen botsing gedetekteerd.

Indien bit 5 de waarde 1 heeft, worden botsingen van deze met een andere sprite niet gedetekteerd.

De sprite$ tabel:

Deze tabel is 2048 bytes lang en bevat de definites van de sprite-vorm. Kleine sprites bestaan uit 8 bytes. Van deze kleine sprites passen er dus 2048 gedeeld door 8 is 256 in de sprite$ tabel. Grote sprites bestaan uit 32 bytes. Derhalve passen er van deze sprites slechts 2048 gedeeld door 32 is 64 in de sprite$ tabel.

5.4 Videogeheugentabellen onder SCREEN 2

Al kent SCREEN 2 dezelfde tabelnamen als SCREEN 1, er zijn toch een aantal verschillen in de inhoud van enkele tabellen. De scherminfo tabel, de matrix tabel en de kleurtabel hebben onder SCREEN 2 een geheel andere indeling, die hierna zal worden toegelicht.

SCREEN 2	BASE-nr:	Start:	Lengte:
scherminfo tabel	BASE(10)	&H1800	768
kleur tabel	BASE(11)	&H2000	6144
matrix tabel	BASE(12)	&H0000	6144
sprite-info tabel	BASE(13)	&H1B00	128
sprite$ tabel	BASE(14)	&H3800	2048
palette tabel	-	&H1B80	32

De scherminfo tabel:

De scherminfo tabel bestaat nu uit drie blokken van 256 bytes. De drie blokken worden door BASIC gevuld met entries die de waarden 0 tot en met 255 hebben. De entries uit het eerste blok zijn gerelateerd aan de eerste 256 entries uit de matrix tabel. De volgende 256 entries uit de scherminfo tabel zijn gerelateerd aan de entries 256 tot en met 511 uit de matrix tabel. De entries uit het derde blok zijn gerelateerd aan de entries 512 tot en met 767 uit de matrix tabel.

Zou men de scherminfo tabel zo wijzigen dat de eerste entry geen nul, maar bijvoorbeeld 255 bevat, dan wil dit zeggen, dat de 256e entry uit de matrix tabel wordt afgedrukt op de eerste positie van het bovenste blok van het beeldscherm (helemaal links boven dus).

De matrix tabel:

Onder SCREEN 2 is het beeldscherm opgebouwd uit 24 regels van 32 bytes elk. Iedere beeldschermpositie bestaat uit een matrix van acht bij acht beeldpuntjes. Een dergelijke matrix kan in acht bytes van acht bits worden gekodeerd. De matrix tabel bestaat uit entries van acht bytes. De eerste 256 entries van acht bytes zijn gerelateerd aan het eerste blok uit de scherminfo tabel. 256 entries zijn nodig om 8 regels van 32 tekenposities te kunnen koderen.

Wanneer een vertikale lijn op het beeldscherm is getrokken, die langs de beeldpuntjes gaat, die op 6 beeldpuntjes van de linkerkant van het scherm gaan, dan zal de eerste entry in de matrix tabel de waarden 4, 4, 4, 4, 4, 4 en 4 hebben. Dit geldt ook voor de 32e entry, de 64e entry, enzovoort. Al deze entries zien er als volgt uit:

```
00000100
00000100
00000100
00000100
00000100
00000100
00000100
00000100
```

Zouden we nu in de scherminfo tabel de tweede entry (entry nummer 1) de waarde 0 geven, dan zal naast de vertikale lijn op het zesde beeldpuntje ook een vertikaal lijntje op het 13e beeldpunt verschijnen. Voorgaande theorie wordt in het volgende programma in de praktijk gebracht.

```
1000 SCREEN 2
1010 OPEN "grp:" AS #1
1020 LINE (5,0)-(5,100)
1030 '***** LEES ENTRY UIT MATRIX TABEL *****
1040 PRESET(0,100)
1050 FOR I=0 TO 7
1060 PRINT #1,RIGHT$("00000000"+BIN$(VPEEK(BASE(12)+I)),8)
1070 NEXT I
1080 '***** WIJZIG DE SCHERM INFO TABEL *****
1090 FOR I=0 TO 7
1100 VPOKE BASE(10)+I*32+1,0
1110 NEXT I
1120 '***** RESULTAAT TE ZIEN OP SCHERM *****
1130 GOTO 1130
```

De kleurtabel:

De kleurtabel loopt parallel aan de matrix tabel. Een byte in de matrix tabel bevat de informatie voor 8 bitjes, die horizontaal naast elkaar liggen. Wanneer een bitje 1 is, dan wil dat zeggen, dat het bijbehorende beeldpuntje met de inktkleur moet worden geschreven. Is een bitje 0, dan wordt het bijbehorende beeldpuntje in achtergrondkleur geschreven. In de kleurtabel staat voor iedere acht bitjes uit de matrix tabel wat de voorgrond en de achtergrondkleuren zijn. Per acht horizontale beeldpuntjes kunnen dus slechts twee verschillende kleuren worden gedefinieerd.

Een entry uit de kleurtabel omvat 1 byte. In de acht bitjes van dat byte worden de voorgrondkleur en de achtergrondkleur gekodeerd. De kodering is als volgt:

16 keer de voorgrondkleur + de achtergrondkleur

Bij een voorgrondkleur 1 (zwart) en een achtergrondkleur 15 (wit) zal de entry in de kleur tabel dus 16 * 1 + 15 = 31 zijn.

De sprite info tabel en de sprite$ tabel zijn zoals onder SCREEN 1 werd omschreven. De palette tabel werd al omschreven onder SCREEN 0 (40 tekens).

5.5 Videogeheugentabellen onder SCREEN 3

Ook nu weer zijn er veel overeenkomsten tussen de tabellen van SCREEN 3 en de andere schermmodes. De kleurtabel bestaat onder SCREEN 3 niet. De sprite info tabel en de sprite$ tabel werden reeds beschreven onder SCREEN 1. De palette tabel werd al beschreven onder SCREEN 0.

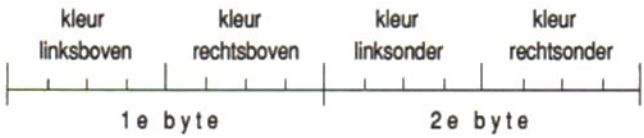
SCREEN 3	BASE-nr:	Start:	Lengte:
scherminfo tabel	BASE(15)	&H0800	768
-	BASE(16)	&H0000	-
matrix tabel	BASE(17)	&H0000	2048
sprite-info tabel	BASE(18)	&H1B00	128
sprite$ tabel	BASE(19)	&H3800	2048
palette tabel	-	&H2020	32

De matrix tabel:

Onder SCREEN 3 is het beeldscherm opgedeeld in blokjes van vier bij vier beeldpuntjes. Deze blokjes zijn gegroepeerd in tekenposities, waarbij steeds twee horizontale blokjes met de daaronderliggende blokjes een tekenpositie van acht bij acht beeldpuntjes vormen. Een tekenpositie ziet er dan als volgt uit:

blokje van 4 x 4 pixels	blokje van 4 x 4 pixels
blokje van 4 x 4 pixels	blokje van 4 x 4 pixels

De kodering van zo'n tekenpositie in een twee bytes entry in de matrix tabel gaat nu als volgt:



Voor iedere tekenpositie is een entry van twee bytes aanwezig in de matrix tabel. Elk blokje van vier bij vier pixels kan een eigen kleur worden toegekend. In de entry in de matrix tabel is voor iedere kleur een groepje van 4 bits beschikbaar, waarmee 16 verschillende kleuren kunnen worden gekodeerd.

Onder SCREEN 3 bestaat het scherm uit 24 regels van 32 tekens. Er zijn dus in totaal 24 * 32 = 768 tekenposities. Daar er voor iedere tekenpositie een twee bytes entry in de matrix tabel staat, is de lengte van de matrix tabel 768 * 2 = 1536 bytes.

De scherminfo tabel:

Deze tabel bestaat uit 768 entries van 1 byte elk, die onder BASIC worden gevuld met oplopende waarden, ongeveer zoals dat ook onder SCREEN 2 werd gedaan. Alleen krijgen nu steeds vier regels dezelfde waarden, waarna de volgende vier regels een hogere waarde krijgen. De volgende opsomming laat dit zien:

```
regel 1:   0  1  2  3   ...  29 30 31
regel 2:   0  1  2  3   ...  29 30 31
regel 3:   0  1  2  3   ...  29 30 31
regel 4:   0  1  2  3   ...  29 30 31
regel 5:  32 33 34 35   ...  61 62 63
regel 6:  32 33 34 35   ...  61 62 63
enz.
```

Iedere waarde in de scherminfo tabel verwijst naar een entry in de matrix tabel. Dit werkt net zoals onder SCREEN 2.

5.6 Videogeheugentabellen onder SCREEN 4

Alle videogeheugentabellen onder SCREEN 4 zijn exakt gelijk aan dezelfde tabellen onder SCREEN 2, behalve de sprite info tabel en de sprite kleurtabel. SCREEN 4 is namelijk volledig kompatibel met SCREEN 2. Alleen de sprite mogelijkheden zijn uitgebreid. Die nieuwe uitgebreide sprite mogelijkheden gelden overigens voor alle nieuwe MSX2 scherminstellingen (SCREEN 4 tot en met 8).

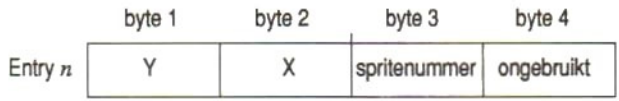
SCREEN 4	BASE-nr:	Start:	Lengte:
scherminfo tabel	BASE(20)	&H1800	768
kleur tabel	BASE(21)	&H2000	6144
matrix tabel	BASE(22)	&H0000	6144
sprite-info tabel	BASE(23)	&H1E00	128
sprite$ tabel	BASE(24)	&H3800	2048
palette tabel	-	&H1B80	32
sprite kleur tabel	-	&H1C00	512

Zie voor de scherminfo tabel, de kleurtabel, de matrix tabel de sprite$ tabel en de palette tabel de beschrijving bij SCREEN 2 (paragraaf 5.4).

De sprite info tabel:

In deze tabel komt voor ieder transparant een entry van 4 bytes voor. Er zijn 32 transparanten. De lengte van deze tabel is derhalve 4 * 32 = 128 bytes. De indeling van iedere entry is bijna gelijk aan de indeling zoals die onder SCREEN 1 (paragraaf 5.3) werd beschreven. Alleen het

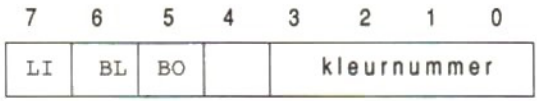
vierde byte van de entry wordt onder SCREEN 4 niet gebruikt. Onder SCREEN 1 was dat het attributendeel, waarmee de kleur, de zichtbaarheid en het al of niet detekteren van botsingen kon worden gekodeerd. Deze mogelijkheden worden nu overgenomen door de sprite kleurtabel (zie hierna). Het formaat van een entry ziet er als volgt uit:



Wat dan nog overblijft zijn de eerste drie bytes. Het eerste byte bevat de *y*-koördinaat, het tweede de *x*-koördinaat en het derde byte bevat het sprite nummer. Het vierde byte wordt gewoon niet gebruikt.

De sprite kleurtabel:

Deze tabel bestaat uit 32 entries van 16 bytes. Iedere entry bevat de kleur informatie voor een sprite. Er zijn 32 transparanten waarop een sprite kan worden afgebeeld. Er zijn daarom ook 32 entries in deze tabel nodig. Ieder byte uit de 16 bytes entry bevat onder meer de kleur informatie voor een horizontaal lijntje van de betreffende sprite. Omdat sprites maximaal 16 by 16 beeldpuntjes groot kunnen zijn, zijn er maximaal 16 horizontale lijntjes nodig. Wanneer kleine sprites worden gebruikt, worden alleen de eerste 8 bytes van iedere entry gebruikt. De indeling van een byte uit een entry ziet er als volgt uit:



De kleur van een sprite-lijntje wordt gekodeerd in vier bitjes en kan dus variëren van 0 tot en met 15.

Als bit 7 de waarde 1 heeft, dan wordt het betreffende sprite-lijntje 32 beeldpunten meer naar links op het beeldscherm afgedrukt. Delen van de sprite die voorbij de linker beeldschermrand komen, verdwijnen, zonder dat ze aan de rechterkant van het scherm weer verschijnen.

Heeft bit 6 de waarde 1, dan wordt dit sprite-lijntje niet zichtbaar afgebeeld. Passeert dit sprite-lijntje een voorliggende sprite, dan worden de overlappende delen wel zichtbaar gemaakt. Passeert dit sprite-lijntje een achterliggende sprite, dan blijft deze onzichtbaar en wordt er geen botsing gedetekteerd.

Indien bit 5 de waarde 1 heeft, dan worden botsingen van dit sprite-lijntje met een andere sprite niet gedetekteerd.

5.7 Videogeheugentabellen onder SCREEN 5

Alle nieuwe beeldschermen, vanaf SCREEN 5, zijn zogenaamde bit-mapped schermen. Dat wil zeggen, dat deze schermen niet langer teken-georiënteerd zijn, maar dat ze gebaseerd zijn op individuele beeldpuntjes. Voor al deze nieuwe schermmodes geldt, dat de sprite mogelijkheden gelijk zijn aan de sprite mogelijkheden van SCREEN 4, zoals in de vorige paragraaf beschreven. We zullen in deze en de volgende paragrafen dan ook niet terugkomen op de sprite tabellen.

SCREEN 5	BASE-nr:	Start:	Lengte:
scherminfo tabel	BASE(25)	&H0000	27136
-	BASE(26)	&H0000	-
-	BASE(27)	&H0000	-
sprite-info tabel	BASE(28)	&H7600	128
sprite$ tabel	BASE(29)	&H7800	2048
palette tabel	-	&H7680	32
sprite kleur tabel	-	&H7400	512

Voor SCREEN 5 geldt, dat er maximaal vier beeldschermpagina's in het videogeheugen kunnen worden opgeslagen. De adressen in voorgaande tabel zijn geen absolute adressen, maar zijn adressen binnen een beeldschermpagina. Iedere beeldschermpagina is 32 kBytes lang. Vier pagina's hebben dus een 128 kBytes video geheugen nodig. De startadressen van de pagina's zijn:

pagina 0	&H00000
pagina 1	&H08000
pagina 2	&H10000
pagina 3	&H18000

Door de startadressen van de tabellen op te tellen bij de startadressen van iedere pagina wordt het absolute adres van een tabel binnen het videogeheugen gevonden.

De enige nieuwe tabel is de scherminfo tabel. Om de indeling van die tabel te kunnen begrijpen moeten we ons eerst even het volgende realiseren.

Het beeldscherm bestaat in deze schermmode uit een matrix van 256 pixels horizontaal bij 212 pixels vertikaal. Ieder afzonderlijk pixel kan een eigen kleur hebben. Het maximale aantal verschillende kleuren dat tegelijkertijd op het scherm kan worden weergegeven is 16. De kleuren worden gekozen uit een palette waarop 16 kleuren worden geselekteerd uit 512 verschillende kleuren.

Om 16 verschillende kleuren te kunnen aangeven, zijn vier bitjes nodig. De scherminfo tabel bevat de kleur informatie voor ieder afzonderlijk pixel. Per pixel zijn dus vier bitjes nodig. In een byte kunnen de kleuren van twee pixels worden gekodeerd. Er zijn 256 * 212 = 54272 pixels. Per twee pixels is een byte nodig. De lengte van de scherminfo tabel zal daarom 54272 / 2 = 27136 bytes zijn.

In de scherminfo tabel worden de kleuren van de pixels gekodeerd in de volgorde waarin de pixels op het scherm verschijnen. Het eerste pixel ligt in de linker bovenhoek van het scherm, het tweede pixel en de volgende rechts daarvan. Het 257e pixel bevindt zich recht onder het eerste pixel, op de tweede beeldlijn. De eerste twee pixels van de eerste beeldlijn zijn gekodeerd in het eerste byte. Het derde en vierde pixel zijn gekodeerd in het tweede byte, enzovoort.

5.8 Videogeheugentabellen onder SCREEN 6

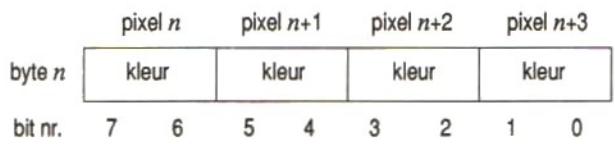
Ook nu is er weer sprake van een zogenaamd "bit-mapped" scherm. En net als onder SCREEN 5 kunnen er maximaal vier beeldschermpagina's tegelijk in het videogeheugen worden opgeslagen. De beginadressen van iedere beeldschermpagina is net zo als onder SCREEN 5.

SCREEN 6	BASE-nr:	Start:	Lengte:
scherminfo tabel	BASE(30)	&H0000	27136
-	BASE(31)	&H0000	-
-	BASE(32)	&H0000	-
sprite-info tabel	BASE(33)	&H7600	128
sprite$ tabel	BASE(34)	&H7800	2048
palette tabel	-	&H7680	32
sprite kleur tabel	-	&H7400	512

De enige nieuwe tabel is de scherminfo tabel. Onder SCREEN 6 zijn slechts vier verschillende kleuren per pixel mogelijk. Om vier verschillende kleuren te koderen zijn slechts twee bitjes nodig. Zodoende kunnen er in een byte kleuren voor vier pixels worden gekodeerd. Hierdoor zou de scherminfo tabel de helft korter kunnen worden ten opzichte van de scherminfo tabel uit SCREEN 5. Echter, daar er onder SCREEN 6 op iedere beeldlijn twee keer zoveel pixels staan (512), wordt de totale lengte van de tabel toch even lang als onder SCREEN 5.

Gaande van links naar rechts, beginnende bij het pixel in de linker bovenhoek, worden de kleurkodes van de bits in de scherm info tabel gekodeerd. De pixels 0 tot en met 3 worden gekodeerd in het eerste byte uit de scherminfo tabel. De pixels 4 tot en met 7 in het tweede byte uit de scherminfo tabel, enzovoort.

Een byte uit de scherminfo tabel ziet er als volgt uit:



Voor iedere kleur zijn twee bitjes beschikbaar. Hieruit volgt, dat alleen de kleurnummers 0 tot en met 3 kunnen worden gebruikt. Door het wijzigen van de palette kan voor deze kleurkodes echter een keuze worden gemaakt uit 512 verschillende kleuren. Er kunnen echter nooit meer dan vier verschillende kleuren tegelijk op het beeldscherm worden geplaatst.

5.9 Videogeheugentabellen onder SCREEN 7

De volgende twee beeldschermmodes kunnen alleen op een MSX2 computer met 128 kBytes videogeheugen worden gebruikt. De scherm info tabellen van deze schermmodes zijn namelijk zo groot, dat er in een 128 kBytes videogeheugen slechts twee pagina's kunnen worden opgeslagen. Dit geldt zowel voor SCREEN 7 als voor SCREEN 8. De startadressen van de pagina's zijn:

pagina 0	&H00000
pagina 1	&H10000

De startadressen van de tabellen uit de volgende opsomming zijn relatief ten opzichte van het startadres van de pagina.

SCREEN 7	BASE-nr:	Start:	Lengte:
scherminfo tabel	BASE(35)	&H0000	54272
-	BASE(36)	&H0000	-
-	BASE(37)	&H0000	-
sprite-info tabel	BASE(38)	&HFA00	128
sprite$ tabel	BASE(39)	&HF000	2048
palette tabel	-	&HFA80	32
sprite kleur tabel	-	&HF800	512

Onder SCREEN 7 wordt het beeldscherm opgebouwd uit 512 pixels horizontaal bij 212 pixels vertikaal. Voor ieder pixel kan uit 16 verschillende kleuren worden gekozen. De scherminfo tabel is net zo ingedeeld als onder SCREEN 5. Daar er twee keer zoveel pixels zijn, wordt de scherminfo tabel ook twee keer zo lang als onder SCREEN 5. Verdere verschillen zijn er echter niet.

5.10 Videogeheugentabellen onder SCREEN 8

Ook deze schermmode kan alleen worden gebruikt op MSX2 computers met 128 kBytes videogeheugen. Dat videogheugen wordt, net als onder SCREEN 7 opgedeeld in twee pagina's met de volgende startadressen:

pagina 0	&H00000
pagina 1	&H10000

De startadressen van de videogheugentabellen, zoals die zijn gegeven in de volgende tabel, zijn relatief ten opzichte van de beginadressen van de video pagina's.

SCREEN 8	BASE-nr:	Start:	Lengte:
scherminfo tabel	BASE(40)	&H0000	54272
-	BASE(41)	&H0000	-
-	BASE(42)	&H0000	-
sprite-info tabel	BASE(43)	&HFA00	128
sprite$ tabel	BASE(44)	&HF000	2048
palette tabel	-	&HFA80	32
sprite kleur tabel	-	&HF800	512

Onder SCREEN 8 is er sprake van een beeldscherm dat wordt gevormd door 256 horizontale pixels bij 212 vertikale pixels – dezelfde resolutie dus als onder SCREEN 5. Het bijzondere van het scherm onder SCREEN 8 is echter, dat voor ieder pixel een keuze uit 256 verschillende kleuren kan worden gemaakt. Om 256 verschillende kleuren te koderen is een heel byte nodig. Met andere woorden, voor ieder pixel op het scherm bevindt zich een byte in de scherm info tabel.

Voor het koderen van de kleur wordt nu niet meer gebruik gemaakt van de kleuren palette. Daarop zaten immers maar 16 verschillende kleuren. De kleur wordt nu rechtstreeks gekodeerd, door de intensiteit van de kleuren rood, groen en blauw aan te geven. Derhalve zal een byte uit de scherminfo tabel er als volgt uitzien:

byte *n*	groen			rood			blauw	
bit nr.	7	6	5	4	3	2	1	0

In formulevorm kan een byte uit de scherminfo tabel als volgt worden gedefinieerd:

32 * groen + 4 * rood + blauw

Groen en rood mogen maximaal de waarde 7 hebben, blauw mag maximaal 3 zijn.

5.11 De Video Display Processor

In de MSX2 computer zit naast de centrale verwerkingseenheid (CPU of *Central Processing Unit*) een tweede processor, die speciaal is ingericht voor het genereren van een videobeeld. Deze wordt *Video Display Processor* (VDP) genoemd. De VDP neemt alle video taken over van de centrale processor. De centrale processor hoeft de videoprocessor alleen maar te vertellen wat deze moet gaan doen. Daarna kan de videoprocessor zelfstandig aan het werk.

Om de centrale processor te kunnen laten kommuniceren met de videoprocessor, is deze laatste uitgerust met een groot aantal registers. Deze registers kunnen door de centrale processor worden uitgelezen en/of gevuld met gegevens. In paragraaf 5.2 zagen we reeds hoe we de centrale processor opdracht kunnen geven een register van de VDP te lezen of te beschrijven. We doen dat met behulp van de systeemvariabele VDP, en wel als volgt:

Lezen	`PRINT VDP(<registernummer>)`
Schrijven	`VDP(<registernummer>) = <waarde>`

Slechts enkele registers zijn voor ons, als BASIC programmeurs, interessant. We zullen daarom niet alle registers uitvoerig behandelen. We noemen alleen de registers en geven waar gewenst een voorbeeldje van het gebruik van een register. De registers zijn opgedeeld in een aantal funktionele groepen. Deze groepen zijn:

VDP mode registers	VDP(0, 1, 9 en 10)
VDP adres registers	VDP(2 t/m 6, 11, 12 en 15)
Tekst kontrole registers	VDP(7, 13 en 14)
VDP kontrole registers	VDP(16 t/m 23)
VDP status registers	VDP(8 en –1 t/m –9)
VDP programma registers	VDP(33 t/m 47)

Alle registers zijn 1 byte (is 8 bits) groot. Elk bit of groepje bits heeft een bepaalde funktie. Door de waarde van de bitjes te wijzigen kunnen

we een funktie wijzigen. De registers zien er als volgt uit:

bit nr.	7	6	5	4	3	2	1	0
register *n*	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
waarde:	128	64	32	16	8	4	2	1

Wanneer een bitje 1 is, vertegenwoordigt het de waarde die er onder staat. Wanneer u een register uitleest en het resultaat is bijvoorbeeld 48, dan wil dat zeggen dat de bitjes 4 en 5 de waarde 1 hebben en alle andere bitjes de waarde nul.

Mode registers:

Bit 6 van VDP(1) kan de weergave van het beeldscherm aan of uit zetten. Het statement, waarmee we dit kunnen doen is:

```
VDP(1)=VDP(1) XOR 64
```

Bit 1 van VDP(1) kan spriteweergave vergroten of verkleinen. Dit doen we met het volgende statement:

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

Bit 5 van VDP(9) verandert kleurnummer 0 van transparant in normale kleur of omgekeerd. Wanneer dit bit 0 is, dan is kleurnummer 0 transparant, is dit bit 1, dan is kleurnummer 0 niet transparant. In dat laatste geval kunnen dus echt 16 verschillende kleuren worden gebruikt. Het niet-transparant maken van kleurnummer 0 gaat als volgt:

```
IF (VDP(9) AND 32)=0 THEN VDP(9)=VDP(9) XOR 32
```

De VDP adres registers:

In deze registers staan de startadressen van de verschillende video tabellen. Het heeft weinig zin om deze registers verder uit te werken, omdat we de adressen uit deze registers eenvoudig kunnen uitlezen met de BASE-funktie. Dit werd aan het begin van dit hoofdstuk al gedemonstreerd.

De tekst kontrole registers:

Met deze registers kan de weergave van tekst onder SCREEN 0 tot en met 3 worden beïnvloed. We hebben hiervan al een voorbeeldje laten zien in paragraaf 5.2. Daarbij zagen we dat met VDP(13) de kleur van de letters kunnen bepalen en met VDP(14) het tempo waarin de kleur moet afwisselen.

De VDP kontrole registers:

In deze registers staat onder meer de ADJUST-waarde. Daar we deze waarde met behulp van het SET ADJUST statement kunnen wijzigen, heeft het geen zin verder op deze registers in te gaan.

De VDP status registers:

Uit deze registers kan alleen worden gelezen. De inhoud van deze registers kan onder meer aangeven of er zich een botsing van sprites voordoet, wat de koördinaten van de lichtpen of muis zijn en nog veel andere zaken. Alle informatie uit deze registers die voor ons als BASIC programmeurs interessant kan zijn, kan met behulp van reguliere BASIC statements worden opgevraagd.

De VDP programma registers:

Deze registers kunnen een (kort) programma bevatten. Dat programma is geschreven in een zeer speciale machinetaal instruktieset. Deze instruktieset wordt alleen door de VDP begrepen. De instrukties dienen door de Centrale processor in de registers te worden geladen, waarna ze door de Videoprocessor worden uitgevoerd. De instrukties hebben te maken met het verplaatsen van de videogeheugen inhoud en met de logische bewerkingen die op kleuren zijn toegestaan (zoals AND, OR, XOR enz.).

Daar het zinnig gebruik van deze registers alleen mogelijk is vanuit een machinetaal programma, zullen we verdere uitwerking van de VDP programma registers achterwege laten.

HOOFDSTUK 6

UITBREIDINGEN VAN MSX1-STATEMENTS

Behalve de uitbreidingen en wijzigingen op de grafische statements en funkties, zoals die al in de voorgaande hoofdstukken zijn behandeld, zijn ook de volgende statements en funkties uitgebreid:

RUN	LOAD	MERGE	SAVE
BLOAD	BSAVE	OPEN	PAD

In de volgende paragrafen zullen we de voor MSX2 geldende nieuwe formaten van deze statements en funkties nader bekijken.

6.1 Programma's wegschrijven, laden en starten

Tot nu toe zijn we er in onze serie leerboeken vanuit gegaan, dat de statements, waarin we een bestand moeten aangeven, alleen bestanden op kassette of schijf kunnen betreffen.

Wij zijn daarbij met opzet voorbijgegaan aan het feit, dat al onder MSX1 de mogelijkheid aanwezig was, daarnaast een kommunikatiepoort aan te geven. De kommunikatiepoort (RS232C-poort) was namelijk in geen enkele MSX1-computer standaard ingebouwd en de aanduiding van de kommunikatiepoort werd alleen "begrepen" door MSX1-computers waarop de kommunikatiepoort was aangesloten.

Nu er een aantal MSX2-computers standaard met een kommunikatiepoort zijn uitgerust, lijkt het ons een goed moment om de betreffende statements te kompleteren, niet alleen met de specifieke MSX2-informatie, maar ook met de RS232-informatie. We zullen daartoe in deze paragraaf eerst de statements SAVE, LOAD, MERGE en RUN nader bekijken, daarna de statements BSAVE en BLOAD en tenslotte het OPEN-statement.

SAVE, LOAD, MERGE en RUN

De syntax van deze statements is nog net zoals onder MSX1. Hieronder volgen van deze statements de schrijfwijzen:

```
SAVE "[<dev>:]<prognaam>"[,A]
LOAD "[<dev>:]<prognaam>"[,R]
MERGE "[<dev>:]<prognaam>"
RUN "[<dev>:]<prognaam>"[,R]
```

De tussen scherpe haakjes staande woorden moeten worden vervangen door een randapparaat-aanduiding (*dev*) of door een werkelijke programmanaam (*prognaam*). De tussen vierkante haken staande delen mogen worden weggelaten.

Het MERGE-statement werkt alleen met programmabestanden die als zogenaamde ASCII-bestanden zijn weggeschreven. Wanneer een programma vanuit het geheugen naar een randapparaat wordt geschreven met het SAVE-statement, en aan dat SAVE-statement is de optie ,A toegevoegd, dan zal dat programma als een ASCII-bestand op dat randapparaat worden opgeslagen.

De toevoeging ,R aan het LOAD-statement wil zeggen, dat het programma, nadat het van het randapparaat in het geheugen is geladen, onmiddellijk wordt gestart, alsof het RUN-statement was gebruikt.

Wanneer een programma met behulp van het RUN-statement vanaf het randapparaat in het geheugen wordt geladen en gestart, worden alle nog openstaande bestanden door het RUN-kommando gesloten. Indien u dit niet wenst, kunt u het RUN-statement laten volgen door de toevoeging ,R. Hierdoor blijven alle bestanden die door een vorig programma geopend werden, ondanks het RUN-kommando, toch open.

Tot zover is er nog geen nieuws onder de zon. Al deze zaken gelden zowel voor MSX1 als voor MSX2. Ook de syntax van deze statements is niet gewijzigd. Waar zit hem dan het verschil van deze statements in MSX2 ten opzichte van MSX1? Dat verschil zit hem in de mogelijke randapparaten (*dev*). Waar in voorgaande formaten de aanduiding *dev* staat, mag nu een van de volgende apparaatnamen worden ingevuld:

```
CAS, A, B, C, D, E, F, COM[<nr>] of MEM
```

CAS staat voor kassette, de letters A tot en met F staan voor een schijveneenheid. COM en MEM zijn nieuw.

COM[<*nr*>]:
COM staat voor kommunikatiepoort. In principe is het mogelijk meerdere kommunikatiepoorten te hebben. Vandaar dat de aanduiding COM dient te worden gevolgd door het poortnummer. Dit kan het nummer 0, 1 of 2 zijn. Indien geen nummer wordt gespecificeerd, wordt door het systeem poortnummer 0 aangenomen. In een later hoofdstuk wordt uitgebreid ingegaan op het gebruik van de kommunikatiepoorten. We zullen daarom hier niet verder ingaan op de mogelijkheden en onmogelijkheden.

MEM:
Dit is de aanduiding voor de memory disk, ook wel RAM-disk genoemd. Dit is de enige echte MSX2-uitbreiding op de genoemde statements.

Onder MSX2 is het namelijk mogelijk om de niet voor BASIC gebruikte delen van het interne geheugen te gebruiken alsof het een extern schijvengeheugen betreft. Daartoe zijn enkele nieuwe statements in MSX2 gedefinieerd, zoals CALL MEMINI, CALL MFILES, CALL MKILL en CALL MNAME. Door nu met CALL MEMINI een RAM-disk te definiëren, kan het gedefinieerde deel van het RAM-geheugen worden gebruikt als een floppy disk. Men kan bijvoorbeeld een programma vanuit het programmageheugen met een SAVE-kommando naar de RAM-disk schrijven door het kommando

```
SAVE "MEM:<prognaam>"
```

te geven. Ook kan men dit programma weer vanaf de RAM-disk teruglezen in het programma geheugen met een LOAD-statement, door in dat statement met MEM: aan te geven dat het programma op RAM-disk staat.

Het grote voordeel van het gebruik van RAM-disk is, dat het laden en wegschrijven veel sneller gaat dan naar een echte floppy-disk. Een nadeel is echter, dat de gehele inhoud van de RAM-disk verloren gaat bij het wegvallen van de spanning. U dient er dus voor te zorgen, dat de

computer niet wordt uitgeschakeld, zolang er nog belangrijke gegevens op de RAM-disk staan. Voordat u de computer uitschakelt zult u eerst de RAM-disk moeten kopiëren naar een floppy disk. Daartoe kunt u het COPY-statement als volgt gebruiken:

```
COPY "MEM:*.*" TO "A:"
```

Daar er een speciaal hoofdstuk van dit boek is gewijd aan het gebruik van de RAM-disk, zullen we er hier niet verder op ingaan.

BSAVE en BLOAD

Ook deze statements hebben nog hetzelfde formaat als onder MSX1. In tegenstelling tot de hiervoorgenoemde statements, is het voor BSAVE en BLOAD niet mogelijk met kommunikatiepoorten of RAM-disk te werken. De formaten zijn als volgt.

Voor het wegschrijven en teruglezen van machinetaalprogramma's:

```
BSAVE "[<devl>:]<prognaam>",<b>,<e>,<s>
BLOAD "[<devl>:]<prognaam>"[,R[,<displ>]]
```

Voor het wegschrijven en teruglezen van scherminhouden:

```
BSAVE "[<dev2>:]<filenaam>",<b>,<e>,S
BLOAD "[<dev2>:]<wfilenaam>",S
```

In voorgaande formaten is onderscheid gemaakt tussen *devl* en *dev2*. Dit is gedaan, omdat een scherminhoud niet naar kassette kan worden weggeschreven en dus ook niet teruggeladen. Waar *devl* staat dient u in te vullen: CAS, A, B, C, D, E of F. Voor *dev2* kunt u alleen A, B, C, D, E of F invullen.

Tot zover is er echter nog weinig nieuws verteld. U zult zich afvragen wat er dan onder MSX2 gewijzigd is aan deze statements. Welnu, wanneer u met de statements BSAVE en BLOAD een scherminhoud wilt wegschrijven of teruglezen, dan wordt onder MSX2 gekeken in welke schermmode de computer is ingesteld. Indien de computer in één van de modes SCREEN 4 tot en met SCREEN 7 is ingesteld, dan zal de

op dat moment aktieve schermpagina worden weggeschreven of zal de gelezen scherminformatie in de aktieve schermpagina worden geschreven. Dit hoeft dus niet dezelfde schermpagina te zijn, die op dat moment op het beeldscherm wordt weergegeven.

Het aktiveren van schermpagina's werd reeds in hoofdstuk 2 van dit boek behandeld. Het lijkt ons daarom niet nodig er hier nog verder op in te gaan.

Het OPEN-statement

Ook het OPEN-statement kent onder MSX2 enkele kleine uitbreidingen ten opzichte van MSX1. We zullen eerst weer het algemene formaat van het statement bekijken, om daarna te zien wat de uitbreidingen zijn.

```
OPEN "[<dev>:]<filenaam>" [FOR <mode>] AS [#]<nr> [LEN=<bytes>]
```

Zoals u ziet, is er aan het formaat helemaal niets gewijzigd. Waar in dit formaat het woordje *dev* staat, kan nu echter één van de volgende randapparaten worden aangegeven:

CRT	tekstscherm
GRP	grafisch scherm
LPT	printer
CAS	kassetterekorder
A t/m F	schijveneenheid
COM<*nr*>	kommunikatiepoort
MEM	RAM-disk

Verder moet u weten wat er gebeurt als u geen mode opgeeft. In dat geval wordt namelijk, indien CRT, GRP of LPT wordt opgegeven, de mode automatisch OUTPUT. Wanneer CAS wordt opgegeven zal de mode automatisch APPEND worden. Voor COM geldt dat de mode zowel INPUT als OUTPUT is. Wanneer tenslotte een disk of MEM wordt opgegeven, dan zal de mode automatisch RANDOM worden.

Voor het overige zijn er geen verschillen tussen het OPEN-statement in MSX1 en MSX2.

6.2 De PAD-funktie

Dit statement heeft zeer belangrijke uitbreidingen ondergaan in MSX2. Onder MSX1 kon met deze funktie alleen de status van een tekenbord, dat was aangesloten op één van de joystick-konnektoren, worden uitgelezen. Onder MSX2 kan nu ook een lichtpen, een muis of een track-ball worden uitgelezen. Al deze invoerapparaten worden op één van de joystick-konnektoren aangesloten. Hierdoor kan onze MSX-computer plotseling een groot aantal extra invoerorganen aan, zonder dat er dure optie-kaarten nodig zijn. Het formaat van de PAD-funktie is als volgt:

PAD (<*nummer*>)

Met het nummer geven we aan welk invoerapparaat we met de PAD-funktie willen lezen. Hieronder volgt een opsomming van de verschillende mogelijkheden.

Reeds bekend van MSX1 waren:

0 - 3	Tekenbord, aangesloten op joystick-konnektor 1
4 - 7	Tekenbord, aangesloten op joystick-konnektor 2

Nieuw onder MSX2 zijn:

8 - 11	Lichtpen
12 - 15	Muis of track-ball op joystick-konnektor 1
16 - 19	Muis of track-ball op joystick-konnektor 2

Lichtpen

Wanneer we de funktie PAD laten volgen door één van de cijfers 8 tot en met 11, dan kunnen we de status van een aangesloten lichtpen opvragen. Wat de betekenis van de verschillende cijfers precies is, zullen we nu gaan zien.

PAD(8)
Hiermee vragen we of de lichtpen koördinaten beschikbaar heeft. Indien het resultaat van deze funktie 0 is, dan zijn er geen koördinaten beschikbaar en hoeven we deze dus ook niet op te vragen. Wanneer er

wel koördinaten beschikbaar zijn, zal het resultaat van deze funktie –1 zijn. Nu heeft het zin om de beschikbare koördinaten op te vragen.

PAD(9)
Deze funktie geeft als resultaat de *x*-koördinaat van de lichtpen, ofwel de horizontale positie van de lichtpen op het scherm.

PAD(10)
Deze funktie geeft als resultaat de *y*-koördinaat van de lichtpen, ofwel de vertikale positie van de lichtpen op het scherm.

PAD(11)
Het resultaat van deze funktie is, dat we kunnen zien of de schakelaar op de lichtpen is ingedrukt of niet. Indien het resultaat 0 is, dan is de schakelaar niet ingedrukt. Is de schakelaar wel ingedrukt, dan zal het resultaat van deze funktie –1 zijn.

Muis en/of track-ball

De muis en de track-ball zijn twee funktioneel gelijkwaardige invoerorganen. In de navolgende beschrijving zullen we alleen nog de muis vermelden, mamar u mag daar evengoed track-ball voor lezen.

De muis is een heel handig aanwijsapparaat, dat vooral wordt gebruikt in tekenprogramma's en in zogenaamde "ikoon-gestuurde" programma's. We zullen aan het einde van dit hoofdstuk een voorbeeldje geven, waarin de muis wordt gebruikt, en waarin u het gemak van een muis kunt herkennen. Voordat we daaraan toe zijn zullen we echter eerst even moeten kijken naar de mogelijke PAD-funkties voor de muis.

PAD(12) en PAD(16)
Deze funkties (voor de muis, aangesloten op respektievelijk joystickkonnektors 1 en 2) geven altijd –1 als resultaat. Het is echter noodzakelijk deze funktie uit te voeren, voordat een PAD-funktie wordt uitgevoerd, waarmee de *x* en *y* koördinaten worden opgevraagd.

PAD(13) en PAD(17)
Deze funkties leveren de *x*-koördinaat van de muis op.

PAD(14) en PAD(18)
Deze funkties geven de *y*-koördinaat van de muis.

PAD(15) en PAD(19)
Deze funkties hebben altijd een 0 als resultaat en hebben eigenlijk geen enkele funktie.

Nu zal het u, als u wel eens een muis heeft gezien, zijn opgevallen dat er op die muis twee drukschakelaars zitten. Die schakelaars zijn niet met een PAD-funktie af te vragen. Het afvragen van de schakelaars zullen we met behulp van de funktie STRIG() moeten doen. Het volgende voorbeeld laat één en ander zien.

```
100 SCREEN 5,0: COLOR 15,4,4
110 OPEN "grp:" AS #1
120 X=128: Y=100
130 ON STRIG GOSUB ,440,,440
140 STRIG(1) ON: STRIG(3) ON
150 '
160 '**********************
170 '* SPRITE definieren *
180 '**********************
190 H$=""
200 FOR I=1 TO 8
210 READ H: H$=H$+CHR$(H)
220 NEXT I
230 DATA 224,192,160,16,8,4,2,1
240 SPRITE$(1)=H$
250 '
260 '**********************
270 '*   MUIS afvragen    *
280 '**********************
290 PUT SPRITE 1,(X,Y),8,1
300 M=PAD(12)
310 X=X+PAD(13): Y=Y+PAD(14)
320 IF XS=X AND YS=Y THEN GOTO 290
330 XS=X: YS=Y
340 LINE (0,0)-(31,15),4,BF
350 STRIG(1) OFF: STRIG(3) OFF
360 PSET(0,0): PRESET(0,0)
370 PRINT #1,X: PRINT #1,Y
```

```
380 STRIG(1) ON: STRIG(3) ON
390 GOTO 290
400 '
410 '**********************
420 '* STRIG afhandeling *
430 '**********************
440 IF STRIG(1) THEN PSET(X,Y):X1=X:Y1=Y
450 IF STRIG(3) THEN LINE(X,Y)-(X1,Y1),,B
460 RETURN
```

Met dit programma kunnen we vierkantjes op het grafisch scherm tekenen. Met de muis kunnen we een hoek en de diagonaal daar tegenover liggende hoek aangeven. Het is een goede demonstratie van het gemak waarmee een schermpositie kan worden bereikt, wanneer gebruik wordt gemaakt van een muis, in plaats van bijvoorbeeld de kursorbesturingstoetsen.

Omdat in dit korte programma alle elementen aanwezig zijn die bij het besturen van een muis van belang, zullen we dit programma stap voor stap doornemen.

De regels 290 tot en met 390 vormen een eindeloze lus, waarbinnen steeds opnieuw de positie van de muis wordt afgevraagd. In de regels 300 en 310 ziet u, hoe de positie van de muis kan worden opgevraagd met behulp van de PAD-funkties. De uiteindelijke koördinaten worden in de variabelen X en Y onthouden. In regel 120 kregen die beide variabelen een initiële waarde. Na uitvoering van de regels 300 en 310 zal de positie van de muis – en dus de inhoud van de variabelen X en Y – zijn gewijzigd, indien de muis werd bewogen.

We zien hier echter helemaal niets van. Het bewegen van de muis heeft namelijk niet automatisch tot gevolg, dat er iets op het scherm wordt afgedrukt. Vandaar, dat met regel 290 een sprite op het scherm wordt afgedrukt, en wel op de koördinaten die in variabelen X en Y staan. Nu kunnen we tenminste zien, waar de muis zich op het scherm bevindt.

De met regel 290 afgedrukte sprite wordt gedefinieerd met de regels 190 tot en met 240. De vorm van de sprite staat in de data-regel en stelt een pijltje voor.

We weten nu dus dat er een pijltje wordt afgedrukt op de koördinaten, zoals die in de variabelen X en Y staan, vlak voordat de nieuwe positie

van de muis wordt opgevraagd. Met regel 320 wordt gekeken of de positie van de muis is gewijzigd. Is de muis niet bewogen, dan wordt onmiddellijk teruggesprongen naar regel 290, waarmee het pijltje opnieuw wordt afgedrukt. Was de muis wel verplaatst, dan worden de regels 330 tot en met 390 uitgevoerd, voordat wordt teruggesprongen naar regel 290.

Met de regels 330 tot en met 390 worden de *x* en *y* koördinaten in de linker bovenhoek van het scherm afgedrukt. Omdat er tussen het positioneren van de kursor en het afdrukken van de koördinaten geen onderbreking mag voorkomen die tot gevolg zou kunnen hebben dat de kursorpositie wordt gewijzigd, wordt in regel 350 het detekteren van de druktoetsen op de muis even geblokkeerd. Deze blokkade wordt met regel 380 weer opgeheven.

Op de muis zitten twee schakelaars. In ons programma willen we die schakelaars gebruiken om er een bepaalde aktie mee te starten. De linker muisschakelaar gebruiken we om de eerste hoek van een vierkant mee aan te geven. Op het moment dat deze schakelaar wordt ingedrukt, moet komen vast te liggen, dat de op dat moment geldende *x* en *y* koördinaten van de muis een hoek van een vierkant aangeven. Met de rechter muisschakelaar willen we de hoek die diagonaal tegenover de eerste hoek ligt aangeven.

Om de schakelaars te kunnen gebruiken, moeten we ze eerst aktiveren. Bovendien moeten we aangeven, naar welke subroutine moet worden gesprongen, wanneer een geaktiveerde schakelaar is ingedrukt. U ziet in regel 130 dat wordt aangegeven, dat er naar regel 440 moet worden gesprongen, wanneer één van de schakelaars op de muis die op joystick-konnektor 1 is aangesloten, wordt ingedrukt. In regel 140 ziet u dat beide schakelaars op de muis worden geaktiveerd.

Wanneer nu tijdens het uitvoeren van de lus (regels 290 tot en met 390) een schakelaar op de muis wordt ingedrukt, dan zal de lus worden onderbroken en zal de subroutine op de regels 440 tot en met 460 worden uitgevoerd. In die subroutine wordt gekeken welke schakelaar werd ingedrukt, de linker [STRIG(1)] of de rechter [STRIG(3)]. Was de linker schakelaar ingedrukt, dan wordt de kursor verplaatst naar de koördinaten uit de variabelen X en Y. Bovendien worden die koördinaten bewaard in twee hulp-variabelen X1 en Y1. Was echter de rechter schakelaar ingedrukt, dan wordt een vierkantje getekend, waar-

van de ene hoek op de huidige koördinaten *x* en *y* ligt en de diagonaal tegenoverliggende hoek op de bewaarde koördinaten *x1* en *y1*. De RETURN-instruktie op regel 460 zorgt er voor, dat er wordt teruggekeerd naar het punt waar vandaan de subroutine was aangeroepen. Dat is in ons geval ergens in de lus van regel 290 tot en met 390.

Hiermee heeft u alle funkties gezien die van belang zijn voor het besturen van een muis. In elk programma met een muis zult u deze funkties tegenkomen. Zo ook in het volgende programma, dat weliswaar iets heel anders doet, maar dat grote overeenkomsten met het vorige programma heeft voor zover het de besturing van de muis betreft.

Wanneer u het volgende programma start, zult u bovenaan het scherm twee gekleurde blokken zien, met daarin de teksten "aanzwellend" en "wegstervend". In het midden van het scherm ziet u het pijltje dat met behulp van de muis kan worden verplaatst. Wanneer u nu met het pijltje een van beide teksten aanwijst, zal er onder die tekst een menu verschijnen. Nu kunt u ofwel de andere tekst aanwijzen of een regel uit het menu aanwijzen. Indien u dat laatste doet, zal er een geluid worden geproduceerd.

Dit programma geeft weer een heel andere toepassing van de muis, namelijk het gebruik van zogenaamde *"pop-down menu's"*. We zullen het programma niet verder uitleggen. Gewapend met de kennis uit het vorige programma moet u in staat zijn de muisbesturing in dit programma te herkennen.

```
100 SCREEN 5,0: COLOR 15,4,1
110 OPEN "grp:" AS #1
120 X=128: Y=100
130 ON STRIG GOSUB ,460,,810
140 STRIG(1) ON: STRIG(3) ON
150 '
160 '*********************
170 '* SPRITE definieren *
180 '*********************
190 H$=""
200 FOR I=1 TO 8
210 READ H: H$=H$+CHR$(H)
220 NEXT I
```

```
230 DATA 224,192,160,16,8,4,2,1
240 SPRITE$(1)=H$
250 '
260 '**********************
270 '*  Beeld opbouwen    *
280 '**********************
290 LINE(1,1)-(126,11),6,BF
300 LINE(128,1)-(254,11),7,BF
310 PSET(20,2):PRESET(20,2):COLOR 1,6:PRINT #1,"aanzwell
end"
320 PSET(148,2):PRESET(148,2):COLOR 1,7:PRINT #1,"wegste
rvend"
330 COLOR 15,4
340 '
350 '**********************
360 '*   MUIS afvragen    *
370 '**********************
380 PUT SPRITE 1,(X,Y),8,1
390 M=PAD(12)
400 X=X+PAD(13): Y=Y+PAD(14)
410 GOTO 380
420 '
430 '**********************
440 '*      STRIG (1)      *
450 '**********************
460 IF Y<12 AND X<127 THEN GOSUB 610
470 IF Y<12 AND X>127 THEN GOSUB 710
480 IF M1=0 OR X>127 THEN GOTO 540
490 PLAY "s13m2000"
500 IF Y>10 AND Y<30 THEN PLAY "o1cdefgab"
510 IF Y>30 AND Y<50 THEN PLAY "o2cdefgab"
520 IF Y>50 AND Y<70 THEN PLAY "o3cdefgab"
530 IF Y>70 AND Y<90 THEN PLAY "o4cdefgab"
540 IF M2=0 OR X<127 THEN GOTO 600
550 PLAY "s9m2000"
560 IF Y>10 AND Y<30 THEN PLAY "o5cdefgab"
570 IF Y>30 AND Y<50 THEN PLAY "o6cdefgab"
580 IF Y>50 AND Y<70 THEN PLAY "o7cdefgab"
590 IF Y>70 AND Y<90 THEN PLAY "o8cdefgab"
600 RETURN
610 '
620 '**********************
630 '*       menu 1        *
```

```
640 '**********************
650 M1=1
660 PSET(20,20):PRESET(20,20):PRINT #1,"oktaaf 1"
670 PSET(20,40):PRESET(20,40):PRINT #1,"oktaaf 2"
680 PSET(20,60):PRESET(20,60):PRINT #1,"oktaaf 3"
690 PSET(20,80):PRESET(20,80):PRINT #1,"oktaaf 4"
700 RETURN
710 '
720 '**********************
730 '*      menu 2        *
740 '**********************
750 M2=1
760 PSET(148,20):PRESET(148,20):PRINT #1,"oktaaf 5"
770 PSET(148,40):PRESET(148,40):PRINT #1,"oktaaf 6"
780 PSET(148,60):PRESET(148,60):PRINT #1,"oktaaf 7"
790 PSET(148,80):PRESET(148,80):PRINT #1,"oktaaf 8"
800 RETURN
810 '
820 '**********************
830 '*     STRIG (3)      *
840 '**********************
850 LINE (0,12)-(255,100),4,BF
860 M1=0: M2=0
870 RETURN
```

HOOFDSTUK 7

DE KLOK-CHIP

In iedere MSX2-computer zit een zogenaamde klok-chip. Dit is een IC (chip), waarin ondermeer een klok zit. Naast de klok zit er in diezelfde chip ook nog een geheugentje. In dat geheugentje worden een aantal gegevens onthouden. Dat de gegevens in dat geheugentje niet verloren gaan – ook niet wanneer de computer gedurende langere tijd niet aan de netspanning is aangesloten – komt doordat er een klein batterijtje in de computer zit. Dat batterijtje wordt automatisch opgeladen, zodra de computer wordt aangeschakeld. Wordt de computer uitgeschakeld, dan voedt het batterijtje de klok-chip. Hierdoor kan de klok altijd doorlopen en kunnen de gegevens in het geheugentje bewaard blijven.

De klok in de chip houdt niet alleen de tijd bij, maar ook de datum. Er zijn speciale BASIC-statements aanwezig, waarmee de tijd en de datum uit de klok-chip kunnen worden gelezen, maar er zijn ook statements, waarmee een nieuwe datum en tijd in de chip kan worden geschreven. We zullen die statements in de laatste paragraaf van dit hoofdstuk behandelen.

De gegevens, die in het geheugentje van de klok-chip kunnen worden opgeslagen, zijn in vier groepen onder te verdelen. Die groepen zijn:

- Initialisatie (SET PASSWORD, PROMPT, TITLE)
- Geluid (SET BEEP)
- Video (SET ADJUST)
- Scherminstelling (SET SCREEN)

Aan elk van deze groepen zullen we een aparte paragraaf wijden.

7.1 Initialisatie

Onder initialisatie verstaan we al die zaken die gebeuren tussen het aanschakelen van de computer en het moment waarop we het eerste kommando aan de computer kunnen geven. Gedurende die tijd voert de computer een machinetaalroutine uit, die is opgeslagen in het ROM *(Read Only Memory)*. Die routine test de belangrijkste onderdelen van de computer, zoals het geheugen. Daarna leest die routine de inhoud van de klok-chip.

Eerst wordt een specifieke positie gelezen, waarin een kode staat, die aangeeft welke soort informatie het tekst-buffertje in de klok-chip bevat. Dit buffertje kan één van de volgende drie soorten informatie bevatten:

1. Een wachtwoord *(password)*
2. Een titel
3. Een prompt

Het tekstbuffertje is slechts 6 tekens groot. Dit houdt in, dat ongeacht de soort, wachtwoord, titel of prompt, de maximale lengte 6 tekens is.

Wanneer de tekst in het buffertje een wachtwoord is, dan zal de computer na het afdrukken van het openingsscherm eerst vragen naar dat wachtwoord. Hiertoe drukt de computer de tekst "Password:" af. Het woord, dat u nu intikt, zal worden vergeleken met het woord in het tekstbuffertje van de klok-chip. Komen beide woorden niet overeen, dan zal de computer niet verder gaan met de initialisatie-procedure, maar blijven wachten tot het juiste wachtwoord wel is ingetikt.

Het kommando waarmee een wachtwoord in de klok-chip kan worden gezet, ziet er als volgt uit:

```
SET PASSWORD "<wachtwoord>"
```

Voordat u dit kommando ingeeft dient u zich te realiseren, dat na dit kommando de computer alleen nog maar volledig kan funktioneren wanneer u het juiste wachtwoord heeft ingetikt. U mag het eenmaal ingestelde wachtwoord dus absoluut niet vergeten.

Zou u het wachtwoord toch vergeten, dan zit er niet veel anders op dan de computer gedurende een zeer lange tijd (meer dan een maand) niet te gebruiken. Na die tijd is de batterij misschien leeg en weet de computer niet meer dat er een wachtwoord was ingesteld. De tijd die u moet wachten kan van computer tot computer verschillen. De tijd is geheel afhankelijk van de kwaliteit van de batterij en de mate waarin deze was opgeladen. Een andere mogelijkheid is, de batterij kort te sluiten. Dit is echter niet aan te raden, omdat u daartoe de computer moet openen en omdat een te snelle ontlading schadelijk is voor de batterij. Kortom, onthoudt u het wachtwoord, dan bespaart u zich een hoop ellende.

Zoals gezegd, is er slechts één tekstbuffer. Wanneer u in die buffer een titel zou zetten, dan kan daar geen wachtwoord meer in staan. Onder een titel verstaan we het woord, dat in het openingsscherm kan worden afgedrukt. Om zo'n woord in het openingsscherm op te nemen, dient het volgende statement te worden gegeven:

```
SET TITLE ["<titel>"][,<wkleurnr>]
```

Stel, dat u dit kommando geeft, met tussen de aanhalingstekens het woord WELKOM. Wanneer u nu op de Reset-toets drukt, of de computer uit- en aanuitschakelt, dan zult u in het openingsscherm het woord WELKOM zien staan. Indien u nu niets meer doet, zal het openingsscherm gewoon blijven staan en start de computer niet verder op. Dit komt doordat de titel precies 6 tekens lang is. In dat geval wacht het systeem totdat er een willekeurige toets is ingedrukt, voordat het verder gaat met initialiseren. Zou u een titel van minder dan zes tekens hebben ingetikt, dan blijft het openingsscherm slechts korte tijd staan en gaat het systeem gewoon verder met initialiseren.

Door aan het statement een kleurnummer toe te voegen, kunt u het titelscherm een bepaalde kleur geven. Alleen de nummers 1 tot en met 4 zijn toegestaan. Bij een standaard instelling van de kleurenpalet zal dat resulteren in de volgende kleuren:

1	blauw
2	groen
3	rood
4	geel

Laat u het kleurnummer weg, dan wijzigt de kleur niet. Alleen het titelwoord wordt dan aangepast. Wilt u alleen de kleur van het scherm wijzigen, dan kunt u de titel weglaten en alleen een komma, gevolgd door een kleurnummer aan het statement toevoegen.

Wanneer de computer gereed is een kommando van u te aksepteren, dan laat hij dat weten door op het scherm het woordje **Ok** af te drukken. We noemen dit de *"prompt"*. Die prompt kunnen we veranderen met het volgende kommando:

```
SET PROMPT "<prompt>"
```

Het zal inmiddels duidelijk zijn, dat wanneer we het tekstbuffertje laden met een prompt, er geen wachtwoord of titel meer in kan staan. Wanneer één van de tot nu toe genoemde drie soorten teksten in het buffertje staat, kunnen de beide andere soorten teksten er niet meer in staan.

Probeert u maar eens een andere prompt te maken, bijvoorbeeld de prompt "bevel?". Geeft u hiertoe het kommando *SET PROMPT "bevel?"* in, gevolgd door het indrukken van de Return-toets. U zult nu zien dat de computer terugkomt met het woord **bevel?** in plaats van met het woordje *Ok*. Wanneer u nu de computer reset, zult u zien dat er geen titel meer op het openingsscherm wordt afgedrukt. Wel onthoudt de computer de laatst ingestelde kleur van het openingsscherm.

Zou u hierna weer een wachtwoord of titel instellen, dan verdwijnt de prompt "bevel?" weer. Wanneer er geen prompt in de klok-chip staat, gebruikt het systeem de standaard prompt "Ok".

7.2 Geluid

Tijdens de aanschakelprocedure geeft de computer een attentie-signaal. Daartoe wordt het BEEP-kommando uitgevoerd. U kunt dat kommando ook zelf geven, door eenvoudigweg het kommando BEEP in te tikken. In MSX1-computers was die BEEP altijd een korte toon. In MSX2-computers zijn meerdere soorten tonen beschikbaar. U kunt die tonen kiezen met het volgende statement:

```
SET BEEP [<toonsoort>][,<volume>]
```

Aan het formaat van dit statement ziet u, dat u zowel de toonsoort als het volume kunt instellen. Beide gegevens worden in de klok-chip opgeslagen. Wanneer u daarna het kommando BEEP geeft, zal de klok-chip worden geraadpleegd naar de soort toon en het volume daarvan, voordat de BEEP ten gehore wordt gebracht. De eenmaal ingestelde BEEP blijft dus ook geldig nadat de computer uitgeschakeld is geweest.

De vier toonsoorten zijn:

1	Korte ééntonige piep
2	Galmende piep
3	Tweetonige piep
4	Drietonige piep

Er zijn vier instelmogelijkheden voor het volume, van 1 tot en met 4. Instelling 1 is het kleinste volume, instelling 4 het hoogste.

Het volgende programma maakt gebruik van alle SET BEEP mogelijkheden. U kunt er alle toonsoorten op alle volumes mee horen. Zodra u een pieptoon hoort, die u bevalt, drukt u op de spatiebalk, waarna die toonsoort in de klok-chip wordt opgeslagen met het SET BEEP statement. Zie daartoe de regels 160 en 370 in het programma.

```
100 CLS: SCREEN 0: WIDTH 80
110 PRINT "Ik ga u nu een aantal verschillende soorten B
EEPS laten horen."
120 PRINT "Druk op de spatiebalk om de op dat moment ten
gehore gebrachte"
130 PRINT "BEEP in de klok-chip vast te leggen."
140 FOR K=1 TO 1000: NEXT K
150 FOR J=1 TO 4
160 SET BEEP J
170 FOR I=1 TO 5
180 FOR K=1 TO 100: NEXT K
190 BEEP
200 I$=INKEY$:IF I$=" " THEN GOTO 240
210 NEXT I
220 NEXT J
230 GOTO 150
240 PRINT "Dit is de gekozen BEEP..."
250 FOR K=1 TO 250: NEXT K
```

```
260 FOR I=1 TO 5
270 FOR K=1 TO 100:NEXT K
280 BEEP
290 NEXT I
300 CLS
310 PRINT "Ik ga u deze BEEP op verschillende volumes la
ten horen."
320 PRINT "Druk weer op de spatiebalk om het gewenste vo
lume van de BEEP"
330 PRINT "in de klok-chip vast te leggen."
340 FOR K=1 TO 1000: NEXT K
350 FOR J=1 TO 4
360 FOR K=1 TO 100: NEXT K
370 SET BEEP ,J
380 FOR I=1 TO 5
390 FOR K=1 TO 100: NEXT K
400 BEEP
410 I$=INKEY$: IF I$=" " THEN GOTO 450
420 NEXT I
430 NEXT J
440 GOTO 350
450 PRINT "Dit is het gekozen volume..."
460 FOR K=1 TO 250: NEXT K
470 FOR I=1 TO 5
480 FOR K=1 TO 100: NEXT K
490 BEEP
500 NEXT I
510 CLS: END
```

7.3 Video

Het beeldschermplaatje dat door de computer wordt geproduceerd, zit precies in het midden van het televisie- of monitorbeeld. Wanneer echter een televisie of monitor niet goed is afgeregeld, dan zit dat televisie- of monitorbeeld niet precies midden op de beeldbuis. Hierdoor zou het door de computer geproduceerde beeld voor het oog ook 'verschoven' op de beeldbuis kunnen staan. MSX2-computers hebben nu de mogelijkheid om het computerbeeld iets te verschuiven ten opzichte van het televisie- of monitorbeeld. Op die manier kan het plaatje voor het oog precies in het midden van de beeldbuis worden geplaatst.

Dit verschuiven van het plaatje op de beeldbuis kan worden gedaan met het volgende statement:

```
SET ADJUST (<horizontaal>,<vertikaal>)
```

Wanneer het plaatje in het midden staat, hebben zowel horizontaal als vertikaal de waarde 0. Willen we het plaatje iets naar links sturen, dan moeten we horizontaal een negatieve waarde geven. Om naar rechts van het midden te schuiven moeten we horizontaal een positieve waarde geven. De toegestane waarden lopen van –7 tot en met +8. Willen we het plaatje naar boven schuiven, dan moeten we vertikaal een negatieve waarde geven. Om het plaatje beneden het midden te schuiven geven we vertikaal een positieve waarde. Ook deze waarden lopen van –7 tot en met +8.

Het volgende programma stelt u in staat het beeld voor het oog precies in het midden van het scherm in te stellen, waarna de gekozen instelling in de klok-chip wordt opgeslagen (zie regel 190).

```
100 A=0: B=0
110 COLOR 15,1,15: SCREEN 1
120 I$=INKEY$:IF I$="" THEN GOTO 120
130 IF I$=" " THEN A=0:B=0
140 IF I$=CHR$(29) AND A>-7 THEN A=A-1
150 IF I$=CHR$(28) AND A<8 THEN A=A+1
160 IF I$=CHR$(30) AND B>-7 THEN B=B-1
170 IF I$=CHR$(31) AND B<8 THEN B=B+1
180 IF I$=CHR$(27) THEN GOTO 210
190 SET ADJUST (A,B)
200 GOTO 120
210 SCREEN 0
220 END
```

7.4 Scherminstelling

Een aantal gegevens, die te maken hebben met de manier waarop het beeldscherm wordt opgebouwd, kunnen in de klok-chip worden opgeslagen. Eenmaal opgeslagen, zal die schermopbouw steeds opnieuw, na het aanschakelen van de computer, worden toegepast. Het formaat van

het statement waarmee de scherminstellingen in de klok-chip worden opgeslagen is heel eenvoudig, namelijk:

```
SET SCREEN
```

Hieraan kunnen geen parameters worden toegevoegd. Alle schermgegevens die op het moment van uitvoering van het SET SCREEN statement zijn ingesteld worden namelijk in de klok-chip geschreven. Het is dus zaak, om het scherm eerst met de daartoe aangewezen statements in te stellen, om daarna het SET SCREEN statement uit te voeren.

De statements, waarmee u schermgegevens kunt wijzigen, die met het SET SCREEN statement in de klok-chip worden geschreven, zijn:

SCREEN
WIDTH
COLOR
KEY ON/OFF

Het volgende programma laat de funktie van het SET SCREEN statement zien. Nadat het programma u een aantal vragen heeft gesteld over schermbreedte, kleuren, funktietoetsen en dergelijke (regels 110 tot en met 270) worden die gegevens in de daaropvolgende regels ingesteld. In regel 280 wordt de kleur ingesteld met een COLOR statement. In regel 290 wordt de weergave van de funktietoetsteksten met KEY ON/OFF ingesteld. In regel 300 wordt de schermmode (SM), de toetsklik (TK) en de printersoort (MP) ingesteld met behulp van het SCREEN statement. De regellengte (het aantal tekens per regel) wordt tenslotte ingesteld met het WIDTH statement van regel 310.

Wanneer u nu op de vraag of de gekozen instellingen juist zijn met "J" antwoordt, dan zal met regel 350 het SET SCREEN statement worden uitgevoerd, waarmee alle ingestelde schermgegevens in de klok-chip worden opgeslagen.

```
100 CLS
110 INPUT "Welke schermbreedte wenst u    (1<=80)";SB
120 IF SB<32 THEN SM=1 ELSE SM=0
130 INPUT "Welke voorgrondkleur wenst u    (1-15)";VK
140 IF VK<1 OR VK>15 THEN VK=1
```

```
150 INPUT "Welke achtergrondkleur wenst u  (1-15)";AK
160 IF AK=VK THEN AK=AK+1
170 IF AK<1 OR AK>15 THEN AK=15
180 INPUT "Welke randkleur wenst u         (1-15)";RK
190 IF RK<1 OR RK>15 THEN RK=15
200 INPUT "Wilt u de funktietoetsen zien    (j/n)";F$
210 IF F$<>"j" AND F$<>"J" AND F$<>"n" AND F$<>"N" THEN 200
220 INPUT "Wilt u de toetsklik horen        (j/n)";T$
230 IF T$<>"j" AND T$<>"J" AND T$<>"n" AND T$<>"N" THEN 220
240 IF T$="J" OR T$="j" THEN TK=1 ELSE TK=0
250 INPUT "Heeft u een MSX-printer          (j/n)";M$
260 IF M$<>"j" AND M$<>"J" AND M$<>"n" AND M$<>"N" THEN 250
270 IF M$="J" OR M$="j" THEN MP=0 ELSE MP=1
280 COLOR VK,AK,RK
290 IF F$="j" OR F$="J" THEN KEY ON ELSE KEY OFF
300 SCREEN SM,,TK,,MP
310 WIDTH SB
320 CLS
330 INPUT "Is het zo goed (j/n)";I$
340 IF I$<>"j" AND I$<>"J" AND I$<>"n" AND I$<>"N" THEN 330
350 IF I$="j" OR I$="J" THEN SET SCREEN ELSE GOTO 100
360 END
```

7.5 Datum en tijd

De belangrijkste funktie van de klok-chip is het opslaan en bijhouden van de datum en de tijd. Voor het instellen van een nieuwe datum en tijd staan de volgende statements ter beschikking:

```
SET DATE "<datum>"[,A]
SET TIME "<tijd>"[,A]
```

De momentele datum en tijd kunnen uit de klok-chip worden gelezen met de volgende statements:

```
GET DATE <var.naam>$[,A]
GET TIME <var.naam>$[,A]
```

Hoe we deze statements gebruiken en wat de betekenis van de verschillende parameters is, zullen we hierna gaan zien. We zullen beginnen met het instellen en lezen van de datum. Daarna komt de tijd aan de beurt.

Datum

De datum wordt in verschillende landen in de wereld op verschillende manieren geschreven. In Nederland hebben de meeste mensen de gewoonte om een datum te beginnen met de dag van de maand, dan de maand en tenslotte het jaar. In veel andere landen is het de gewoonte om te beginnen met de maand, dan de dag van de maand en tenslotte het jaar. Er zijn echter ook landen, waar men begint met het jaartal, dan de maand en tenslotte de dag.

MSX2-computers worden in verschillende versies gemaakt, afhankelijk van het land waarvoor ze worden gemaakt. In de MSX-standaard is echter vastgelegd, dat iedere fabrikant in het ROM-geheugen moeten aangeven welk datum-formaat de betreffende computer gebruikt. Hiertoe zijn drie bitjes in ROM-adres &H2B belegd. Het betreft de bitjes 6, 5 en 4. Het adres &H2B is eenvoudig uit te lezen met een PEEK-funktie. De drie bitjes kunnen dan met behulp van een AND-bewerking uit het gelezen byte worden afgezonderd. Eén en ander gaat als volgt in zijn werk:

```
&H2B:  x x x x x x x x
       0 1 1 1 0 0 0 0
       ---------------- AND
       0 x x x 0 0 0 0 / 16 = x x x
```

De drie bitjes die we willen afzonderen, worden met een 1 ge-AND en alle overige bitjes met een 0. Het resultaat zal zijn, dat alle bitjes 0 zijn, behalve de bitjes die met een 1 werden ge-AND. Die laatste bitjes behouden hun oorspronkelijke waarde. Daar de waarde van het meest rechtse bitje 16 is, delen we het geheel door 16, waarna het resultaat kan variëren van 0 tot 7 (xxx). In de MSX-standaard zijn nu als volgt drie waarden vastgelegd voor drie verschillende datumformaten:

0 = JJ/MM/DD
1 = MM/DD/JJ
2 = DD/MM/JJ

Het volgende programma zoekt voor u uit wat het in uw computer gebruikte datum-formaat is. U doet er goed aan dit programma een keer in te tikken en uit te voeren, omdat u voor een juist gebruik van de SET en GET DATE statements dit formaat moet kennen.

```
10 A=(PEEK(&H2B) AND (64+32+16))/16
20 PRINT "Het datum-formaat van deze computer is: ";
30 IF A=0 THEN PRINT "JJ/MM/DD"
40 IF A=1 THEN PRINT "MM/DD/JJ"
50 IF A=2 THEN PRINT "DD/MM/JJ"
60 END
```

U weet nu, welk datum-formaat uw computer gebruikt. In de hiernavolgende behandeling van GET en SET DATE zullen we het formaat MM/DD/JJ gebruiken. Mocht uw computer een ander formaat vereisen, dan dient u het hier gebruikte formaat steeds te vervangen door het voor uw computer geldige formaat.

SET DATE

Het SET DATE statement dient te worden gevolgd door een string, waarin de in te stellen datum staat. Zou men bijvoorbeeld de datum 18 april 1989 willen instellen, dan dient men het volgende statement te geven:

```
SET DATE "04/18/89"
```

Hierin vallen ons twee dingen op. Het eerste is, dat de dagen, maanden en jaren van elkaar zijn gescheiden door een breukstreep. Het tweede is, dat er altijd twee cijfers worden gebruikt, ook als er eigenlijk slechts 1 cijfer is. Dat ene cijfer wordt dan aangevuld met voorlopende nullen.

Het is natuurlijk ook toegestaan om in plaats van een string een variabele te gebruiken. Die variabele dient dan van te voren te zijn geladen met de gewenste datum. Een voorbeeld hiervan is:

```
INPUT "Datum";D$
SET DATE D$
```

Volgens het officiële formaat van het SET DATE statement, kan hieraan nog een parameter worden toegevoegd. Dat is de ,A optie. Wanneer die optie wordt toegevoegd, wordt een zogenaamde alarm-datum ingesteld. Er zijn echter geen specifieke alarm-statements in MSX2 gedefinieerd, zodat er van deze optie weinig gebruik wordt gemaakt.

GET DATE

Zo goed als we een datum in de klok-chip kunnen laden, kunnen we deze er ook weer uit lezen. De datum in de klok-chip zal elke dag automatisch worden aangepast. Hebben we eenmaal de juiste datum en tijd in de klok-chip geladen, dan kunnen we daarna altijd de juiste datum uit de klok-chip teruglezen met het GET DATE statement. Daartoe dienen we aan het statement de naam van een string-variabele toe te voegen. De datum wordt dan vanuit de klok-chip in die variabele gecopieerd. Een voorbeeld is:

```
GET DATE D$
PRINT D$
```

Nu staat de datum in variabele D$, geheel volgens het aan het begin van deze paragraaf vermelde formaat. We kunnen deze datum dan ook zonder meer afdrukken met een PRINT-statement.

Ook aan het GET DATE statement kan de ‚A optie (alarm-optie) worden toegevoegd. Zou u dat doen, dan zal het resultaat zijn, dat alleen de dag van de maand in de opgegeven variabele wordt gezet.

Tijd

De tijd wordt altijd geschreven in de volgorde: uren, minuten en sekonden. Dit geldt voor alle computers. De uren, minuten en sekonden dienen van elkaar te worden gescheiden door een dubbele punt. Voor ieder van de komponenten (uren, minuten en sekonden) geldt, dat er altijd twee cijfers moeten worden geschreven, ook als het getal kleiner is dan 10.

Het volgende voorbeeld laat zien, hoe de tijd in de klok-chip kan worden geladen. Let daarbij vooral op de schrijfwijze (dubbele punten en voorlopende nullen).

```
SET TIME "09:44:02"
```

Wanneer u nu de computer uit- en aan zou zetten, dan zult u zien dat de tijd, ondanks dat de computer heeft uitgestaan, netjes is bijgehouden. Het uitlezen van de tijd kunt u met de volgende statements uitvoeren:

```
GET TIME T$
PRINT T$
```

Ook aan de SET en GET TIME statements kan de ,A optie worden toegevoegd. Wanneer deze optie is toegevoegd, worden alleen de uren en minuten in de klok-chip geladen en er weer uitgelezen.

7.6 Een analoge klok

Ter afsluiting van dit hoofdstuk nog een voorbeeldprogramma, waarin de statements worden gebruikt waarmee de tijd en de datum kunnen worden ingesteld en uitgelezen.

Met de regels 110 tot en met 130 kan een nieuwe tijd worden ingesteld. Vervolgens wordt met de regels 210 tot en met 410 een klok op het beeldscherm getekend. De regels 460 tot en met 780 bepalen de stand van de wijzers (uren-, minuten- en sekondenwijzer). Hiertoe wordt met regel 460 de tijd uit de klok-chip gelezen. Vanaf regel 470 wordt het aantal uren uit de tijd-string gedestilleerd. Vanaf regel 550 gebeurt hetzelfde voor het aantal minuten en voor de sekonden gebeurt dat vanaf regel 640.

Vanaf regel 800 vindt u een subroutine, die u waarschijnlijk goed zult kunnen gebruiken in uw eigen toepassingen. Het is een routine waarmee een punt op een cirkelomtrek kan worden bepaald. Het in de listing opgenomen kommentaar zal voldoende zijn om met deze routine te kunnen werken.

```
100 X=128: Y=95
110 INPUT "Wilt u een tijd instellen";I$
120 IF I$<>"j" AND I$<>"J" THEN GOTO 150
130 INPUT "Nieuwe tijd (UU:MM:SS)";T$
140 SET TIME T$
150 OPEN "grp:" AS #1
160 COLOR 9,15: CLS: SCREEN 4
170 '
180 '********************************
190 '* OPBOUWEN KLOK OP BEELDSCHERM *
200 '********************************
210 AF=1.35
```

```
220 CIRCLE (X,Y),80,1,,,AF
230 CIRCLE (X,Y),50,9,,,AF
240 R=65
250 FOR GR=0 TO 330 STEP 30
260 U=GR/30+3: IF U>12 THEN U=U-12
270 U$=STR$(U)
280 U$=RIGHT$(U$,LEN(U$)-1)
290 GOSUB 800
300 IF LEN(U$)=2 THEN XO=XO-6
310 PSET(XO,YO)
320 PRESET(XO,YO):PRINT #1,U$;
330 NEXT GR
340 PAINT (X,Y),9
350 DRAW "BM204,140"
360 PRINT #1,"TIJD"
370 GET DATE D$
380 DRAW "BM20,140"
390 PRINT #1,"DATUM"
400 DRAW "BM8,150"
410 PRINT #1,D$
420 '
430 '*****************************
440 '* BEPALEN STAND DER WIJZERS *
450 '*****************************
460 GET TIME T$
470 UU$=LEFT$(T$,2)
480 UU=VAL(UU$):IF UU=>12 THEN UU=UU-12
490 GR=30*UU+VAL(MID$(T$,4,2))/2+270
500 IF GR>360 THEN GR=GR-360
510 R=20
520 GOSUB 800
530 XU=XO: YU=YO
540 MI$=MID$(T$,4,2)
550 MI=VAL(MI$)
560 GR=6*MI+270
570 IF GR>360 THEN GR=GR-360
580 R=35
590 GOSUB 800
600 XM=XO: YM=YO
610 LINE (200,150)-(240,157),15,BF
620 DRAW "BM200,150"
630 PRINT #1,LEFT$(T$,5);
640 SE$=RIGHT$(T$,2)
```

```
650 SE=VAL(SE$)
660 GR=6*SE+270
670 IF GR>360 THEN GR=GR-360
680 R=50
690 GOSUB 800
700 XS=XO: YS=YO
710 GET TIME T$
720 LINE (X,Y)-(XS,YS),15
730 LINE (X,Y)-(XM,YM),15
740 LINE (X,Y)-(XU,YU),15
750 IF SE$<>RIGHT$(T$,2) THEN LINE (X,Y)-(XS,YS),9: IF P
LAY(0)=0 THEN PLAY "o7t255s9m200164b":GOTO 640 ELSE GOTO
 640
760 IF MI$<>MID$(T$,4,2) THEN PLAY "l32t128s14m2000o414c
":LINE (X,Y)-(XM,YM),9:GOTO 540
770 IF UU$<>LEFT$(T$,2) THEN FOR I=0 TO UU:PLAY "t12812s
9m4000o2c":NEXT I:LINE (X,Y)-(XU,YU),9:GOTO 470
780 GOTO 710
790 '
800 '*******************************
810 '* bepaal punt op cirkelomtrek *
820 '*-----------------------------*
830 '* input:  middelpunt (X,Y)    *
840 '*         straal (R)          *
850 '*         afpl.factor (AF)    *
860 '*         hoek graden (GR)    *
870 '*-----------------------------*
880 '* output: hor. pos.   (XO)    *
890 '*         vert. pos.  (YO)    *
900 '*******************************
910 RAD=GRADEN*.017453
920 XO=X+(R*COS(RAD))/AF
930 YO=Y+R*SIN(RAD)
940 RETURN
```

HOOFDSTUK 8

DE RAM-DISK

Zoals de titel van dit hoofdstuk al zegt, gaan we het nu hebben over een disk, die in het geheugen van onze computer wordt gesimuleerd. Dat betekent, dat een deel van het geheugen gaat worden gebruikt alsof het een schijveneenheid was. Omdat de toegang tot het geheugen veel sneller is dan de toegang tot een schijveneenheid, kunnen we stellen, dat een RAM-disk een heel snelle schijveneenheid is. Bovendien, omdat het lezen uit of schrijven naar het geheugen geheel elektronisch gaat, is die RAM-disk absoluut geruisloos.

De term RAM-disk is al heel lang een bekende term in de Personal Computer wereld. Voor die mensen onder u, die naast MSX ook te maken hebben met PC's geldt de volgende waarschuwing. Een RAM-disk in een PC is niet hetzelfde als een RAM-disk in een MSX2-computer. Het belangrijkste verschil zit namelijk daarin, dat een RAM-disk in een PC onder het operating systeem MS-DOS kan worden gebruikt, terwijl de RAM-disk in de MSX2-computer alleen onder MSX-BASIC kan worden gebruikt.

In de loop van dit hoofdstuk zullen we achtereenvolgens gaan zien wat een MSX RAM-disk is, hoe deze kan worden gemaakt en hoe deze kan worden gebruikt. Zowel bij het maken van een RAM-disk als bij het gebruiken daarvan komen een aantal nieuwe MSX2-statements aan de orde.

8.1 Wat is een RAM-disk?

Alle MSX2-computers zijn met minimaal 64 kBytes RAM-geheugen uitgerust. De meesten zelfs met 128 kBytes. Wanneer u echter de computer opstart onder BASIC, dan zal het u zijn opgevallen dat er slechts tussen de 23 en 28 kBytes aan vrij te gebruiken RAM-geheugen beschikbaar is. Het werkelijke aantal vrije geheugenplaatsen

is afhankelijk van het aantal aangesloten schijveneenheden. Heeft u geen schijveneenheden, dan zal er ongeveer 28 kBytes vrije ruimte zijn. Voor iedere schijveneenheid is echter een stukje systeemgeheugen nodig, waarin de administratie van die schijveneenheid wordt bijgehouden en waarin de gelezen sektoren tijdelijk kunnen worden opgeslagen. Zodoende kan de vrije geheugenruimte teruglopen tot 23 kBytes of zelfs nog minder.

Wat nu opvalt is, dat er nooit meer dan 28 kBytes vrije ruimte is, terwijl we toch een 64 kBytes of nog groter RAM-geheugen hebben. Hoe komt dit? Voor het antwoord hierop moeten we nog even teruggaan naar de geheugenindeling van MSX-computers. De Z80 mikroprocessor kan slechts 64 kBytes geheugen adresseren. Wanneer we nu MSX-BASIC opstarten, dan worden alle adressen van 0 tot en met 32767 gebruikt om er het ROM-geheugen, met daarin het basis I/O-systeem en de BASIC-interpreter, mee te adresseren. Hierdoor blijven er nog slechts 32768 adressen over, waarmee het RAM kan worden geadresseerd. Het basis I/O-systeem moet een aantal variabele gegevens bijhouden. Dit kan alleen maar in een RAM-geheugen. Daarom zal het basis I/O-systeem een deel (ongeveer 4000 bytes) van het RAM-geheugen gebruiken voor administratie. Het gevolg is, dat er slechts ruim 28000 bytes overblijven voor de gebruiker.

Wanneer we nu een RAM-geheugen van 64 kBytes of meer hebben, dan lopen de adressen van dat geheugen van 0 tot en met 65535. We zagen echter, dat de adressen van het ROM van 0 tot en met 32767 liepen. Die lopen dus parallel aan het RAM. Omdat we BASIC hebben geaktiveerd, heeft de ROM prioriteit gekregen boven het RAM. Elk adres onder de 32 kBytes zal dan ook een adres van het ROM zijn.

Door nu wat speciale elektronika aan de computer toe te voegen, is het mogelijk om over te schakelen van ROM naar RAM. Daartoe zijn dan wel speciale machinetaalroutines nodig. Deze routines worden door de uitgebreide BASIC-instruktieset van de MSX2-computer automatisch aangeroepen. Hierdoor is het mogelijk om met speciaal daarvoor bestemde statements het onder BASIC normaal niet-gebruikte deel van het RAM-geheugen toch te gebruiken.

U zult het nu wel hebben geraden, de RAM-disk is in feite niets anders is dan het geheugendeel dat normaal niet te bereiken is vanuit BASIC. De statements die met die RAM-disk hebben te maken, behandelen dat

geheugendeel alsof het een schijveneenheid is. Men kan er dus bestanden en programma's naar toe schrijven, er een inhoudsopgave van opvragen, bestanden verwijderen of herbenoemen, enzovoort.

8.2 Een RAM-disk aanmaken

Na het opstarten van de computer (in BASIC) is er nog geen RAM-disk 'aanwezig'. Als u dat echter zou willen, zou u dat als eerste aktie kunnen doen. Het is echter niet nodig om de RAM-disk direkt na het opstarten aan te maken. U kunt dat doen op ieder moment dat u dat wilt. Wel zult u ervoor moeten zorgen dat de RAM-disk aangemaakt is, voordat u probeert er iets naar toe te schrijven of vanaf te lezen. Het aanmaken van de RAM-disk kunt u met het volgende statement doen:

```
CALL MEMINI [(<aantal bytes>)]
```

Met aantal bytes kunt u aangeven hoe groot de RAM-disk moet worden. Er zijn de volgende mogelijkheden:

1. Aantal bytes wordt weggelaten
2. Aantal bytes is een door u te kiezen waarde
3. Aantal bytes is 0

Wanneer u het aantal bytes helemaal niet opgeeft, u geeft dus alleen het kommando CALL MEMINI, dan zal de RAM-disk automatisch zo groot mogelijk worden gemaakt. Dit is de meest voor de hand liggende manier van werken. Ik zou u ook willen aanraden om de RAM-disk altijd op deze manier aan te maken.

U mag, indien u dat wenst, zelf een waarde voor aantal bytes kiezen. Daarbij dient u er rekening mee te houden, dat de werkelijke waarde wel eens anders kan zijn dan de door u opgegeven waarde. De RAM-disk zal namelijk altijd een grootte krijgen die een veelvoud is van blokjes van 256 bytes. Bovendien zal de vrije RAM-disk ruimte drie blokjes van 256 bytes kleiner zijn dan de door u opgegeven ruimte. Net als op een normale schijf, moet ook op de RAM-disk een inhoudsopgave worden bijgehouden. De inhoudsopgave van de RAM-disk is korter dan die van een normale schijf. Hierdoor kunnen er maximaal 32 bestanden en programma's op de RAM-disk worden geschreven.

Afhankelijk van het type MSX-computer dat u heeft, kan de minimum toegestande grootte van de RAM-disk verschillen. Daarover later meer.

De derde mogelijkheid is, de waarde 0 voor aantal bytes op te geven. Hiermee wordt de RAM-disk uit het systeem verwijderd. U heeft dan dus geen RAM-disk meer.

Het volgende programma vraagt u hoe groot de RAM-disk moet worden en maakt vervolgens de RAM-disk aan volgens de door u gewenste grootte. U kunt dit programma goed gebruiken om te bepalen wat de minimale grootte van de RAM-disk op uw computer is.

```
10 '*****************************
20 '* Kreeer RAM-disk (dev=MEM) *
30 '*****************************
40 '
50 CLS
60 INPUT "Grootte van de RAM-disk (=<32 kBytes)";G
70 PRINT "Afgerond op een blok van 256 bytes...";
80 CALL MEMINI(G+767)
```

Als u een te kleine waarde opgeeft, zult u de boodschap *"No RAM disk"* krijgen. Geeft u een te grote waarde op, dan krijgt u de boodschap *"Overflow"*. Geeft u een toegestane waarde op, dan krijgt u de boodschap *"xxxx bytes allocated"*. Hierin is *xxxx* de werkelijke grootte van de aangemaakte RAM-disk.

Waarschuwing:
Wanneer u een RAM-disk heeft aangemaakt en er bestanden naartoe heeft geschreven, dan mag u de grootte van die RAM-disk niet meer wijzigen. Iedere keer dat het MEMINI-statement wordt uitgevoerd, wordt namelijk de RAM-disk opnieuw aangemaakt. Dit houdt in, dat alle gegevens die op de oude RAM-disk stonden, verdwenen zullen zijn op het moment dat er een nieuwe RAM-disk wordt aangemaakt.

8.3 Gebruik van de RAM-disk

Hebben we de RAM-disk eenmaal aangemaakt, dan kunnen we deze gaan gebruiken. Daartoe zijn een aantal nieuwe statements in MSX2 gedefinieerd en zijn een aantal bestaande MSX1-statements uitgebreid.

De nieuwe statements zijn:

CALL MFILES
CALL MKILL
CALL MNAME

De voor MSX2 uitgebreide MSX1-statements zijn:

SAVE, LOAD, MERGE en RUN
OPEN

De uitbreiding van deze statements ten opzichte van MSX1 is, dat er een nieuw randapparaat aan de mogelijkheden is toegevoegd, namelijk het randapparaat MEM. In hoofdstuk 6 werden deze uitbreidingen al uitgebreid beschreven. We zullen er daarom hier vanuit gaan dat deze uitgebreide statements bekend zijn.

Wanneer u met SAVE een programma naar de RAM-disk heeft geschreven, dan kunt u dat programma met LOAD of RUN weer teruglezen. Had u dat programma als ASCII-file weggeschreven, dan kunt u het bovendien met MERGE vanaf de RAM-disk koppelen met een op dat moment in het geheugen staand programma.

Nadat u met het OPEN-statement een bestand op de RAM-disk heeft geopend, kunt u er met de statements PRINT en PRINT USING gegevens naar toe schrijven. Met de statements INPUT, LINE INPUT en INPUT$ kunt u gegevens uit het RAM-disk bestand lezen. Al deze PRINT en INPUT statements moeten worden gebruikt volgens het bestandsformaat. Dat wil zeggen, dat er aan de statements een file-nummer moet worden toegevoegd. Dit is precies hetzelfde als wanneer een normaal schijvenbestand wordt gelezen of beschreven.

Om nu de nieuwe RAM-disk statements te kunnen oefenen, dient u een kort programma in te tikken. Dit programma mag bijvoorbeeld geheel uit REM-regels bestaan. Om dit programma nu op RAM-disk te zetten, geeft u het kommando:

```
SAVE "MEM:TEST"
```

U hoort niets, doch na zeer korte tijd, veel korter dan u van een schijf gewend bent, verschijnt de Ok-prompt al weer op het scherm. Dit houdt in, dat het programma nu op de RAM-disk staat. Dit kunnen we kontroleren door een inhoudsopgave van de RAM-disk te vragen. Dat doen we als volgt:

```
CALL MFILES
```

U krijgt nu een opgave van alle bestanden die op de RAM-disk staan, gevolgd door de nog beschikbare ruimte. U ziet dat dit statement precies zo werkt als het gewone FILES-statement.

Nu zullen we het bestand op de RAM-disk gaan herbenoemen, ofwel een andere naam gaan geven. Ook hiertoe is een nieuw MSX2-statement beschikbaar. Stel dat we het bestand TEST1 willen gaan noemen. We geven dan het kommando:

```
CALL MNAME ("TEST" AS "TEST1")
```

Merk hierbij op, dat het formaat enigszins verschilt van het NAME-statement. De bestandsnamen staan nu tussen haakjes en er mogen geen randapparaatnamen aan de bestandsnamen worden toegevoegd. Voor het overige doet dit statement precies hetzelfde als het oorspronkelijke NAME statement. Dat het bestand TEST nu TEST1 heet kunt u kontroleren door nog eens "CALL MFILES" te geven.

Het laatste nieuwe MSX2-statement voor gebruik van de RAM-disk is CALL MKILL. Om het bestand TEST1 van de RAM-disk te verwijderen, dienen we het volgende kommando te geven:

```
CALL MKILL ("TEST1")
```

Ook hier geldt weer, dat de bestandsnaam tussen haakjes moet staan en dat er geen randapparaat-naam aan de bestandsnaam mag worden toegevoegd. Na uitvoering van dit kommando is het bestand TEST1 van de RAM-disk verwijderd.

Wanneer u de computer zou uitzetten, verliest het RAM-geheugen zijn inhoud. Indien u in het RAM-geheugen een RAM-disk heeft gedefinieerd en u heeft op die RAM-disk een aantal programma's en bestanden staan, dan kunt u de computer pas uitzetten nadat u de

programma's en bestanden van de RAM-disk naar een floppy disk heeft gekopieerd. Het zal u echter zijn opgevallen, dat het COPY-statement niet is uitgebreid met het randapparaat MEM. We kunnen de RAM-disk dus niet kopiëren met het COPY-statement.

Als er dan geen kommando is, waarmee bestanden van RAM-disk naar floppy disk kunnen worden gekopieerd, dan moeten we daar maar een programma voor schrijven. Het volgende programma kopieert bestanden van RAM-disk naar floppy disk. Tikt u het eerst in, en schrijf het daarna naar floppy disk.

```
100 '**************************
110 '* Kopieren van files van *
120 '*   RAM-disk naar disk   *
130 '**************************
140 '
150 CLS: CLEAR 500: MAXFILES=2
160 PRINT "De volgende bestanden staan op RAM-disk:"
170 PRINT
180 CALL MFILES
190 PRINT
200 INPUT "Welk bestand wilt u kopieren";B$
210 OPEN "MEM:"+B$ FOR INPUT AS #1
220 OPEN "A:"+B$ FOR OUTPUT AS #2
230 IF EOF(1) THEN END
240 LINE INPUT #1,I$
250 PRINT #2,I$
260 GOTO 230
```

Dit programma is niet alleen nuttig in het gebruik; het laat bovendien een aantal van de hiervoor vermelde mogelijkheden zien. Zo ziet u, dat in regel 210 een bestand op RAM-disk wordt geopend. Met regel 240 wordt een record uit het RAM-disk bestand gelezen met behulp van een LINE INPUT statement. Het gelezen record wordt naar het in regel 220 op schijf A: geopende bestand geschreven met het PRINT statement van regel 250. Met regel 250 wordt gekontroleerd of het einde van het bestand op de RAM-disk al is bereikt.

Omdat er twee bestanden worden geopend (één op RAM-disk en één op floppy disk), wordt in regel 150 het maximum aantal bestanden dat tegelijkertijd geopend mag zijn op 2 gesteld. Omdat het string-geheugen standaard 200 bytes groot is, zou het kunnen gebeuren dat er

een record wordt gelezen dat te groot is om in het string-geheugen te passen. Vandaar dat de grootte van het string-geheugen in regel 150 op 500 bytes wordt ingesteld.

Met dit programma kunt u alle bestanden, die u op RAM-disk kunt opslaan, kopiëren naar een floppy-disk. Wij hadden eerlijk gezegd aanvankelijk onze twijfels over de mogelijkheid om BASIC-programma's, die niet als ASCII-bestanden waren weggeschreven, te kunnen kopiëren. In de praktijk is echter gebleken, dat ook BASIC-programma's met dit kopieerprogramma van RAM-disk naar floppy-disk kunnen worden overgebracht.

HOOFDSTUK 9

SCHIJVEN: DE PUNTJES OP DE I

In deel 3 uit deze leerboekenserie heeft u al veel kunnen leren over het gebruik van de schijf, zowel vanuit MSX-BASIC als vanuit MSX-DOS. In dat boek hebben we ons echter beperkt tot de gebruikersaspekten. We hebben gezien, dat het bijhouden van de bestandsadministratie een gekompliceerde zaak is en dat we blij mogen zijn dat het systeem dat allemaal voor ons doet. Hierdoor kunnen wij ons beperken tot het gebruik van de ons ter beschikking staande statements.

Een gevolg daarvan is, dat we nog weinig weten over wat het systeem nu precies allemaal voor ons doet. Ook weten we daardoor weinig van de organisatie van schijven en kunnen we vragen zoals:

- Waar staat welk bestand?
- Wat mag wel of niet met dat bestand worden gedaan?
- Is er nog ruimte op de schijf?
- Waar zit die ruimte?
- Kan ik schijven van andere computers lezen?

niet beantwoorden.

In dit hoofdstuk zullen we, zoals de titel al aanduidt, dieper ingaan op de organisatie van schijven. Voor sommige mensen zal dit zelfs te diep gaan. Indien u de schijf alleen maar wilt gebruiken, zoals in deel 3 van de leerboekenserie werd getoond, dan zult u de kennis uit dit hoofdstuk niet nodig hebben. Wat u waarschijnlijk wel zult willen hebben, zijn een aantal van de programma's uit dit hoofdstuk. Daarbij valt vooral te denken aan de programma's die een soort uitgebreide directory geven. Met deze programma's kunt u namelijk onder MSX-BASIC directories maken waar zelfs nog meer gegevens over de bestanden in staan, dan in de MSX-DOS directory.

Voor al diegenen, die wel eens te maken hebben met schijfjes die op een computer met een ander formaat zijn aangemaakt, is vooral het

theoretische deel van dit hoofdstuk van belang. Stel dat u een enkelzijdige schijveneenheid heeft en een vriend of kollega heeft een dubbelzijdige schijveneenheid. Kunt u dan probleemloos schijven van uw kollega in uw eenheid lezen en omgekeerd? Het antwoord op die vraag en nog vele andere zult u in dit hoofdstuk vinden.

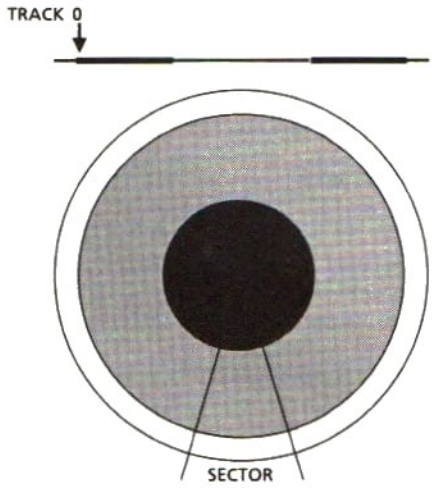
Daarnaast zullen in dit hoofdstuk een nog niet eerder behandelde BASIC-funktie en een BASIC-statement worden besproken. Het betreft hier de funktie DSKI$ en het statement DSKO$. Verder zult u een nieuwe toepassing zien van de funktie VARPTR, die al in deel 2 van deze leerboeken serie werd behandeld.

9.1 De fysieke opbouw van de schijf

De schijf bestaat uit een rond schijfje kunststof, waarop aan beide zijden een magnetiseerbaar laagje is gespoten. Door nu de schijf rond te draaien en een elektromagneetje tegen de onder- of bovenzijde van de schijf te houden, kunnen er door dat elektromagneetje nulletjes en eentjes op de schijf worden geschreven. Wanneer dat magneetje netjes stilgehouden wordt, zal het duidelijk zijn, dat die nulletjes en eentjes in cirkelvorm op het schijfje worden geschreven. We noemen zo'n cirkel een *"track"*, of in Nederlands, *"spoor"*.

Door nu het elektromagneetje dichter naar het middelpunt van de schijf te bewegen, of juist verder daarvandaan, zal het magneetje van het zojuist geschreven spoor af gaan en op een ander spoor terecht komen. Het elektromagneetje waar we het nu over hebben wordt in een schijveneenheid de "lees/schrijf-kop" genoemd. Wanneer die kop boven een eerder beschreven spoor wordt gehouden, dan zal het magnetisme van dat spoor een heel klein stroompje opwekken in het spoeltje in de leeskop. Na versterking van dit signaal kan dat weer worden omgezet naar nulletjes en eentjes.

Van het totale oppervlak van de schijf wordt slechts een klein deel gebruikt om er sporen op te schrijven. Het buitenste deel van de schijf wordt niet gebruikt, omdat daar mogelijk mechanische vervormingen aan het schijfje optreden. Het binnenste deel wordt niet gebruikt, omdat daar de snelheid (in millimeters per sekonde) te laag wordt, waardoor de 'bitdichtheid' te hoog zou worden. Op het overblijvende deel (in *afbeelding 9-1* het grijze gebied) kunnen sporen worden geschreven.



Afb. 9-1 De fysieke indeling van een schijf

Nu is er slechts één soort schijf, namelijk een schijf, die aan beide zijden met een te magnetiseren materiaal is bespoten. Er zijn echter verschillende soorten schijveneenheden. Er zijn schijveneenheden met slechts 1 lees/schrijfkop, maar er zijn ook eenheden met twee lees/schrijfkoppen. Indien een schijveneenheid slechts één lees/schrijfkop heeft, dan kan die eenheid slechts één kant van de schijf gebruiken. Zou je in zo'n eenheid een schijfje stoppen dat is aangemaakt op een eenheid met 2 koppen, dan kun je slechts de helft lezen van wat er op die schijf staat. Wat je dan leest en wat de gevolgen zijn, zullen we later zien.

Omdat er schijveneenheden zijn met slechts één lees/schrijfkop, hebben schijven-fabrikanten een markt gevonden voor schijfjes die aan één kant een onbetrouwbare laag bleken te hebben. U moet zich voorstellen, dat de schijfjes worden gesneden uit grote vellen kunststof. Wanneer nu tijdens de kwaliteitskontrole van zo'n vel blijkt, dat er aan éénn zijde van dat vel, een slecht magnetiseerbare laag zit, dan worden de schijfjes uit dat vel niet weggegooid, doch ze worden als enkelzijdige schijfjes verkocht. Meestal zijn deze schijfjes veel goedkoper dan dubbelzijdige schijfjes, terwijl ze voor het doel waarvoor ze zijn

gemaakt, uitstekend voldoen. Deze schijfjes mogen echter nooit in een dubbelzijdige schijveneenheid worden gebruikt, omdat dan de kans op fouten in weggeschreven bestanden zeer groot is.

Een nieuwe schijf is nog nooit gemagnetiseerd. Er staan dus nog geen sporen op. Daarom moet een nieuwe schijf, voordat deze kan worden gebruikt, worden ge-*formateerd*. Dit wil dus niets anders zeggen, dan dat er op die schijf sporen moeten worden geschreven.

Wanneer we te maken hebben met een enkelzijdige schijveneenheid, dan zullen alle sporen aan dezelfde zijde van de schijf worden geschreven. Het eerste spoor krijgt daarbij het nummer 0. Dit is aangegeven in *afbeelding 9-1*, bovenin de tekening. Het volgende spoor, spoor nummer 1, zal dan naast spoor 0 liggen, op dezelfde zijde van de schijf, net iets dichter naar het midden van de schijf. Het laatste spoor ligt het meest naar binnen.

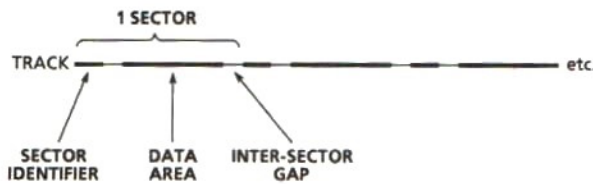
Hebben we echter te maken met een dubbelzijdige schijveneenheid, dan liggen de sporen aan beide zijden van de schijf. Het eerste spoor krijgt weer spoor nummer 0 en ligt op precies dezelfde plaats als spoor 0 van een enkelzijdige schijf. Nu komt het echter. Het tweede spoor, spoor nummer 1 ligt aan de onderkant van de schijf, bijna recht tegenover spoor nummer 0. Het derde spoor, spoor nummer 2 ligt weer aan de bovenkant, vlak naast spoor nummer 0. Het vierde spoor ligt weer onder, enz.

U kunt zich misschien voorstellen wat er gebeurt, wanneer een bestand zo groot is, dat het over meerdere sporen staat opgeslagen. Stel dat het een dubbelzijdig schijfje is en we proberen dat bestand te lezen in een enkelzijdige schijveneenheid. Een deel van het bestand zit aan de bovenkant en kan worden gelezen, doch een ander deel zit aan de onderkant van de schijf en kan niet worden gelezen. Wanneer het systeem hier geen voorzieningen tegen zou treffen, zouden we kunnen denken, dat we het hele bestand hebben gelezen, terwijl we in werkelijkheid slechts een deel daarvan hebben gelezen. Gelukkig kan het systeem ons voor dit soort fouten behoeden. We zullen dat straks gaan zien.

Nu hebben we nog niet alle mogelijke verschillen gehad. Sommige schijveneenheden hebben een dermate nauwkeurige positionering van de lees/schrijf-kop en kunnen zo'n dun spoortje schrijven, dat ze in

staat zijn op het beschrijfbare deel van de schijf 80 sporen te schrijven, terwijl andere schijveneenheden er slechts 40 sporen in kwijt kunnen. Het zal duidelijk zijn dat schijven, die met sporen zijn beschreven op een 80-sporen schijveneenheid, niet kunnen worden gebruikt op een eenheid die slechts voor 40 sporen geschikt is.

We weten nu, dat een schijf is onderverdeeld in een aantal cirkelvormige sporen. Met de huidige stand der techniek is het mogelijk op zo'n spoor vele duizenden tekens te schrijven. Dit is nogal veel tegelijk. Vandaar, dat de sporen zijn opgedeeld in hapklare brokken, die *sektoren* worden genoemd. In *afbeelding 9-1* is te zien wat onder een sektor wordt verstaan. Daarin is ook duidelijk te zien, dat een dicht bij het middelpunt van de schijf gelegen sektor veel korter is (in millimeters) dan een sektor verder uit het middelpunt. Wil je toch een vast aantal bytes in iedere sektor kwijt, ongeacht de plaats van die sektor, dan zal het duidelijk zijn dat de bitdichtheid in sektors dichtbij het middelpunt veel groter is dan de bitdichtheid van de sektors die verder uit het middelpunt liggen.



Afb. 9-2 De opbouw van sektoren op een track

Op MSX-schijfjes zijn alle sporen onderverdeeld in 9 sektoren. Wanneer we nu een track van de schijf nemen en rechtuit leggen, dan krijgen we de situatie uit *afbeelding 9-2*. Daarin zien we, dat een sektor uit vier deeltjes bestaat:

- sektor identifier (sektor nummer)
- korte onderbreking
- data area
- inter-sektor gap

Dit komt dan op elk spoor negen keer voor. De sektor identifier is het volgnummer van de sektor. Wanneer we een bepaalde sektor willen lezen of beschrijven, dan zal het systeem eerst de kop moeten

positioneren boven het spoor, waarop die sektor zich behoort te bevinden en vervolgens moet net zo lang worden gelezen, tot het gezochte sektor nummer is gevonden. Nu is er een hele korte tijd, waarin het systeem kan beslissen of het wil omschakelen van lezen naar schrijven of niet. Dat omschakelen gaat weliswaar elektronisch, doch het kost wel tijd. De korte onderbreking tussen de sektor identifier en de data area verschaft het systeem de benodigde tijd.

De data area is voor alle MSX-schijven groot genoeg om er 512 bytes informatie in te zetten. Willen we, na het beschrijven van een sektor, nog een andere sektor lezen of beschrijven, dan zullen we weer moeten omschakelen van schrijven naar lezen. Ook dit kost weer tijd, die ons geboden wordt door de *intersektor gap* tussen twee sektoren.

We weten nu, hoe een schijf fysiek is ingedeeld in sporen (tracks) en sektoren. Bovendien is meegedeeld hoeveel sporen en sektoren en hoevel bytes iedere sektor van een MSX-schijfje bevat. MSX is echter een produkt van Microsoft en dit bedrijf heeft ook de systeemsoftware voor Personal Computers ontworpen. Het is dan ook niet verwonderlijk, dat de indeling van MSX-schijven komt uit de algemene indeling van schijven, zoals Microsoft die heeft gedefinieerd. Volgens die algemene indeling zijn er meer mogelijkheden dan er in MSX worden gebruikt. Wanneer de afmetingen van de schijfjes gelijk zijn, kunnen alle soorten schijven worden gelezen. Het leuke hiervan is, dat MSX-schijven ook in Personal Computers kunnen worden ingelezen, immers deze computers kennen dezelfde onderverdelingen van schijven.

9.2 De logische indeling van de schijf

Nu zijn we dan zover, dat we de schijf in stukjes van 512 bytes hebben onderverdeeld. Ieder stukje heeft zijn eigen nummer, en is zodoende individueel te adresseren. We kunnen dus beginnen, gegevens naar de schijf te schrijven. Stel, we willen er een programma opzetten. We weten, dat de schijf nog leeg is, dus we kunnen beginnen met de eerste sektor. We moeten nu wel onthouden welke sektoren allemaal door dat programma in beslag worden genomen, want anders lopen we de kans dat het volgende bestand dat we op de schijf gaan zetten, het vorige overschrijft. We moeten dus ergens een tabel gaan aanleggen, waarin staat welke sektoren gebruikt worden voor het opslaan van ieder bestand. Die tabel kunnen we niet in het geheugen bewaren, want als we

de computer hebben uitgezet, is de tabel ook verdwenen. Dus laten we die tabel maar op de schijf zetten. De volgende keer dat we die schijf dan weer lezen, lezen we er eerst de tabel van af. Echter, om dat te kunnen, moeten we die tabel wel op een vaste plaats zetten.

Op het eerste gezicht mag het gebruik van de schijf eenvoudig lijken, maar wanneer je er wat langer over nadenkt en nagaat wat er allemaal moet gebeuren en wat er allemaal fout kan gaan, zoals in de vorige alinea, dan kom je er toch wel achter dat er nogal wat bij komt kijken. Denk je maar eens in, dat er een aantal bestanden achter elkaar op aaneengesloten sektoren staan en dat een bestand dat ergens tussen de andere bestanden staat, wat langer gaat worden. Hoe los je dat op? De uitbreiding van dat bestand achter het laatste bestand zetten, of het gehele uitgebreide bestand achteraan zetten? En wat doe je in dat laatste geval met de nu niet meer gebruikte tussenliggende sektoren?

Gelukkig hoeven wij ons met al dat soort zaken niet bezig te houden. Dat doet het disk-BASIC voor ons. Toch is het wel interessant om eens te zien hoe disk-BASIC dat doet.

In grote lijnen wordt een schijf daartoe onderverdeeld in vier gebieden, die in de hierna genoemde volgorde vanaf sektor 0 op de schijf staan:

- een boot-sektor
- een File Allocation Tabel
- een directory (inhoudsopgave)
- het data gebied

De boot-sektor is de eerste sektor op de schijf (sektor nummer 0). Deze sektor wordt door het formateringsprogramma gevuld met gegevens over de soort schijf. Zo vindt u daarin het aantal sektoren op de schijf, het aantal sporen, het aantal sektoren per track, enzovoort. Wat er precies in staat zullen we straks zien.

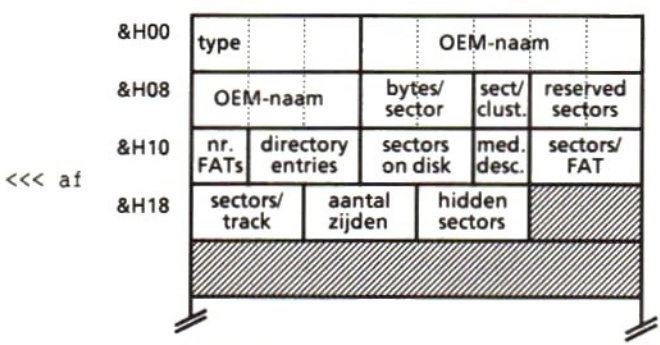
De File Allocation Tabel bevat voor ieder kluster (sektors worden meestal samengevoegd tot zogenaamde "klusters") een aanduiding, waaruit blijkt of de betreffende kluster nog vrij is of al wordt gebruikt voor het opslaan van een bestand. Afhankelijk van het totaal aantal sektoren op de schijf kan deze File Allocation Tabel (meestal aangeduid met FAT) groter of kleiner zijn. Op floppy disks neemt deze tabel meestal 4 tot 6 sektoren in beslag.

De directory bevat voor ieder bestand een entry, waarin behalve de naam van dat bestand ook gegevens over de grootte en de aanmaaktijd en -datum staan. Wie wel eens met MSX-DOS heeft gewerkt, weet dat het DIR-kommando al dit soort gegevens laat zien. Ook wij zullen straks deze gegevens zien en ze door een programma laten afdrukken. Afhankelijk van het type (de kapaciteit) van de schijf zal de directory 2 of meer sektoren in beslag nemen.

Het data gebied bevat de eigenlijke bestanden. Alle sektoren, die niet tot de boot-sektor, de FAT of de directory behoren, worden geacht tot het data gebied te behoren en kunnen worden gebruikt voor het opslaan van bestanden.

De boot-sektor

De boot-sektor bevat belangrijke informatie over het type schijf. Hierdoor is het voor een computersysteem mogelijk om te bepalen of het een bepaalde schijf wel of niet kan lezen. De meeste computers zijn in staat om meerdere verschillende soorten schijven te lezen, zo ook de MSX-computer. *Afbeelding 9-3* laat zien, wat er in die boot-sektor staat. De afbeelding is zo opgezet, dat er steeds rijen van 8 bytes in zijn afgebeeld. Het eerste (meest linkse) byte op de tweede rij is dus het negende byte.



Afb. 9-3 De indeling van het eerste deel van de boot-sektor

Type:
De eerste drie bytes bevatten een machinetaal sprong-instruktie, die in MS-DOS computers wordt gebruikt. De operand van die instruktie staat in het eerste byte (byte nummer 0) van de boot- sektor en kan hexadecimaal EB of E9 zijn. Wanneer het eerste byte van de boot-sektor één van deze twee waarden bevat, dan weet men dat de betreffende schijf een MS-DOS schijf is. En aangezien het formaat van MSX gelijk is aan MS-DOS, weet men dus ook dat de schijf door een MSX-computer kan worden gelezen.

OEM-naam:
De lengte van dit veld is 8 bytes. De fabrikant van de disk-controller en de schijveneenheid zet hier zijn eigen aanduiding in. Schijven die ik heb geformateerd met mijn Philips VG8235 computer bevatten de naam "NMS 2.2", terwijl schijven die ik met mijn AVT-drive heb geformateerd de aanduiding "DWDPF510" bevatten.

Bytes/sektor:
Dit twee bytes veld bevat een aanduiding van het aantal bytes dat iedere sektor groot is. Voor alle MSX-floppies geldt dat iedere sektor 512 bytes bevat.

Sect/kluster:
In dit veld van 1 byte vindt u het aantal sektoren waaruit een kluster bestaat. Bij het lezen van een schijf wordt altijd een kluster gelezen. Wanneer er twee sektoren tot een kluster zijn samengevoegd, zijn er dus twee sektoren gelezen wanneer er 1 kluster is gelezen.

Reserved sectors:
Het aantal sektoren dat is gereserveerd. Deze sektoren bevinden zich tussen de laatste sektor van de directory en kluster nummer 1.

Nr.FAT's:
Dit byte bevat het aantal FAT's. Meestal wordt er van de FAT een kopie bijgehouden, zodat er twee FAT's zijn.

Directory entries:
In deze twee bytes staat aangegeven hoeveel files er in de directory kunnen worden opgenomen. Iedere directory entry is 32 bytes. Hiermee is dus vrij eenvoudig uit te rekenen, hoeveel sektoren er nodig zijn voor de directory.

Sektors on disk:
Deze twee bytes bevatten het totaal aantal sektoren dat op de disk staat. Bij negen sektoren per track en 80 tracks zal er in dit veld dus 9* 80=720 staan.

Med. Desc.:
Dit byte bevat de Medium Descriptor. Dit kan een waarde van hexadecimaal F8 tot en met FF zijn. Deze waarde is een verwijzing naar een rij in een tabel, waarin dezelfde gegevens zijn opgenomen, die ook in de andere velden van de boot-sektor zijn te vinden. De medium descriptor wordt op sommige computersystemen gebruikt, in plaats van het lezen van alle afzonderlijke velden.

Sektors/FAT:
Dit twee bytes veld geeft aan uit hoeveel sektoren iedere FAT bestaat. Wanneer er in dit veld 2 staat, en in het veld Nr.FAT's staat ook twee, dan is de totale lengte van de FAT's 2*2=4 sektoren.

Sektors/track:
In dit tweebytes veld staat hoeveel sektoren er op ieder spoor staan. Dit kan, afhankelijk van het type schijf 8 of 9 zijn. Voor niet-MSX schijven zouden dit nog andere waarden kunnen zijn.

Aantal zijden:
Dit tweebytes veld geeft het aantal zijden van de schijf aan. Voor MSX-schijven kan dit 1 of 2 zijn. Harde schijven hebben meestal meerdere zijden, doch die worden nog niet aan MSX-computers gebruikt.

Hidden sektors:
Dit veld van twee bytes bevat het aantal verborgen sektoren. Onder MSX wordt dit niet gebruikt. In sommige computersystemen kunnen sektoren echter voor de gebruiker worden verborgen, terwijl het systeem die sektoren wel kan gebruiken.

Na deze velden volgt nog een machinetaalroutine, de zogenaamde boot-routine. De rest van deze sektor is leeg.

U zult nu wel nieuwsgierig zijn geworden naar de inhoud van de bootsektoren op uw schijfjes. Welnu, die nieuwsgierigheid hoeft u niet zo lang meer op de proef te stellen. We zullen een programma maken,

waarmee de boot-sektor kan worden gelezen en op een voor ons, mensen, duidelijke manier kan worden afgedrukt. Voordat we de listing daarvan geven, moeten we echter eerst een nog niet eerder behandelde funktie bespreken, namelijk de funktie DSKI$. Het formaat van deze funktie is:

```
DSKI$(<drive>,<sektor nr.>)
```

Dit is een funktie en zoals bekend, kan een funktie nooit zelfstandig worden geprogrammeerd. We zouden kunnen programmeren:

```
I$=DSKI$(1,0)
```

Dit heeft tot gevolg, dat sektor nummer 0 van disk drive nummer 1 (komt overeen met drive A:) wordt gelezen. Echter de gelezen inhoud van de sektor wordt niet, zoals u misschien zou denken, in variabele I$ terecht. Dit kan ook niet, want een sektor is 512 bytes, terwijl de maximale lengte van een variabele 255 bytes is. Waar komt die informatie dan wel terecht?

In het systeemgeheugen staat een veld, waarin een adres staat. Dat adres verwijst naar het eerste byte van het geheugengebied, waar de gelezen sektor is opgeslagen. Het bedoelde veld in het systeemgeheugen is het tweebytes veld op de adressen hexadecimaal F351 en F352. Wanneer we dit veld met behulp van PEEK-funkties uitlezen, weten we op welk adres de sektor-informatie in het geheugen kunnen vinden.

In het volgende programma wordt op regel 200 de boot-sektor gelezen. Met regel 210 wordt vervolgens bepaald, waar de gelezen informatie zich in het geheugen van de computer bevindt. Met de daaropvolgende regels 220 tot en met 380 worden de gegevens uit de boot-sektor leesbaar afgedrukt. Een aardigheidje in dit programma is, dat het u vraagt of de informatie op het beeldscherm of op de printer moet worden afgedrukt. Afhankelijk van uw antwoord wordt met regel 180 randapparaat CRT of LPT geopend als bestandsnummer 1. In de PRINT-statements ziet u, dat er altijd naar bestandsnummer 1 wordt geschreven. Of dat naar printer of beeldscherm is, ligt er aan of CRT of LPT was geopend.

```
100 '************************
110 '* De boot-sektor lezen *
120 '************************
130 '
140 CLEAR 1000: SCREEN 0: WIDTH 80: CLS
150 Z$="jJnN"
160 INPUT "Afdrukken op printer (j/n)";I$
170 IF INSTR(Z$,I$)=0 THEN GOTO 160
180 IF I$="j" OR I$="J" THEN OPEN "LPT:" AS #1 ELSE OPEN
"CRT:" AS #1
190 CLS
200 I$=DSKI$(1,0)
210 MA=PEEK(&HF351)+256*PEEK(&HF352)
220 FOR I=0 TO 7
230 N$=N$+CHR$(PEEK(MA+3+I))
240 NEXT I
250 PRINT #1,"Gegevens uit de boot-sektor:"
260 PRINT #1,
270 PRINT #1,"De naam van deze schijf is:              "
;N$
280 PRINT #1,"Het aantal bytes per sektor is:         ";
PEEK(MA+11)+256*PEEK(MA+12)
290 PRINT #1,"Het aantal sektors per kluster is:      ";
PEEK(MA+13)
300 PRINT #1,"Het aantal gereserveerde sektors is:    ";
PEEK(MA+14)+256*PEEK(MA+15)
310 PRINT #1,"Het aantal FAT's is:                    ";
PEEK(MA+16)
320 PRINT #1,"Het maximum aantal directory entries is:";
PEEK(MA+17)+256*PEEK(MA+18)
330 PRINT #1,"Het aantal sektoren is:                 ";
PEEK(MA+19)+256*PEEK(MA+20)
340 PRINT #1,"De medium-descriptor is:                "
;HEX$(PEEK(MA+21))
350 PRINT #1,"Het aantal sektoren per FAT is:         ";
PEEK(MA+22)+256*PEEK(MA+23)
360 PRINT #1,"Het aantal sektoren per track is:       ";
PEEK(MA+24)+256*PEEK(MA+25)
370 PRINT #1,"Het aantal zijden van de schijf is:     ";
PEEK(MA+26)+256*PEEK(MA+27)
380 PRINT #1,"Het aantal verborgen sektoren is:       ";
PEEK(MA+28)+256*PEEK(MA+29)
390 END
```

De "output" van dit programma ziet er als volgt uit. Deze voorbeelden zijn gemaakt van een 3,5 inch enkelzijdige Philips schijf en van een 5,25 inch enkelzijdige AVT schijf.

```
Gegevens uit de boot-sector:

De naam van deze schijf is:                NMS  2.2
Het aantal bytes per sector is:            512
Het aantal sectors per cluster is:         2
Het aantal gereserveerde sectors is:       1
Het aantal FAT's is:                       2
Het maximum aantal directory entries is:   112
Het aantal sectoren is:                    720
De medium-descriptor is:                   F8
Het aantal sectoren per FAT is:            2
Het aantal sectoren per track is:          9
Het aantal zijden van de schijf is:        1
Het aantal verborgen sectoren is:          0
```

voorbeeld 9-1

```
Gegevens uit de boot-sector:

De naam van deze schijf is:                DWDPF510
Het aantal bytes per sector is:            512
Het aantal sectors per cluster is:         1
Het aantal gereserveerde sectors is:       1
Het aantal FAT's is:                       2
Het maximum aantal directory entries is:   64
Het aantal sectoren is:                    360
De medium-descriptor is:                   FC
Het aantal sectoren per FAT is:            2
Het aantal sectoren per track is:          9
Het aantal zijden van de schijf is:        1
Het aantal verborgen sectoren is:          0
```

voorbeeld 9-2

Voordat we nu verder gaan met het behandelen van de directory, volgt hier nog een programma, waarmee u elke gewenste sektor van de schijf kunt lezen. De inhoud van die sektor wordt zowel hexadecimaal als in ASCII op het scherm afgedrukt. Daar de totale inhoud van de sektor op die manier niet in één keer op het scherm past, wordt eerst de ene helft afgedrukt, waarna op het indrukken van een willekeurige toets de tweede helft wordt afgedrukt.

Met dit programma kunt u alle bytes van elke willekeurige sektor afdrukken. Dit kan u straks en later nog goed van pas komen, wanneer u wat dieper wilt doordringen tot de geheimen van de schijf. We zullen het programma in ieder geval gaan gebruiken voor het onderzoeken van de FAT. Ook kunt u hiermee de directory sektoren heel goed

onderzoeken, al zullen we daarvoor nog een speciaal programma geven.

In de listing ziet u dat eerst de boot-sektor wordt gelezen, om te bepalen wat het maximum sektornummer is dat u mag intikken. Vanaf regel 210 wordt dan de door u gekozen sektor op het scherm afgedrukt. Dit programma bevat verder geen nieuwe koncepten, zodat we het niet verder zullen uitdiepen.

```
100 '*******************************
110 '* Een sektor van schijf lezen *
120 '*******************************
130 '
140 CLEAR 1000: SCREEN 0: WIDTH 80: CLS
145 '
146 '*******************************
147 '* Bepaal maximum sektornummer *
148 '*******************************
150 I$=DSKI$(1,0)
160 MA=PEEK(&HF351)+256*PEEK(&HF352)
170 SM=PEEK(MA+19)+256*PEEK(MA+20)-1
180 PRINT "Welke sektor wilt u lezen? (0 -";SM;")";
190 INPUT S
200 IF S>SM THEN GOTO 180
205 '
206 '*******************************
207 '*  Lees de opgegeven sektor.  *
208 '*******************************
210 CLS
220 I$=DSKI$(1,S)
230 MA=PEEK(&HF351)+256*PEEK(&HF352)
240 FOR I=0 TO 15
250 LOCATE 0,I: PRINT I*16;
260 FOR J=0 TO 15
270 LOCATE 10+J*3,I: PRINT HEX$(PEEK(MA+I*16+J));
280 LOCATE 64+J,I: PRINT CHR$(PEEK(MA+I*16+J));
290 NEXT J
300 NEXT I
310 LOCATE 0,20
320 PRINT "Druk op een willekeurige toets . . . ."
330 IF INKEY$="" THEN GOTO 330
340 CLS
```

```
360 LOCATE 0,I: PRINT I*16+256;
370 FOR J=0 TO 15
380 LOCATE 10+J*3,I: PRINT HEX$(PEEK(MA+256+I*16+J));
390 LOCATE 64+J,I: PRINT CHR$(PEEK(MA+256+I*16+J));
400 NEXT J
410 NEXT I
420 END
```

De directory

In de directory staat voor ieder bestand, dat op de schijf wordt opgeslagen, een entry. Iedere entry is 32 bytes lang. Zodoende passen er 512/32=16 entries in een sektor. Afhankelijk van het type schijf wordt een bepaald maximum aan het aantal directory entries gesteld. Dit maximum aantal entries is terug te vinden in de boot-sektor.

Wanneer we een bestand (dit kan ook een programma zijn) naar de schijf schrijven, dan zal dat bestand worden weggeschreven naar sektoren, die nog niet door andere bestanden bezet zijn. Gegevens over nog vrije sektoren haalt het systeem uit de FAT. In de directory wordt behalve de naam van het bestand ook aangegeven wat het eerste kluster is, waar het bestand op schijf is opgeslagen. Met behulp van de FAT kunnen we dan het volgende kluster vinden. Op de FAT komen we later nog terug.

Laten we eerst eens kijken, wat er allemaal over bestanden in de directory staat. Hierna volgt een overzicht van de inhoud van een directory entry. De gegevens staan in de gegeven volgorde in de directory entry. Van ieder gegeven wordt het eerste en het laatste byte opgegeven, als hexadecimale waarden, relatief ten opzichte van het begin van de entry. Voor de tweede entry dient u bij de gegeven verplaatsingen &H20 op te tellen, bij de derde &H40, enzovoort.

&H00 - &H07	file-naam
&H08 - &H0A	file-naam uitbreiding
&H0B - &H0B	file-attributen
&H0C - &H15	niet gebruikt
&H16 - &H17	aanmaak-tijd
&H18 - &H19	aanmaak-datum
&H1A - &H1B	eerste kluster van de file
&H1C - &H1F	file-grootte in bytes

Hierna volgt een korte omschrijving van deze velden, waarna we een programma zullen geven, dat al deze velden uitleest en op het scherm of op de printer afdrukt.

File-naam:
Dit is een veld van 8 bytes, waarin de naam van het bestand staat indien deze entry gebruikt wordt. Wanneer de entry nog nooit gebruikt is geweest, zal de eerste positie van dit veld de hexadecimale waarde 00 bevatten. Dit is meteen een aanduiding, dat het einde van de directory is bereikt. Wanneer deze entry wel een bestandsnaam heeft bevat, maar dat bestand is gewist van de schijf, dan zal de eerste positie van dit veld de hexadecimale waarde E5 bevatten. Een laatste mogelijkheid is, dat het eerste byte van deze entry de hexadecimale waarde 2E bevat. Is dat het geval, dan slaat deze entry op een sub-directory. MSX maakt geen gebruik van sub-directories, doch wanneer we een schijf van een Personal Computer in onze eenheid zouden leggen, dan is de mogelijkheid aanwezig, dat er op die schijf wel een sub-directory staat.

File-naam uitbreiding:
Dit 3-bytes veld bevat, indien van toepassing, de file-naam uitbreiding. Het zal u opvallen, dat de punt tussen de file-naam en de uitbreiding ontbreekt. Die punt wordt er door de software tussen gevoegd, doch staat niet op de schijf.

File-attributen:
Onder MSX is dit veld altijd hexadecimaal 00, hetgeen zoveel wil zeggen als: "Dit is een normaal bestand". Onder andere operating systemen kunnen er in dit veld echter een aantal interessante waarden staan. De mogelijke waarden zijn:

`&H00`	Normale file
`&H01`	Read-only file
`&H02`	Verborgen file (onzichtbaar in de directory)
`&H04`	Systeem file (onzichtbaar in de directory)
`&H08`	Volume label in eerste 11 bytes
`&H10`	Entry hoort bij een sub-directory
`&H20`	Archive-bit (file is gewijzigd)

Aanmaak-tijd:
In deze twee bytes staat de binaire waarde van de tijd in uren, minuten en sekonden, en wel als volgt:

```
&H16   M M M S S S S S
&H17   H H H H H M M M
```

HHHHH de binaire waarde voor de uren
MMMMMM de binaire waarde voor de sekonden
SSSSS het aantal keren 2 sekonden

In het programma, dat na de omschrijving van de velden van een directory entry wordt gegeven, komt een routine voor waarmee de aanmaak-tijd uit deze bytes wordt gedekodeerd.

Aanmaak-datum:
Dit veld van twee bytes wordt op dezelfde manier gebruikt als het veld aanmaak-tijd, alleen staan er nu binaire waarden in voor het jaartal, de maand en de dag van de maand, en wel als volgt:

```
&H18   M M M D D D D D
&H19   J J J J J J J M
```

JJJJJJJ de binaire waarde voor het jaartal (0=1980, 119=2099)
MMMM de binaire waarde voor de maand
DDDDD de binaire waarde voor de dag

Ook het uitkoderen van deze bitjes wordt in het volgende programma getoond.

Eerste kluster:
Dit is het kluster-nummer van het eerste kluster dat aan het bestand is toegekend. Wanneer een schijfje zojuist is geformateerd, dan zal het eerste bestand dat er op wordt geschreven, kluster-nummer 2 krijgen toegewezen als eerste kluster. Kluster-nummer 1 wordt dus nooit gebruikt. Later zullen we zien, hoe we een kluster-nummer kunnen vertalen naar een sektornummer.

File-grootte:
De file-grootte wordt opgegeven in bytes. Dit kan dus een tamelijk groot getal zijn, vandaar dat er voor het aangeven van de file-grootte vier bytes zijn gereserveerd. Bedenk daarbij ook, dat ditzelfde directory-formaat ook geldt voor harde schijven, waarop veel meer informatie kan dan op een floppy.

Alle tot nu toe besproken directory-gegevens worden met het volgende programma afgedrukt, naar keuze op het beeldscherm of op de printer. Met de regels 1140 tot en met 1260 wordt de plaats en de grootte van de directory bepaald. Daarna wordt de directory sektor voor sektor gelezen, totdat de laatste sektor is gelezen, of totdat het einde van de directory is bereikt.

Aan het kommentaar in de listing kunt u duidelijk zien wat elk deel van het programma doet. Let u vooral op de routines voor het dekoderen van de aanmaak-datum en -tijd.

```
1000 '*********************************
1010 '*       Directory lezen         *
1020 '*********************************
1030 '
1040 CLEAR 1000: SCREEN 0: WIDTH 80: CLS
1050 INPUT "Afdrukken op printer (j/n)";I$
1060 IF I$<>"j" AND I$<>"J" AND I$<>"n" AND I$<>"N" THE
N 1050
1070 IF I$="n" OR I$="N" THEN OPEN "CRT:" AS #1
1080 IF I$="j" OR I$="J" THEN OPEN "LPT:" AS #1
1090 CLS
1100 '
1110 '*********************************
1120 '*       Lees boot sektor        *
1130 '*********************************
1140 I$=DSKI$(1,0)
1150 MA=PEEK(&HF351)+256*PEEK(&HF352)
1160 'Aantal Fats:
1170 AF=PEEK(MA+16)
1180 'Sektors/Fat:
1190 SF=PEEK(MA+22)+256*PEEK(MA+23)
1200 'Eerste directory sektor:
1210 S1=AF*SF+1
1220 'Directory Entries:
1230 DE=PEEK(MA+17)+256*PEEK(MA+18)
1240 'Laatste directory sektor:
1250 SL=DE\32+S1-1
1260 IF DEMOD32<>0 THEN SL=SL+1
1270 '
1280 '*********************************
1290 '*   Lees een directory sektor   *
```

```
1300 '*********************************
1310 FT=0
1320 I$=DSKI$(1,S1)
1330 MA=PEEK(&HF351)+256*PEEK(&HF352)
1340 '
1350 '*********************************
1360 '*   Lees een directory entry    *
1370 '*********************************
1380 CLS
1390 PRINT #1,"File-naam:      Size:      Tijd:        Datu
m:   Kluster:  Type:"
1400 FOR I=0 TO 15
1410 '
1420 '*********************************
1430 '*  File-naam + File-naam ext.  *
1440 '*********************************
1450 N$="": U$=""
1460 IF PEEK(MA+I*32)=&H0 THEN GOTO 2090
1470 IF PEEK(MA+I*32)=&HE5 THEN PRINT #1,"Gewiste (nu du
s lege) entry": GOTO 2030
1480 FOR J=0 TO 7
1490 N$=N$+CHR$(PEEK(MA+I*32+J))
1500 NEXT J
1510 FOR J=8 TO 10
1520 U$=U$+CHR$(PEEK(MA+I*32+J))
1530 NEXT J
1540 PRINT #1,USING "\       \";N$;:PRINT #1,".";:PRINT #
1,USING"\ \";U$;
1550 '
1560 '*********************************
1570 '*    File grootte in bytes      *
1580 '*********************************
1590 FS=PEEK(MA+I*32+&H1C)+256*PEEK(MA+I*32+&H1D)+65536!
*PEEK(MA+I*32+&H1E)
1600 PRINT #1,USING "#########";FS;
1610 FT=FT+FS
1620 '
1630 '*********************************
1640 '*   Aanmaak-tijd van de file    *
1650 '*********************************
1660 T1=PEEK(MA+32*I+23)
1670 T2=PEEK(MA+32*I+22)
1680 HH=(T1 AND &B11111000)/8
```

```
1690 MM=((T1 AND &B00000111)*8)+((T2 AND &B11100000)/32)
1700 SS=(T2 AND &B00011111)*2
1710 PRINT #1,USING "######";HH;: PRINT #1,":";
1720 PRINT #1,USING "##";MM;: PRINT #1,":";
1730 PRINT #1,USING "##";SS;
1740 '
1750 '*********************************
1760 '*  Aanmaak-datum van de file    *
1770 '*********************************
1780 D1=PEEK(MA+32*I+25)
1790 D2=PEEK(MA+32*I+24)
1800 JJ=((D1 AND &B11111110)/2)+80
1810 MO=((D1 AND &B00000001)*2)+((D2 AND &B11100000)/32)
1820 DD=(D2 AND &B00011111)
1830 PRINT #1,USING "######";DD;: PRINT #1,"-";
1840 PRINT #1,USING "##";MO;: PRINT #1,"-";
1850 PRINT #1,USING "##";JJ;
1860 '
1870 '*********************************
1880 '*  Eerste kluster in de file    *
1890 '*********************************
1900 PRINT #1,USING "#######";PEEK(MA+32*I+&H1A)+256*PEE
K(MA+32*I+&H1B);
1910 '
1920 '*********************************
1930 '*       File Attributen         *
1940 '*********************************
1950 AT=PEEK(MA+32*I+&HB)
1960 IF AT=0 THEN PRINT #1,"    Normaal"
1970 IF AT=1 THEN PRINT #1,"    Read only"
1980 IF AT=2 THEN PRINT #1,"    Verborgen"
1990 IF AT=4 THEN PRINT #1,"    Systeem"
2000 IF AT=8 THEN PRINT #1,"    Volume label"
2010 IF AT=16 THEN PRINT #1,"    Sub-directory"
2020 IF AT=32 THEN PRINT #1,"    /Archived"
2030 NEXT I
2040 LOCATE 0,20: PRINT "druk op een willekeurige toets"
2050 I$=INKEY$: IF I$="" THEN GOTO 2050
2060 LOCATE 0,20: PRINT SPC(36)
2070 S1=S1+1
2080 IF S1=<SL THEN GOTO 1320
2090 PRINT #1,: PRINT #1,: PRINT #1,"einde van de direct
ory"
```

```
2100 PRINT #1,(S1-5)*16+I;"entries gebruiken samen";FT;"
bytes"
2110 END
```

De "output" van dit programma ziet er uit, zoals is weergegeven in de hiernavolgende afbeelding. Het betreft hier een directory van een 5,25 inch schijfje, dat vanuit de MSX-computer is beschreven met bestanden, die naderhand zijn gewijzigd op een Personal Computer. Van de gewijzigde bestanden was het attributen-byte op &H20 gezet door die PC. Het leek ons leuk om u juist deze directory te laten zien.

```
File-naam:      Size:      Tijd:        Datum:    Cluster:  Type:
H9PROG1 .         1172    13:41:22     14- 7-87        2    Normaal
H7PROG4 .         1280    11:57:18     12- 7-87        5    /Archived
H6PROG1 .         1024     1:30: 4      1- 1-80        9    /Archived
H6PROG2 .         2304     1:34:50      1- 1-80       11    /Archived
H7PROG2 .         1152     0: 5:36      1- 1-80       16    /Archived
H7PROG1 .          361    18:18:54     11- 7-87       17    Normaal
H7PROG5 .          256    14:54: 4     12- 7-87        3    /Archived
H7PROG3 .         2432     0:11:28      1- 1-80       23    /Archived
H9PROG2 .         1458    13:40:42     14- 7-87       19    Normaal
H8PROG1 .          384     0: 1:16      1- 1-80        8    /Archived
H8PROG2 .          512     0: 1:50      1- 1-80       18    /Archived
Gewiste (nu dus lege) entry
H9PROG3 .         3899    13:41:52     14- 7-87       33    Normaal

einde van de directory
 13 entries gebruiken samen 16234 bytes
```

voorbeeld 9-3

Waar staan de bestanden?

Tot nu toe hebben we alleen gekeken naar sektoren, waarin informatie over de op de schijf staande bestanden staat. Naar de bestanden zelf hebben we nog niet gekeken. Toch is dat buitengewoon nuttig. Wat is namelijk het geval?

Wanneer we een BASIC-programma naar schijf schrijven, of we schrijven een BASIC-programma als ASCII-bestand naar schijf, of wanneer we een machinetaalprogramma naar schijf schrijven, dan kunnen we in de directory niet meer zien wat voor soort bestand het is. Indien we een passende bestandsnaamuitbreiding hebben gebruikt, dan kunnen we aan de bestandsnaam zien, met welk soort bestand we naar alle waarschijnlijkheid te maken hebben. Helemaal zeker kunnen we er niet

van zijn; we zouden bij het wegschrijven een vergissing kunnen hebben gemaakt.

Het eerste byte van alle hiervoor genoemde bestanden geeft echter aan om welk soort bestand het gaat. Zo zal het eerste byte van de eerste sektor van een BASIC-programma de hexadecimale waarde FF hebben. Het eerste byte van een machinetaalprogramma heeft de hexadecimale waarde FE en een ASCII-bestand begint met een hexadecimale waarde die tussen 30 en 39 zal liggen.

Zouden we nu in staat zijn de eerste sektor van een bestand te vinden, dan zouden we naar het eerste byte in die sektor kunnen kijken, en afhankelijk van de waarde van dat byte een extra stukje informatie over dat bestand in de directory zetten.

Het wordt echter nog interessanter. Wanneer een machinetaalprogramma op schijf wordt gezet, dan wordt niet alleen in het eerste byte vermeld dat het bestand een machinetaalprogramma bevat, maar dan wordt dat byte gevolgd door drie twee-bytes adresvelden. Achtereenvolgens staan in die drie velden het beginadres, het eindadres en het startadres van de machinetaalroutine. U zult zich herinneren, dat het begin- en eindadres de RAM-adressen zijn, van waar af en tot waar aantoe het machinetaalprogramma in het geheugen zal worden geladen. Het startadres geeft aan op welk RAM-adres het machinetaalprogramma moet worden gestart.

Deze adressen heeft u bijvoorbeeld nodig, wanneer u een kopie naar kassette wilt maken van een machinetaalprogramma, dat op uw disk staat. Ook is het belangrijk deze adressen te kennen, om te kunnen bepalen of het machinetaalprogramma in het geheugen van uw computer kan worden geladen. Zoals gezegd, wanneer u de computer met aangesloten schijveneenheden opstart, wordt een deel van het geheugen voor de administratie van die schijven gebruikt. Uiteraard is die ruimte dan niet meer beschikbaar voor andere programma's. Wanneer nu blijkt dat een machinetaalprogramma te groot is om nog in het geheugen te passen, dan kunt u op zijn minst proberen de computer nog eens op te starten, nu echter terwijl u de Control-toets ingedrukt houdt. Op die manier wordt er slechts één schijveneenheid geaktiveerd en dat neemt minder ruimte in beslag, waardoor er dus meer ruimte overblijft voor andere programma's.

In het directory programma uit de vorige paragraaf hadden we het kluster nummer van het eerste kluster van ieder bestand opgezocht en afgedrukt. Het data-gebied van de schijf is opgedeeld in klusters. Het aantal sektoren dat wordt samengevoegd tot een kluster is computer-afhankelijk. Meestal zullen er 2 sektoren worden samengevoegd tot een kluster. Er zijn echter ook systemen waarin iedere sektor een kluster is. Hieronder volgt een voorbeeld van twee verschillende typen schijven. Aan de hand van dat voorbeeld zal het duidelijk zijn waar het data-gebied start:

5,25 inch AVT-drive (180 kBytes)			3,5 inch Philips drive (360 kBytes)		
Sek-tor:	**Klus-ter:**	**Omschrijving:**	**Sek-tor:**	**Klus-ter:**	**Omschrijving:**
0		boot-sektor	0		boot-sektor
1		FAT1/1	1		FAT1/1
2		FAT1/2	2		FAT1/2
3		FAT2/1	3		FAT2/1
4		FAT2/2	4		FAT2/2
5		directory	5		directory
6		directory	6		directory
7		gereserveerd	7		directory
8	1	data-gebied	8		directory
9	2	1e file-kluster	9		gereserveerd
10	3	2e file-kluster	10	1	data-gebied
11	4	3e file-kluster	11		data-gebied
12	5	4e file-kluster	12	2	1e file-kluster
13	6	5e file-kluster	13		-,,-

Uit dit voorbeeld kunt u opmaken dat het eerste kluster waarop een bestand kan kan staan, kluster nummer 2 is. Dat is zo op alle schijven. Kluster nummer 2 slaat echter niet altijd op dezelfde sektoren, zoals ook uit voorgaande tabel blijkt. In het ene geval is kluster nummer 2 gelijk aan 1 sektor, namelijk sektor nummer 9. In het andere geval is kluster nummer 2 gelijk aan twee sektoren, namelijk de sektoren nummers 12 en 13.

Hoe berekenen we nu het bij een bepaald kluster nummer behorende sektornummer? Om daarbij schijvenonafhankelijk te werk te kunnen gaan, zullen we zoveel mogelijk gebruik moeten maken van de infor-

matie die we uit de boot-sektor van een schijf kunnen halen. Op die manier kunnen we een formule samenstellen, die er als volgt uitziet:

boot sektor
aantal FATs keer aantal sektoren per FAT
aantal directory entries gedeeld door 32 (naar boven afronden)
aantal gereserveerde sektoren
aantal sektoren per kluster keer kluster-nummer
\+ ———————————————————————————
sub-totaal

Van dit subtotaal trekken we dan nog eenmaal het aantal sektoren per kluster af, waarmee we het sektornummer hebben gevonden dat bij een bepaald klusternummer hoort. In BASIC zouden we deze formule schrijven als:

```
SN=1+FATs*sFAT+dirsects+gereserv+sclust*clnr-sclust
```

Hierin is:

`FATs`	aantal FATs
`sFAT`	sektoren per FAT
`dirsects`	aantal directory sektoren
`gereserv`	aantal gereserveerde sektoren
`sclust`	sektoren per kluster
`clnr`	klusternummer

Vullen we voor de verschillende variabelen waarden in die we uit de boot-sektor hebben gelezen, dan gaat die formule er als volgt uitzien:

```
SN=1+(2*2)+4+1+(2*klusternummer)-2=
```

Alle ingevulde waarden gelden voor de 3,5 inch 360 kB enkelzijdige floppy. Door nu in deze formule een kluster nummer in te vullen is het bijbehorende sektor nummer eenvoudig te berekenen. Voor kluster nummer 2 wordt het sektor nummer 12. Kluster nummer 5 zal sektor nummer 18 opleveren, enzovoort.

In het volgende programma is de hiervoor behandelde kennis verwerkt. Nadat met de regels 1150 tot en met 1330 de boot sektor gegevens zijn ingelezen, wordt vanaf regel 1380 de directory gelezen. Steeds wanneer

een directory sektor in het geheugen is ingelezen, worden alle bestandsentries uit die directory sektor gelezen. Met de regels 1510 tot en met 1600 wordt de bestandsnaam gelezen en met regel 1650 wordt het eerste klusternummer gelezen dat bij dat bestand hoort. Vervolgens wordt met de regels 1700 en 1710 het bij de kluster behorende sektornummer uitgerekend. Nu wordt de betreffende sektor gelezen en wordt er gekeken van welk type het betreffende bestand is (regels 1760 tot en met 1810). Indien hierbij zou blijken dat het betreffende bestand een machinetaalprogramma bevat, dan wordt de subroutine van regels 2000 tot en met 2030 uitgevoerd. Na het lezen van de eerste sektor van een bestand moet de directory sektor weer worden ingelezen om een volgend bestand te kunnen onderzoeken. Dat wordt vanaf regel 1860 gedaan.

```
1000 '*********************************
1010 '*       Directory lezen         *
1020 '*********************************
1030 '
1040 CLEAR 1000: SCREEN 0: WIDTH 80: CLS
1050 INPUT "Afdrukken op printer (j/n)";I$
1060 IF I$<>"j" AND I$<>"J" AND I$<>"n" AND I$<>"N" THE
N 1050
1070 IF I$="n" OR I$="N" THEN OPEN "CRT:" AS #1
1080 IF I$="j" OR I$="J" THEN OPEN "LPT:" AS #1
1090 INPUT "Welke disk drive wilt u lezen (1=A, 2=B, enz
)";DD
1100 CLS
1110 '
1120 '*********************************
1130 '*       Lees boot sektor        *
1140 '*********************************
1150 I$=DSKI$(DD,0)
1160 MA=PEEK(&HF351)+256*PEEK(&HF352)
1170 'Aantal Fats:
1180 AF=PEEK(MA+16)
1190 'Sektors/Fat:
1200 SF=PEEK(MA+22)+256*PEEK(MA+23)
1210 'Eerste directory sektor:
1220 S1=AF*SF+1
1230 'Directory Entries:
1240 DE=PEEK(MA+17)+256*PEEK(MA+18)
1250 'Laatste directory sektor:
```

```
1260 SL=INT(DE/32)+S1-1
1270 IF DE/32<>INT(DE/32) THEN SL=SL+1
1280 'Aantal directory sektors:
1290 AD=SL-S1+1
1300 'Gereserveerde sektoren:
1310 GS=PEEK(MA+14)+256*PEEK(MA+15)
1320 'Aantal sektoren per kluster:
1330 SC=PEEK(MA+13)
1340 '
1350 '*********************************
1360 '*   Lees een directory sektor   *
1370 '*********************************
1380 I$=DSKI$(DD,S1)
1390 MA=PEEK(&HF351)+256*PEEK(&HF352)
1400 '
1410 '*********************************
1420 '*   Lees een directory entry    *
1430 '*********************************
1440 CLS
1450 PRINT  #1,"File-naam:      Sektor:  Type:     Begin
:    Eind:     Start:"
1460 FOR I=0 TO 15
1470 '
1480 '*********************************
1490 '*  File-naam + File-naam ext.  *
1500 '*********************************
1510 N$="": U$=""
1520 IF PEEK(MA+I*32)=&H0 THEN GOTO 1940
1530 IF PEEK(MA+I*32)=&HE5 THEN NEXT I
1540 FOR J=0 TO 7
1550 N$=N$+CHR$(PEEK(MA+I*32+J))
1560 NEXT J
1570 FOR J=8 TO 10
1580 U$=U$+CHR$(PEEK(MA+I*32+J))
1590 NEXT J
1600 PRINT #1,USING "\      \";N$;:PRINT #1,".";:PRINT #
1,USING"\ \";U$;
1610 '
1620 '*********************************
1630 '*  Eerste kluster in de file   *
1640 '*********************************
1650 CL=PEEK(MA+32*I+&H1A)+256*PEEK(MA+32*I+&H1B)
1660 '
```

```
1670 '*********************************
1680 '*  Eerste sektor van de file    *
1690 '*********************************
1700 ES=1+AF*SF+AD+GS+SC*CL-SC
1710 PRINT #1,USING "########";ES;
1720 '
1730 '*********************************
1740 '* Lees eerste sektor van file   *
1750 '*********************************
1760 I$=DSKI$(DD,ES)
1770 TY=PEEK(MA)
1780 IF TY=255 THEN PRINT #1,"      BASIC";
1790 IF TY=254 THEN PRINT #1,"      MCODE";:GOSUB 1960
1800 IF TY>47 AND TY<58 THEN PRINT #1,"      ASCII";
1810 IF TY<48 OR (TY>57ANDTY<254) THEN PRINT #1,"      ??
???";
1820 '
1830 '*********************************
1840 '*  Lees directory sektor weer   *
1850 '*********************************
1860 I$=DSKI$(DD,S1)
1870 PRINT #1,
1880 NEXT I
1890 LOCATE 0,20: PRINT "druk op willekeurige toets..."
1900 I$=INKEY$: IF I$="" THEN GOTO 1900
1910 LOCATE 0,20: PRINT SPC(36)
1920 S1=S1+1
1930 IF S1=<SL THEN GOTO 1380
1940 PRINT #1,: PRINT #1,: PRINT #1,"einde van de direct
ory"
1950 END
1960 '
1970 '*********************************
1980 '* Begin-, eind- en start-adres *
1990 '*********************************
2000 PRINT #1,USING "##########"; PEEK(MA+1)+256*PEEK(MA
+2);
2010 PRINT #1,USING "##########"; PEEK(MA+3)+256*PEEK(MA
+4);
2020 PRINT #1,USING "##########"; PEEK(MA+5)+256*PEEK(MA
+6);
2030 RETURN
```

De volgende afdruk laat een voorbeeld zien van de output van het voorgaande programma. U ziet daarin, dat het programma in staat is onderscheid te maken tussen BASIC, ASCII en machinetaal programma's. Ook ziet u de begin-, eind- en startadressen van machinetaalprogramma's. Wanneer een programma niet tot één van de drie genoemde soorten behoort, worden vraagtekens afgedrukt. U ziet dat gebeuren bij de MSXDOS.SYS en COMMAND.COM bestanden.

```
File-naam:        Sector:   Type:     Begin:    Eind:     Start:
MSXDOS   .SYS          9    ?????
COMMAND  .COM         14    ?????
HCOPY    .            27    BASIC
SCRNDU   .            28    MCODE     49152     49408     49152
HEXITMC  .            29    BASIC
MCHEXD   .            38    MCODE     53248     53504     53248
HEXIT    .            39    BASIC
SCRND002.BAK          45    ASCII
SCRND001.             52    BASIC
SCRND001.BAK          59    ASCII
SCRND001.ASC          66    ASCII
SCRND002.             74    BASIC
SCRND002.ASC          81    ASCII
SPEL     .            93    BASIC
APPB1    .BAK        108    ASCII
APPA     .BAK        121    ASCII
```

voorbeeld 9-4

De File Allocation Tabel

De meeste bestanden zijn zo groot, dat ze niet in één kluster passen. We hebben al gezien, dat een kluster soms slechts 1 sektor groot is en soms 2 sektoren groot. Er zijn ook systemen waar 3 sektoren per kluster worden gegroepeerd. Een sektor is 512 bytes groot. Een kluster kan dientengevolge 512, 1024 of soms zelfs 1536 bytes groot zijn. U zult het met ons eens zijn, dat de meeste bestanden wel groter zijn.

We weten nu, hoe we het eerste kluster van een bestand kunnen terugvinden. Maar, hoe vinden we nu die andere klusters terug die bij datzelfde bestand behoren? Als we dat eenmaal weten, weten we ook hoe het systeem zijn gegevens op schijf kan terugvinden.

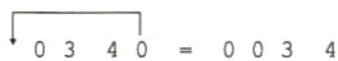
Om alle kluster, die bij een bepaald bestand behoren, te kunnen terugvinden, moeten we gebruik gaan maken van de FAT (File Allocation Tabel). Deze tabel staat op iedere schijf opgeslagen in de sektoren, direct volgend op de boot-sektor. In de boot-sektor staat, hoe lang de FAT is (uit hoeveel sektoren deze bestaat). In de meeste systemen

wordt de FAT twee maal vlak achter elkaar op schijf gezet. Dit wordt alleen uit veiligheid gedaan. Wanneer de FAT niet meer te lezen zou zijn, zou er praktisch geen enkel bestand meer terug te vinden zijn. Vandaar dat de FAT twee keer op schijf wordt gezet. Is de ene FAT niet meer te lezen, dan is er altijd nog een tweede.

De FAT bestaat uit een groot aantal entries. Voor iedere kluster is er een entry. Iedere entry is anderhalve byte lang. Deze maat van anderhalve byte maakt de indeling van de FAT erg gekompliceerd. Er zijn echter wat regeltjes en formules die we kunnen hanteren om toch de juiste entry te vinden.

Zoals gezegd, bevat de FAT een entry voor ieder kluster. Het kluster nummer is dan ook de ingang tot de FAT. Stel dat we uit de directory te weten zijn gekomen, dat kluster nummer 2 het eerste kluster van een bepaald bestand is. Om nu de entry in de FAT te vinden voor het betreffende klusternummer, moeten we het klusternummer met anderhalf vermenigvuldigen. Dit levert als resultaat de waarde 3 op. Dat betekent, dat de entry uit de FAT die we willen zien, op de posities 3 en 4 vanaf het begin van de FAT staan (er vanuit gaande, dat de FAT op positie 0 begint).

Wanneer we deze twee bytes uit de FAT lezen, dan zullen we (zeker op een pas geformateerde schijf, waarop nog maar één bestand is geschreven) naar alle waarschijnlijkheid de hexadecimale waarden 03 en 40 vinden. Nu komt de grote truuk. Omdat het kluster nummer waar deze entry bij hoort een even nummer was, moeten we de cijfers uit de gevonden waarden als volgt verschuiven:



Van het verschoven resultaat gebruiken we alleen de linker drie tetraden (3 tetraden is gelijk aan anderhalve byte). En zoals u ziet, bevatten de linker drie tetraden het getal 003. Dat wil zeggen dat het volgende kluster, waar een deel van het bestand staat, kluster nummer 3 is.

Nu kunnen we weer gaan rekenen. Anderhalf keer het nieuw gevonden kluster nummer (3) is 4 en een half. Hiervan nemen we alleen het gehele deel. Zo krijgen we de waarde 4. Dat betekent, dat de entry uit

de FAT die we nu willen zien op de posities 4 en 5 vanaf het begin van de FAT staat. Lezen we die posities uit de FAT, dan zullen we naar alle waarschijnlijkheid de hexadecimale waarden 40 en 00 vinden. De tweede grote truuk moet worden uitgehaald, omdat deze entry bij een oneven kluster nummer hoort. Weer moeten we met de cijfers uit de gevonden waarden schuiven, doch nu als volgt:

En omdat het kluster nummer oneven was, gebruiken we van het verschoven resultaat alleen de drie rechter tetraden. Daarin staat nu het getal 004, hetgeen wil zeggen, dat het volgende kluster waarin een deel van het bestand staat, kluster nummer 4 is.

Wanneer we op deze manier doorgaan met het uitkoderen van de FAT, zullen we op een gegeven moment een klusternummer tegenkomen, dat de hexadecimale waarde FF8 of hoger heeft. Wanneer we die waarde tegenkomen, weten we dat het laatst gevonden kluster het laatste kluster was, waarin een deel van ons bestand was opgeslagen.

Voor de volledigheid zij nog vermeld, dat u in een FAT ook de waarde FF7 kunt tegenkomen. Die waarde wordt door sommige formateringsprogramma's in de FAT gezet op de plaats van die klusters, waarin tijdens het formateren lees/schrijffouten zijn gevonden. Door in die kluster-posities de waarde FF7 te zetten, zal het systeem tijdens het wegschrijven van bestanden het betreffende kluster overslaan. Op die manier kan een schijf waarop enkele slechte plekjes zitten, toch veilig worden gebruikt, al wordt de totale kapaciteit van de schijf natuurlijk wel wat kleiner.

Met het volgende programma wordt u gevraagd een bestandsnaam op te geven. Het programma zoekt het opgegeven bestand op in de directory en zoekt er het eerste klusternummer bij. Hierna wordt de bij dat kluster behorende FAT-entry opgezocht in de FAT en wordt daaruit het volgende klusternummer gedestilleerd. Het programma doet precies hetzelfde als wat wij zojuist in theorie hebben gedaan.

Met de regels 1400 tot en met 1570 wordt de bestandsnaam opgevraagd. U kunt eerst de bestandsnaam en vervolgens de bestandsnaamuitbreiding ingeven. Daar de bestandsnaam op schijf in hoofdletters

staat, wordt uw ingave omgezet in hoofdletters, voordat deze wordt vergeleken met de bestandsnamen uit de directory. Het opzoeken van de namen uit de directory en het vergelijken wordt gedaan met de regels 1690 tot en met 1830. Is het gevraagde bestand gevonden, dan wordt naar regel 1880 gesprongen. Na het lezen van het eerste klusternummer van het bestand wordt de bijbehorende FAT entry gelezen met de regels 1940 tot en met 2010. Na het afdrukken van de gevonden gegevens (regels 2060 tot en met 2140) wordt het volgende klusternummer bepaald. U ziet in regel 2200, dat het programma wordt beëindigd, indien het gevonden kluster nummer hexadecimaal FF7 of hoger is.

```
1000 '*********************************
1010 '* File-gegevens lezen uit FAT  *
1020 '*********************************
1030 '
1040 CLEAR 1000: SCREEN 0: WIDTH 80: CLS
1050 INPUT "Afdrukken op printer (j/n)";I$
1060 IF I$<>"j" AND I$<>"J" AND I$<>"n"AND I$<>"N"THEN G
OTO 1050
1070 IF I$="n" OR I$="N" THEN OPEN "CRT:" AS #1
1080 IF I$="j" OR I$="J" THEN OPEN "LPT:" AS #1
1090 INPUT "Welke disk drive wilt u lezen (1=A, 2=B, enz
)";DD
1100 CLS
1110 '
1120 '*********************************
1130 '*       Lees boot sektor        *
1140 '*********************************
1150 I$=DSKI$(DD,0)
1160 MA=PEEK(&HF351)+256*PEEK(&HF352)
1170 'Aantal Fats:
1180 AF=PEEK(MA+16)
1190 'Sektors/Fat:
1200 SF=PEEK(MA+22)+256*PEEK(MA+23)
1210 'Eerste directory sektor:
1220 S1=AF*SF+1
1230 'Directory Entries:
1240 DE=PEEK(MA+17)+256*PEEK(MA+18)
1250 'Laatste directory sektor:
1260 SL=INT(DE/32)+S1-1
1270 IF DE/32<>INT(DE/32) THEN SL=SL+1
1280 'Aantal directory sektors:
```

```
1290 AD=SL-S1+1
1300 'Gereserveerde sektoren:
1310 GS=PEEK(MA+14)+256*PEEK(MA+15)
1320 'Aantal sektoren per kluster:
1330 SC=PEEK(MA+13)
1340 'Logische Sektornr. Offset:
1350 LS=1+AF*SF+AD+GS+SC
1360 '
1370 '*********************************
1380 '*      Opvragen File-naam       *
1390 '*********************************
1400 CLS
1410 IF DD=0 OR DD=1 THEN FILES "a:"
1420 IF DD=2 THEN FILES "b:"
1430 IF DD=3 THEN FILES "c:"
1440 IF DD=4 THEN FILES "d:"
1450 PRINT #1,:PRINT #1,
1460 INPUT "Filenaam: ";F$
1470 INPUT "Extensie: ";E$
1480 PRINT #1,
1490 L=LEN(F$):IF L<8 THEN FOR I=L TO 7: F$=F$+" ": NEXT I
1500 L=LEN(E$):IF L<3 THEN FOR I=L TO 2: E$=E$+" ": NEXT I
1510 FOR I=1 TO LEN(F$)
1520 F=ASC(MID$(F$,I,1)): IF F>96 AND F<123 THEN MID$(F$
,I,1)=CHR$(F-32)
1530 NEXT I
1540 FOR I=1 TO LEN(E$)
1550 F=ASC(MID$(E$,I,1)): IF F>96 AND F<123 THEN MID$(E$
,I,1)=CHR$(F-32)
1560 NEXT I
1570 F$=F$+"."+E$
1580 '
1590 '*********************************
1600 '*   Lees een directory sektor   *
1610 '*********************************
1620 I$=DSKI$(DD,S1)
1630 MA=PEEK(&HF351)+256*PEEK(&HF352)
1640 FOR I=0 TO 15
1650 '
1660 '*********************************
1670 '* Filenaam zoeken in directory *
1680 '*********************************
1690 IF PEEK(MA+I*32)=&H0 THEN GOTO 2220
```

```
1700 IF PEEK(MA+I*32)=&HE5 THEN GOTO 1800
1710 N$="": U$=""
1720 FOR J=0 TO 7
1730 N$=N$+CHR$(PEEK(MA+I*32+J))
1740 NEXT J
1750 FOR J=8 TO 10
1760 U$=U$+CHR$(PEEK(MA+I*32+J))
1770 NEXT J
1780 N$=N$+"."+U$
1790 IF N$=F$ THEN GOTO 1880
1800 NEXT I
1810 S1=S1+1
1820 IF S1=<SL THEN GOTO 1620
1830 GOTO 2220
1840 '
1850 '*********************************
1860 '*  Eerste kluster van de file   *
1870 '*********************************
1880 PRINT #1,"File-gegevens van ";F$: PRINT #1,
1890 CL=PEEK(MA+32*I+&H1A)+256*PEEK(MA+32*I+&H1B)
1900 '
1910 '*********************************
1920 '*  Lees een entry uit de FAT.   *
1930 '*********************************
1940 FP=INT(CL*1.5)
1950 F1=FP: F2=FP+1: F3=FP+2
1960 J=INT((F1+1)/512):F1=F1-J*512
1970 I$=DSKI$(DD,J+1)
1980 F1=PEEK(MA+F1)
1990 J=INT((F2+1)/512):F2=F2-J*512
2000 I$=DSKI$(DD,J+1)
2010 F2=PEEK(MA+F2)
2020 '
2030 '*********************************
2040 '*      Druk de gegevens af      *
2050 '*********************************
2060 H1$=RIGHT$("00"+HEX$(F1),2)
2070 H2$=RIGHT$("00"+HEX$(F2),2)
2080 PRINT #1,"FAT-entry = ";H1$;" ";H2$;
2090 PRINT #1,"    Kluster =";USING "####";CL;
2100 PRINT #1,"    sektor(s): ";
2110 FOR S=1 TO SC
2120 PRINT #1,USING "####";(CL-2)*SC+LS+S-1;
```

```
2130 NEXT S
2140 PRINT #1,
2150 '
2160 '********************************
2170 '*  Bepaal volgende kluster nr  *
2180 '********************************
2190 IF CL/2=INT(CL/2) THEN CL=(F2 AND 15)*256+F1 ELSE C
L=F2*16+(F1 AND 240)/16
2200 IF CL=>&HFF7 THEN PRINT #1,"Laatste kluster": END
2210 GOTO 1940
2220 PRINT #1,: PRINT #1,: PRINT #1,F$;" niet gevonden."
2230 END
```

Ook bij dit programma nog een paar voorbeeldjes van de werking. Het bestand HEXIT staat blijkbaar op een 5,25 inch enkelzijdige disk drive, met slechts één sektor per kluster. Immers, de klusternummers en de sektornummers lopen gelijkelijk op.

Een belangrijk punt is te zien in de gegevens van het bestand LEESDIR.ASC. U ziet daarin, dat het bestand in de klusters 61, 62, 65 en 66 staat. Met andere woorden, een bestand hoeft niet altijd op aaneengesloten klusters te staan. Wanneer een bestand naar schijf wordt geschreven, wordt het eerste deel van het bestand op het eerste vrije kluster gezet. Wanneer dat kluster vol is, wordt het eerstvolgende vrije kluster gezocht. Dat kluster kan een heel eind verder op de schijf staan.

```
File-gegevens van HEXIT

FAT-entry = 03 40     Cluster =   2     sector(s):    9
FAT-entry = 40 00     Cluster =   3     sector(s):   10
FAT-entry = 05 60     Cluster =   4     sector(s):   11
FAT-entry = 60 00     Cluster =   5     sector(s):   12
FAT-entry = 07 80     Cluster =   6     sector(s):   13
FAT-entry = 80 00     Cluster =   7     sector(s):   14
FAT-entry = 09 A0     Cluster =   8     sector(s):   15
FAT-entry = A0 00     Cluster =   9     sector(s):   16
FAT-entry = 0B C0     Cluster =  10     sector(s):   17
FAT-entry = C0 00     Cluster =  11     sector(s):   18
FAT-entry = FF EF     Cluster =  12     sector(s):   19
Laatste cluster
```

```
File-gegevens van DATACOM .ASC

FAT-entry = 13 40     Cluster =  18     sector(s):   44  45
FAT-entry = 40 01     Cluster =  19     sector(s):   46  47
FAT-entry = 15 60     Cluster =  20     sector(s):   48  49
FAT-entry = 60 01     Cluster =  21     sector(s):   50  51
FAT-entry = 17 F0     Cluster =  22     sector(s):   52  53
FAT-entry = F0 FF     Cluster =  23     sector(s):   54  55
Laatste cluster


File-gegevens van LEESFILE.ASC

FAT-entry = 8F 04     Cluster =  71     sector(s):  150 151
FAT-entry = 49 A0     Cluster =  72     sector(s):  152 153
FAT-entry = A0 04     Cluster =  73     sector(s):  154 155
FAT-entry = FF CF     Cluster =  74     sector(s):  156 157
Laatste cluster


File-gegevens van LEESDIR .ASC

FAT-entry = E0 03     Cluster =  61     sector(s):  130 131
FAT-entry = 41 F0     Cluster =  62     sector(s):  132 133
FAT-entry = 20 04     Cluster =  65     sector(s):  138 139
FAT-entry = FF FF     Cluster =  66     sector(s):  140 141
Laatste cluster
```

voorbeeld 9-5

Wanneer u nu met het programma, waarmee u iedere gewenste sektor op het beeldscherm kunt afdrukken, de FAT sektoren afdrukt, dan zal het u opvallen dat er heel veel entries in die FAT staan die de hexadecimale waarde 000 hebben. Dit is vooral het geval in de FAT van een schijf die nog niet zo heel erg vol is. Wanneer een entry 000 is, dan wil dat zeggen dat het betreffende kluster nog vrij is. Wanneer een bestand op schijf moet worden geschreven zal het systeem naar een entry zoeken, die nog 000 is.

Nu weten we alles van de FAT en hoe het systeem daarvan gebruik maakt. In de paragraaf over de directory leerden we hoe voor ieder bestand een aantal gegevens, zoals het eerste klusternummer van dat bestand, werd opgeslagen. Daarin werd ook vermeld, dat de eerste letter van een gewist bestand wordt vervangen door de hexadecimale kode E5. De rest van de directory entry wordt echter intact gelaten. Door eenvoudigweg de kode E5 te vervangen door de kode van een letter, kunnen we het bestand weer in de directory laten opnemen. We

hebben dan ook de verwijzing naar het eerste kluster, dat bij dat bestand behoort, teruggevonden.

Indien er tussen het moment van het wissen van het bestand en het moment waarop de kode E5 weer wordt vervangen door een letter, geen andere bestanden naar de schijf zijn geschreven, dan zullen alle sektoren waar het bestand stond, voordat het werd gewist, nog de informatie van het gewiste bestand bevatten. Het wissen van een bestand betekent namelijk niet dat alle sektoren worden gewist. Alleen de administratie van die sektoren wordt gewist of verminkt. Dat wil dus zeggen, dat de door het bestand gebruikte klusters in de FAT-administratie weer worden vrijgegeven voor gebruik door andere bestanden (door in de betreffende entries de waarde 000 te zetten) en dat de eerste letter van de bestandsnaam in de directory de kode E5 wordt.

Het is in principe mogelijk een per ongeluk gewist bestand weer toegankelijk te maken. We moeten dan uitzoeken in welke sektoren dat bestand staat. De eerste sektor (het eerste kluster) kunnen we uit de directory halen. De volgende bij dat bestand behorende sektoren moeten na het eerste kluster volgen. Door het programma te draaien waarmee de inhoud van sektoren op het beeldscherm wordt afgedrukt, kunnen de bij hetzelfde bestand behorende sektoren worden gevonden. Vervolgens kunnen we met behulp van de tot nu toe opgedane kennis de klusternummers berekenen en zelf een nieuwe FAT opbouwen.

Om in een directory sektor een byte te kunnen wijzigen (bijvoorbeeld van de kode E5 naar een letter) of om een FAT-sektor te kunnen wijzigen, moeten we in staat zijn een sektor naar schijf te schrijven. Het naar schijf schrijven van een sektor is net zo eenvoudig als het lezen van een sektor. Hiertoe kunnen we gebruik maken van het volgende statement:

```
DSK0$ <drive>,<sektor nr.>
```

Merk op, dat er in het formaat van dit statement, in tegenstelling tot het formaat van de funktie DSKI$, geen haakjes voorkomen.

Voordat we dit statement geven, moeten we de informatie die we naar schijf willen schrijven gereed zetten in het geheugengebied, waarvan het beginadres te vinden is in de systeemlokaties &HF351 en &HF352.

Waarschuwing:
Voordat we een programma gaan bekijken, waarmee we sektoren op schijf kunnen wijzigen, willen we u waarschuwen voor de schade, die u daarmee kunt aanrichten. Het is zelfs mogelijk, dat u, door verkeerde informatie in de boot-sektor te zetten, een schijf volkomen onleesbaar maakt. Wees erg voorzichtig met het volgende programma en gebruik het alleen op schijven waarvan u een goede back-up heeft.

Het volgende programma vraagt u, van welke schijf u een sektor wilt zien en welke sektor u wilt zien. Vervolgens wordt u gevraagd, vanaf welk byte u de sektor wil zien. Hierop drukt het programma 16 bytes af in zowel hexadecimaal als in ASCII-kode. U ziet dan dat de kursor op de eerste hexadecimale kode staat. Nu kunt u een nieuwe hexadecimale kode intikken. Alle waarden die niet hexadecimaal zijn worden genegeerd. Ook kleine letters in plaats van A tot en met F worden genegeerd. U kunt dus alleen werkelijk hexadecimale ingaven plegen. Met de kursortoetsen kunt u de kursor naar links en naar rechts verplaatsen. Heeft u de gewenste wijziging aangebracht, dan wordt het geheugengebied, dat was gevuld met DSKI$ en dat u met het programma heeft gewijzigd, met een DSKO$-statement weer naar schijf geschreven. Hierna kunt u ofwel doorgaan (en nog eens kontroleren of de gemaakte wijzigingen goed zijn aangebracht), of stoppen.

De listing is van zoveel kommentaar voorzien, dat het ons niet nodig lijkt er nog meer aandacht aan te besteden. Nogmaals willen wij u waarschuwen voor de effekten die dit programma op uw schijven kan hebben, indien u per ongeluk een verkeerd byte wijzigt. Wees voorzichtig, ga uiterst nauwkeurig te werk, en zorg altijd voor een goede backup van uw schijf, voordat u er met dit programma op gaat schrijven.

```
100 '*********************************
110 '* Inhoud van sektoren wijzigen *
120 '*********************************
130 '
140 CLS: SCREEN 0: WIDTH 80: CLEAR 1000
150 INPUT "Schijfnummer";DK
160 IF DK<0 OR DK >6 THEN GOTO 150
170 IF DK=0 THEN DK=1
180 INPUT "Sektornummer";SN
190 '
200 '*********************************
210 '*   Gevraagde sektor inlezen    *
220 '*********************************
230 I$=DSKI$(DK,SN)
```

```
240 MA=PEEK(&HF351)+256*PEEK(&HF352)
250 '
260 '*********************************
270 '* 16 bytes, vanaf B0 afdrukken *
280 '*********************************
290 INPUT "Vanaf welke byte wilt u de sektor zien"; B0
300 LOCATE 0,10: PRINT USING "###";B0;:PRINT "   ";
310 FOR I=B0 TO B0+15
320 A=PEEK(MA+I)
330 LOCATE 6+(I-B0)*3,8:IF CHR$(A)>" " AND CHR$(A)<"" T
HEN PRINT CHR$(A);
340 LOCATE 6+(I-B0)*3,10:PRINTRIGHT$("00"+HEX$(PEEK(MA+I
)),2);
350 NEXT I
360 LOCATE 0,15: PRINT "Plaats kursor op het te wijzigen
byte"
370 PRINT "Tik een nieuwe hexadecimale waarde in"
380 '
390 '*********************************
400 '* wijzig op scherm en geheugen *
410 '*********************************
420 X=0:P=0:H$=""
430 LOCATE X*3+P+6,10,1
440 I$=INKEY$: IF I$="" THEN GOTO 440
450 IF I$=CHR$(27) THEN GOTO 540
460 IF I$=CHR$(29) THEN GOSUB 640
470 IF I$=CHR$(28) THEN GOSUB 680
480 IF (I$=>"0" AND I$=<"9") OR (I$=>"A" AND I$=<"F") TH
EN GOSUB 720
490 GOTO 430
500 '
510 '*********************************
520 '*      Wijzigen op schijf       *
530 '*********************************
540 DSKO$ DK,SN
550 LOCATE 0,18: PRINT "Gewijzigde sektor weggeschreven.
Doorgaan? (j/n)";
560 I$=INKEY$:IF I$="" THEN GOTO 560
570 IF I$<>"j" AND I$<>"J" THEN CLS: END
580 LOCATE 0,8: PRINT SPC(60)
590 LOCATE 0,18: PRINT SPC(60)
600 LOCATE 0,0
610 GOTO 150
```

```
620 '
630 '***** subroutine LINKS
640 IF X>0 THEN X=X-1
650 RETURN
660 '
670 '***** subroutine RECHTS
680 IF X<15 THEN X=X+1
690 RETURN
700 '
710 '***** subroutine WIJZIGEN
720 IF P=0 THEN H$=H$+I$: P=1: PRINT I$;: RETURN
730 IF P=1 THEN H$=H$+I$: P=0: PRINT I$;: POKE MA+B0+X,V
AL("&H"+H$): H$="":IF X<15 THEN X=X+1
740 RETURN
```

9.3 Wat weet de computer over schijven?

Wat wij tot nu toe in dit hoofdstuk hebben gezien, is wat er allemaal op schijf staat. Wat we nog niet hebben bekeken is, hoe de computer van al deze informatie gebruik maakt. Wel kunnen we ons nu een voorstelling maken van wat de computer allemaal moet doen. Immers wij hebben nu zelf een aantal programma's gemaakt waarmee we de schijf kunnen lezen en beschrijven.

Over het algemeen zal de computer echter meer doen dan wij tot nu toe met onze programma's hebben gedaan. Immers, de computer moet niet alleen bestanden op de schijf opzoeken, lezen en schrijven, maar moet uit die bestanden ook nog bepaalde stukjes (records) lezen. Wat de computer eigenlijk moet doen is, een gebruikersvriendelijke "interface" zijn tussen ons en de schijf. Wij geven de computer door middel van een aantal relatief eenvoudige BASIC-statements een aantal opdrachten, en de computer voert die uit, hoe ingewikkeld dat ook mag zijn. Wij zeggen tegen de computer dat we record nummer 35 uit bestand X willen lezen en de computer zoekt uit welke schijf en welke sektoren daartoe moeten worden gelezen en geeft ons uiteindelijk het gevraagde record terug.

Voordat wij de computer kunnen vragen een record voor ons uit een bestand te lezen, zullen we de computer dat bestand eerst moeten laten openen. Wat doet de computer nu, wanneer wij hem hebben opgedragen een bepaald bestand te openen? De computer zal eerst kijken

of het opgegeven bestand inderdaad op de opgegeven schijf staat. Is het bestand gevonden, dan wordt voor dat bestand een administratie-gebied aangemaakt. Een dergelijk gebied wordt een *File Control Block* (kortweg FCB) genoemd. Wij merken daar niets van, maar de computer houdt in dat File Control Block alle voor de toegang van dat bestand belangrijke gegevens bij.

In werkelijkheid houdt de computer zelfs twee File Control Blocks bij. Het ene blok wordt gebruikt tussen de systeemsoftware (in ROM) en het applikatieprogramma (dit zullen we vanaf nu het BASIC-FCB noemen) en het andere wordt gebruikt tussen de systeemsoftware en de schijveneenheid (dit noemen we het DOS-FCB).

Hoewel wij als BASIC-programmeurs van deze File Control blokken niets hoeven te weten, zal enige kennis van de inhoud van die blokken ons toch wel wat meer inzicht in de werking van onze computer verschaffen, voorzover het tenminste de verwerking van schijvenbestanden betreft. We zullen dan ook zodadelijk de genoemde FCB's opzoeken en met een programma de inhoud ervan afdrukken.

Voor we daaraan beginnen is er echter nog iets dat we moeten weten. Wanneer we de computer aanschakelen, en we hebben aan die computer een eveneens aangeschakelde schijveneenheid aangesloten, dan worden er automatisch twee schijveneenheden geïnitialiseerd. In werkelijkheid is er misschien maar één schijveneenheid, maar het systeem doet alsof er twee zijn. We kunnen de computer dan ook opdracht geven, om een kopie van de schijf in eenheid A: te maken, naar de schijf in eenheid B:. Wanneer de kopie wordt gemaakt, zal de computer, na enige tijd de schijf in eenheid A: te hebben gelezen, ons vragen de schijf in eenheid B: te laden, waarna de ingelezen informatie naar de schijf in eenheid B: wordt geschreven. We kunnen echter tijdens het opstarten van de computer op de Control-toets drukken. In dat geval wordt er slechts één schijveneenheid geïnitialiseerd. Wanneer we een tweede schijveneenheid met een eigen ROM-cartridge op onze computer aansluiten, kunnen we zelfs na het aanschakelen plotseling vier schijveneenheden hebben.

Het aantal schijven dat aan het systeem is aangesloten, wordt tijdens het opstarten van de computer onderzocht en in het systeemgeheugen vastgelegd. Vandaar dat het geen zin heeft, om na het opstarten van het systeem nog een andere schijveneenheid aan te schakelen. Wanneer die

extra schijveneenheid niet was aangeschakeld op het moment dat we de computer aanschakelden, dan wordt die niet geïnitialiseerd.

Waarom is het nu zo belangrijk, dat de computer weet, welke schijveneenheden er aan het systeem zijn aangesloten? Welnu, wanneer wij per ongeluk een verkeerde schijveneenheid opgeven, dan kan de computer ons meteen vertellen, dat de opgegeven schijveneenheid niet aan het systeem is aangesloten. Bovendien is er voor iedere schijveneenheid een stukje werkgeheugen nodig. Dat stukje werkgeheugen gaat van de geheugenruimte voor het BASIC af. Het is uiteraard zeer gewenst, dat we meteen vanaf het begin weten hoeveel geheugen er voor BASIC beschikbaar is.

Het volgende programma laat zien, wat er in het systeemgeheugen aan informatie over schijveneenheden wordt opgeslagen tijdens het aanschakelen van de computer. Vanaf adres &HFB21 wordt een soort tabel gemaakt, waarin per *"slot"* het aantal geïnitialiseerde schijveneenheden staat.

```
100 '********************************
110 '* Geinitialiseerde disk-drives *
120 '********************************
130 '
140 CLS
150 PRINT "Aantal  Slot"
160 PRINT "disks:  nr.:"
170 PRINT "------------"
180 FOR I=0 TO 6 STEP 2
190 IF PEEK(&HFB21+I)=0 THEN GOTO 230
200 PRINT USING "####";PEEK(&HFB21+I);
210 PRINT "      ";RIGHT$("00"+HEX$(PEEK(&HFB22+I)),2)
220 NEXT I
230 PRINT "------------"
240 END
```

De computer weet dus, hoeveel schijveneenheden er zijn aangesloten, en via welke uitbreidings "slots" deze schijven zijn te benaderen.

Wanneer wij nu een bestand openen, dan kontroleert de computer of de schijveneenheid, die in het OPEN-statement wordt genoemd, wel aanwezig is. Is dat zo, dan wordt voor het nieuw te openen bestand een FCB gemaakt. Ook die FCB's zijn terug te vinden. Daartoe dienen we

gebruik te maken van de funktie VARPTR. Wanneer we VARPTR laten volgen door het nummer, waaronder we het bestand hebben geopend, dan krijgen we het adres van het eerste byte van het BASIC-FCB. In het BASIC-FCB vinden we een adres, dat verwijst naar het DOS-FCB.

Zolang we in BASIC werken, hoeven we zelf niets met die FCB's te doen. Daarom zullen we hier niet dieper ingaan op de indeling van deze control blokken. Het volgende programma zoekt de FCB's echter op en drukt de inhoud af van de verschillende velden die erin voorkomen. De output van het programma geeft meteen een goed overzicht van alle FCB-velden.

In regel 150 wordt een bestand geopend. Met regel 160 wordt een record naar dat bestand geschreven. Door het openen van het bestand zullen de FCB's voor dat bestand worden aangemaakt.

In het BASIC-FCB zit een buffer van 256 bytes. Dit buffer wordt gebruikt voor het tijdelijk opslaan van de records die naar het bestand moeten worden geschreven of die uit het bestand worden gelezen. In het geval van schrijven naar een bestand zullen de records vanuit dit buffer naar het geheugengebied worden geschreven waar de inhoud van de sektor staat, waarna de sektorinhoud naar de schijf wordt geschreven. In het geval van lezen, zal een sektor van schijf worden gelezen, waarna een record van de sektorinhoud in het buffer wordt gekopieerd. Met de VARPTR-funktie van regel 170 wordt het start-adres van het BASIC-FCB gelezen.

Met de regels 180 tot en met 370 wordt het BASIC-FCB op het scherm afgedrukt. Let daarbij vooral op regel 230, waarmee het adres van het DOS-FCB uit het BASIC-FCB wordt gelezen. Het hier gevonden DOS-FCB-adres wordt vanaf regel 400 gebruikt, om er de inhoud van het DOS-FCB mee op te zoeken en af te drukken. Het zal u opvallen, dat de gegevens in het DOS-FCB voor een groot deel afkomstig zijn uit de directory en uit de FAT. Hieruit kunnen we dan ook opmaken, dat dit FCB wordt gebruikt voor het verkrijgen van toegang tot de schijf.

```
100 '*****************************
110 '*    File Control Blokken    *
120 '*****************************
130 '
```

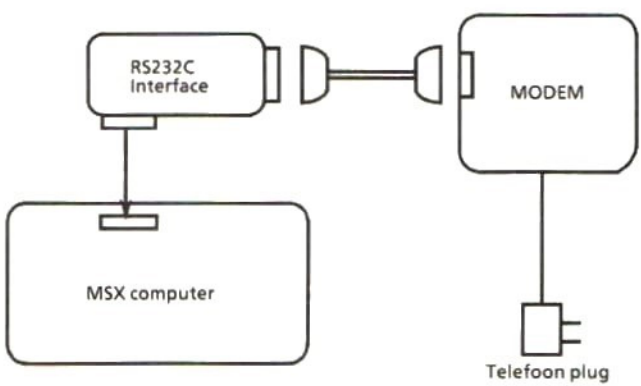
```
140 CLS: SCREEN 0: WIDTH 80: CLEAR 1000
150 OPEN "a:wessel.tst" FOR OUTPUT AS #1
160 PRINT #1,"Dit is een test-record dat we terug zien i
n de data-buffer"
170 MA=VARPTR(#1)+65536!
180 PRINT "FCB-BASIC:": PRINT
190 PRINT "Mode waarin bestand werd geopend:   ";
200 IF PEEK(MA)=1 THEN PRINT "INPUT"
210 IF PEEK(MA)=2 THEN PRINT "OUTPUT"
220 IF PEEK(MA)=4 THEN PRINT "RANDOM"
230 FC=PEEK(MA+1)+256*PEEK(MA+2)
240 PRINT "Adres van het FCB-DOS              ";FC
250 PRINT "Het Back-up teken is:              ";CHR$(PEE
K(MA+3))
260 PRINT "De disk waarop het bestand staat:  ";PEEK(MA+
4)
270 PRINT "Positie binnen databuffer is:      ";PEEK(MA+
6)
280 PRINT "Flag-informatie:                    ";RIGHT$(
"00"+H
EX$(PEEK(MA+7)),2)
290 PRINT "Pseudo-kop positie:                ";PEEK(MA+
8)
300 PRINT "Hierna volgt het data-buffer:"
310 PRINT
320 FOR J=0 TO 3
330 FOR I=0 TO 63
340 PRINT CHR$(PEEK(MA+9+J*64+I));
350 NEXT I
360 PRINT
370 NEXT J
380 PRINT: PRINT: PRINT "Druk op een willekeurige toets
om FCB-DOS te zien"
390 I$=INKEY$:IF I$="" THEN GOTO 390
400 CLS: PRINT "FCB-DOS:"
410 PRINT
420 PRINT "Nummer van de schijveneenheid:     ";PEEK(FC)
430 PRINT "De file naam van het bestand:       ";
440 FOR I=1 TO 8
450 PRINT CHR$(PEEK(FC+I));
460 NEXT I
470 PRINT ".";
480 FOR I=9 TO 11
```

```
490 PRINT CHR$(PEEK(FC+I));
500 NEXT I
510 PRINT
520 PRINT "Het huidige kluster nummer is:     ";PEEK(FC+
12)+256*PEEK(FC+13)
530 PRINT "De record grootte is:              ";PEEK(FC+
14)+256*PEEK(FC+15)
540 PRINT "File grootte is:                   ";PEEK(FC+
16)+256*PEEK(FC+17)+65536!*PEEK(FC+18)
550 PRINT "De datum is:                       ";PEEK(FC+
20) AND 31;"-";
560 PRINT (PEEK(FC+20) AND 224)/32+(PEEK(FC+21) AND 1)*8
;"-";
570 PRINT (PEEK(FC+21) AND 254)/2+1980
580 PRINT "De tijd is:                        ";(PEEK(FC
+23) AND 248)/8;":";
590 PRINT (PEEK(FC+23) AND 7)*8+(PEEK(FC+22) AND 224)/32
;":";
600 PRINT (PEEK(FC+22) AND 31)*2
610 PRINT "Device identification is:          ";CHR$(PE
EK(FC+24));":"
620 PRINT "Directory entry nummer is:         ";PEEK(FC+
25)
630 PRINT "Eerste kluster van de file is:     ";PEEK(FC+
26)+256*PEEK(FC+27)
640 PRINT "Laatst gebruikte kluster is:       ";PEEK(FC+
28)+256*PEEK(FC+29)
650 PRINT "Afstand vanaf eerste kluster is:   ";PEEK(FC+
30)+256*PEEK(FC+31)
660 PRINT "Het huidige record is:             ";PEEK(FC+
32)
670 PRINT "Random Record pointer              ";
680 FOR I=0 TO 3
690 PRINT RIGHT$("00"+HEX$(PEEK(FC+33+I)),2);" ";
700 NEXT I
710 CLOSE
720 END
```

HOOFDSTUK 10

DE OPTIONELE SERIËLE INTERFACE

De titel van dit hoofdstuk geeft het al aan, de seriële interface (RS232C) is niet op alle MSX computers aanwezig. Voor de meeste MSX computers geldt, dat er standaard geen RS232C interface aanwezig is. Een dergelijke interface dient men als optie bij de computer aan te schaffen. Die optie kan zowel op MSX1 als op MSX2 computers worden gebruikt. *Afbeelding 10-1* laat een konfiguratie zien van een MSX computer met een RS232C interface en een modem. In plaats van het modem zou ook een printer of een andere computer kunnen worden aangesloten. Zoals we later zullen zien heeft het aansluiten van een bepaald apparaat ook bepaalde konsekwenties voor de instelling en werking van de RS232C interface.



Afb. 10-1 Een konfiguratie met RS232C interface en modem

Een RS232C interface is een seriële interface. Dit wil zeggen, dat er tekens bit voor bit over die interface kunnen worden verstuurd. Doordat de tekens slechts per bit tegelijk worden verstuurd is er slechts één

enkele draad nodig om er de tekens over te versturen. Bij parallelle gegevensoverdracht zijn – afhankelijk van de gebruikte kode – zeven of acht draden nodig om een teken te versturen.

Met die ene draad voor het versturen van de bits van een teken zijn we er echter nog niet. Om ook tekens serieel te kunnen ontvangen zullen we nog een draad nodig hebben. Verder hebben we nog een aantal draden nodig waarmee we het zenden en ontvangen kunnen kontroleren en besturen. Straks zullen we zien hoe het proces van zenden en ontvangen in zijn werk gaat.

Daar we de seriële interface meestal zullen gebruiken om er via de telefoonlijn een verbinding met een andere computer mee op te bouwen, zullen we tussen de seriële interface en de telefoonlijn ook nog een modem moeten schakelen. Ook voor het besturen van die modem zijn nog een aantal kontrole en besturingssignalen op de seriële interface aanwezig. Het woord modem is een samentrekking van de woorden *modulator* en *demodulator*. Voordat de digitale computersignalen op de telefoonlijn kunnen worden gezet, moeten deze eerst gemoduleerd worden. Aan de andere kant van de telefoonlijn moeten deze gemoduleerde signalen eerst worden gedemoduleerd, voordat ze weer als digitaal signaal in de ontvangende computer kunnen worden gebruikt.

Wanneer we een printer met een seriële interface willen aansluiten op de seriële interface van onze MSX computer, en de printer staat op een tamelijk korte afstand van de computer, dan behoeven we niet via een modem te werken. In dat geval kunnen we de beide seriële interfaces direkt koppelen door middel van een meerdraads kabeltje. In dit hoofdstuk zullen we beide soorten verbindingen (direkt en via modem) kort behandelen, voordat we overgaan tot het behandelen van de BASIC-statements. De BASIC-statements stellen ons in staat gegevens tussen computers onderling of tussen computers en randapparaten (zoals printers) uit te wisselen. Maar, allereerst zullen we kijken hoe de gegevens worden overgedragen.

10.1 Overdracht van teken-kodes

De gegevensoverdracht over de seriële interface gaat volgens het asynchrone principe. Dit wil zeggen, dat de ontvanger van gegevens op ieder afzonderlijk teken opnieuw moet synchroniseren.

U moet zich dat zo voorstellen. Zolang er niets wordt verzonden maakt het zendende station de zendlijn "hoog". De ontvanger weet dan, dat er niets wordt verzonden. Zodra de zender een teken wil versturen, maakt deze eerst de lijn "laag" gedurende de tijd die nodig is voor het versturen van een bitje. Dit wordt dan ook een *startbit* genoemd. Voor de ontvanger is dit een teken dat de zender een teken gaat versturen. Op de overgang van de lijn van hoog naar laag start de ontvanger een synchronisatie klok. Deze klok geeft aan wanneer er een volgend bit van het teken op de lijn staat. Steeds wanneer er een volgend bit op de lijn staat zal de ontvanger de waarde van dat bit inlezen.

Om de overdracht suksesvol te laten zijn zal de klok van de ontvanger op dezelfde snelheid moeten draaien als de klok, waarmee de zender zijn bitjes op de lijn zet. De snelheid waarmee bitjes op de lijn worden gezet wordt uitgedrukt in **baud**. Zowel de zender als de ontvanger zullen dezelfde baud-snelheid moeten gebruiken. Er is voor onze MSX computer keuze uit de snelheden 50, 75, 110, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600 of 19200 baud. De meeste printers hebben een meer beperkte keuzemogelijkheid. Veel gebruikte snelheden zijn 9600 voor verbindingen zonder modems en 75, 300, 1200 of 2400 bij modemverbindingen.

Voor de kodering van afzonderlijke tekens wordt meestal de ASCII kode gebruikt. Er zijn slechts 128 verschillende ASCII kodes mogelijk. Die 128 verschillende kodes zijn met 7 bitjes te koderen. Het is echter ook mogelijk om de uitgebreide ASCII kode set te gebruiken, waarbij 256 verschillende kodes bestaan. In dat geval zal gebruik moeten worden gemaakt van 8 bitjes per kode. Het aantal bitjes per kode kan zowel op de computer als op de meeste printers worden ingesteld. Op de printer wordt dat meestal door middel van een schakelaartje gedaan. Op de computer wordt dat met behulp van een BASIC-statement gedaan.

We weten nu dat ieder teken, of dit nu een zeven-bits kode is of een acht-bits kode, wordt voorafgegaan door een startbit. Om de ontvanger

in staat te stellen het begin van een volgend teken te herkennen, wordt ieder teken door de zender beëindigd met een *stopbit*. Dit bitje maakt de lijn "hoog". Hierdoor zal de ontvanger het begin van het startbit ("laag") van het volgende teken direkt herkennen. Het aantal stopbitjes is instelbaar. Op onze computer kunnen we kiezen uit 1, 1,5 of 2 stopbits. We zullen hetzelfde aantal moeten kiezen waarop onze printer rekent.

Tussen het laatste databit en het stopbit wordt door de zender nog een bitje tussengevoegd. Dit tussengevoegde bitje heeft tot doel dat de ontvanger kan kontroleren of deze alle databitjes goed heeft ontvangen. Dit bitje wordt het *"pariteitsbit"* genoemd. Op onze computer hebben we de keuze uit geen pariteit, even pariteit of oneven pariteit. Welke keuze u maakt is niet zo belangrijk, zolang aan beide zijden maar dezelfde pariteit is gekozen. Kiest u voor geen pariteit, dan is er natuurlijk geen kontrole op korrektheid mogelijk.

De opbouw van een teken, zoals hiervoor beschreven, kan als volgt worden weergegeven:

start	1	2	3	4	5	6	7	E	stop

Hierin is gekozen voor een zeven-bits kode met even pariteit en een stopbit. Bij elkaar zijn dus 10 bitjes nodig om een 7-bits kode over te sturen.

10.2 Een seriële verbinding zonder modems

Laten we er eens vanuit gaan, dat we een verbinding tussen een computer en een printer hebben (in de plaats van een printer mag u ook een andere computer denken). De printer staat normaal gesproken na het aanschakelen gereed om gegevens, die door de computer op de seriële interface worden gezet, te lezen en vervolgens af te drukken.

Die gegevens zullen met een vaste snelheid moeten worden verzonden. Bovendien moeten zowel de computer als de printer van te voren weten op welke snelheid de gegevens worden verzonden. Stel dat de com-

puter de bitjes van een teken twee keer zo snel verstuurt als de printer denkt dat ze zullen komen. De printer zal dan alleen het eerste, derde, vijfde en zevende bitje zien en de overige bitjes missen. Het zal duidelijk zijn, dat er van die vier ontvangen bitjes geen zinnig teken te maken is. Zowel de computer als de printer moeten dus precies op dezelfde snelheid worden ingesteld. Die snelheid wordt aangegeven met *baud*; er wordt dan ook meestal van *baud-snelheid* gesproken.

Bij een snelheid van 9600 baud (zonder gebruik van modems) kunnen 9600 bitjes per sekonde worden overgestuurd. Zijn beide zijden van de interface op dezelfde snelheid ingesteld, dan mag de computer er vanuit gaan dat de printer de ontvangen gegevens korrekt ontvangt. De meeste printers drukken tegenwoordig wel af met een snelheid van meer dan 120 regels per minuut. Dit is twee regels per sekonde. Stellen we nu, dat een regel 80 tekens bevat, dan drukt de printer dus 2 keer 80 is 160 tekens per sekonde af. Stel dat ieder teken uit tien bitjes bestaat, dan drukt de printer dus af met een snelheid van 1600 bitjes per sekonde. U ziet, dat dit veel langzamer is dan de snelheid waarmee de gegevens over de lijn worden gestuurd. Gelukkig hebben de meeste printers daarom tegenwoordig een *buffergeheugen*, waarin meestal wel enkele regels tijdelijk kunnen worden opgeslagen. Maar wat gebeurt er als ook dat buffer vol is, of wanneer de printer helemaal geen buffer heeft?

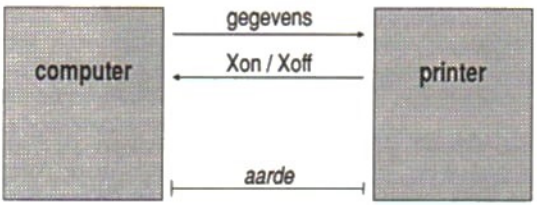
De computer mag alleen gegevens naar de printer sturen wanneer deze aangeeft dat hij in staat is om gegevens te ontvangen. De printer kan dit niet aangeven langs dezelfde lijn waarover de computer zijn gegevens verstuurd. Dan zouden immers zowel de computer als de printer signalen op dezelfde lijn zetten en daarmee elkaars signalen verstoren. De printer heeft twee mogelijkheden om aan de computer kenbaar te maken dat hij in staat is een volgend teken te ontvangen.

Voor de eerste mogelijkheid dient de printer zelf ook te kunnen zenden (dus niet alleen gegevens van de computer ontvangen). De printer kan dan een bepaalde kode naar de computer sturen waarmee de computer wordt gevraagd om even te wachten met het versturen van een volgend teken. Zodra de printer weer een teken kan ontvangen zal hij een andere kode naar de computer sturen. De twee kodes die hier worden bedoeld zijn **Xon** en **Xoff**.

Na het aanschakelen van computer en printer gaat de computer er vanuit, dat de printer niet in staat is gegevens te ontvangen. Pas wan-

neer de printer de kode *Xon* (hexadecimaal &H11) heeft verstuurd, begint de computer gegevens naar de printer te sturen. De computer blijft net zolang gegevens sturen totdat de kode *Xoff* (hexadecimaal &H13) van de printer wordt ontvangen. Nu wacht de computer weer met het versturen van gegevens totdat de printer met een *Xon* kode aangeeft dat er weer nieuwe gegevens kunnen worden afgedrukt.

Bij gebruik van deze methode zijn in principe slechts drie draden op de seriële interface nodig; Een draadje waarover de computer gegevens verstuurt die door de printer worden ontvangen, een draadje waarover de printer Xon/Xoff kodes verstuurt die door de computer worden ontvangen, en een retour-draad (gemeenschappelijke massadraad). Het principe ziet u hierna:



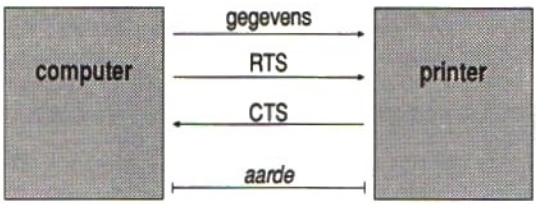
Niet alle printers zijn in staat om gegevens (Xon/Xoff kodes) te versturen. Printers die dat niet kunnen zullen de computer op een andere manier moeten laten weten dat ze tijdelijk even geen nieuwe gegevens kunnen verwerken. Die andere manier is het zogenaamde **ready/busy protokol**. In dit protokol zijn naast de zend- en ontvangstdraden nog een aantal draadjes nodig op de seriële interface.

Wanneer de computer (onder het ready/busy protokol) gegevens naar de printer wil versturen, dan zal deze eerst het RTS signaal aktiveren. Door **RTS** *(Ready To Send)* te aktiveren zegt de computer tegen de printer: "Ik heb gegevens voor u, bent u gereed om ze te ontvangen?". Wanneer de printer inderdaad gereed is gegevens te ontvangen, dan zal deze het signaal **CTS** *(Clear To Send)* aktiveren. Hiermee zegt de printer tegen de computer: "Ja, ik ben klaar voor ontvangst, stuur gegevens."

Zodra de buffer van de printer nu volloopt, zal de printer het signaal CTS de-aktiveren. De computer weet dan, dat de printer tijdelijk even

geen gegevens kan ontvangen en stopt verdere verzending van gegevens totdat de printer het signaal CTS weer aktiveert. Dan worden de overgebleven gegevens alsnog verstuurd. Zodra de computer geen gegevens meer te versturen heeft, de-aktiveert hij het signaal RTS. De printer weet dan dat er geen gegevens meer te verwachten zijn en de-aktiveert zelf het signaal CTS.

De signalen RTS en CTS zijn draadjes op de seriële interface. Naast de zenddraad zijn dus nog twee draden nodig, één voor RTS en één voor CTS. Onder het ready/busy protokol zijn dus vier draadjes nodig.



Bij de behandeling van de BASIC-statements zullen we de hiervoor beschreven begrippen weer tegenkomen. We moeten onze computer namelijk vertellen welk protokol hij moet gebruiken. We zullen dan hetzelfde protokol moeten gebruiken dat ook door de printer wordt gebruikt. Sommige printers kennen beide protokollen, zodat uit één van beide gekozen kan worden. De meeste printers kennen slechts één van beide protokollen. Er is dan geen keuze mogelijk op de printer en dus zal de computer op de printer moeten worden aangepast.

10.3 Een seriële verbinding met modems

Wanneer we een verbinding willen maken tussen onze computer en de computer van een vriend die in een andere stad woont, dan zullen we gebruik willen maken van het openbare telefoonnet. Via dat telefoonnet kunnen we in feite verbindingen leggen tussen iedere gewenste plaats in de gehele wereld.

Een telefoonlijn is echter alleen geschikt voor het overdragen van stemgeluiden. Digitale signalen van de computer zouden niet aan de andere kant van de lijn aankomen. Om dit probleem op te lossen is het modem ontwikkeld. Het modem maakt van de digitale signalen van de

computer *analoge* signalen (dit wordt *moduleren* genoemd), die dezelfde eigenschappen hebben als stemgeluid. Dit analoge signaal wordt via de telefoonlijn naar de andere computer gestuurd. Die andere computer is echter niet in staat analoge signalen te verwerken. Vandaar dat tussen de telefoonlijn en de computer weer een modem dient te worden geschakeld. Dat modem maakt van de ontvangen analoge signalen weer *digitale* signalen (dit wordt *demoduleren* genoemd).

In feite maakt het, zoals u uit voorgaande kunt afleiden, voor de computer niets uit of de computers direkt met elkaar zijn verbonden of dat er modems en een telefoonnetwerk tussen beide computers in zitten. Hiermee zou deze paragraaf ten einde kunnen zijn, ware het niet dat er toch nog enkele probleempjes op te lossen zijn. Wat denkt u dat er zou gebeuren, wanneer we vergeten de stekker van het modem in het stopkontakt te steken. Precies, het digitale computer signaal wordt niet omgezet naar analoog en er wordt geen analoog signaal over de telefoonlijn naar de andere computer gestuurd. Met andere woorden, we kunnen zoveel tekens naar het modem sturen als we willen, maar er komt geen enkel teken over de telefoonlijn.

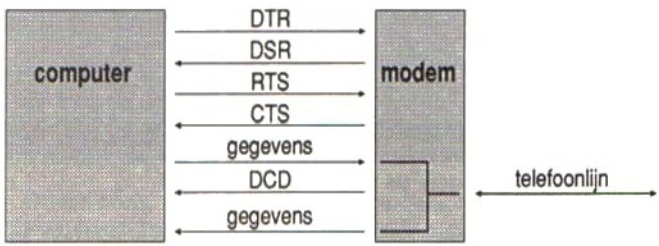
De computer zal daarom, voordat er tekens naar het modem worden gestuurd, eerst moeten kontroleren of het modem gereed is om tekens te moduleren en te versturen over de lijn. Daartoe heeft de computer een speciale verbinding met het modem. Die verbinding heeft de naam **DTR** *(Data Terminal Ready)*. Door dit signaal te aktiveren vertelt de computer het modem dat de computer gereed is om gegevens te versturen c.q. te ontvangen. Het modem zal, indien de spanning is aangeschakeld, hierop antwoorden met een ander signaal, en wel met **DSR** *(Data Set Ready)*. Hiermee geeft het modem aan de computer te kennen dat het aangeschakeld en "stand-by" is.

Wanneer nu de computer gegevens via het modem (dat stand-by is) wil versturen, dient deze net als in het ready/busy protokol het signaal RTS te aktiveren. Het modem zal na een korte vertraging dan het signaal CTS teruggeven, waarna de computer mag beginnen met het versturen van gegevens. De vertraging is nodig, omdat het modem eerst een draaggolf moet opbouwen. De digitale computersignalen worden dan als analoog signaal door het modem op die draaggolf gemoduleerd.

Tot nu toe hebben we alleen gekeken wat er gebeurt bij het verzenden van gegevens. Laten we nu eens kijken hoe het ontvangen van

gegevens in zijn werk gaat. De computer moet allereerst het modem in de stand-by stand zetten, door het signaal DTR naar het modem te sturen. Het modem zal aangeven dat het stand-by is door het signaal DSR te aktiveren. Zodra het modem een draaggolf op de telefoonlijn ziet, zal deze het signaal **DCD** *(Data Carrier Detected)* aktiveren. De ontvangende computer weet nu, dat er gegevens zullen worden ontvangen, die door de computer aan de andere kant van de lijn worden verzonden.

De tot nu toe besproken signalen zijn in de volgende tekening nog eens samengevat.



Er zijn nog meer signalen aanwezig op de RS232C interface tussen de computer en het modem. We zullen ze hier verder niet behandelen. Mocht in de volgende paragraaf bij de behandeling van de BASIC-statements uitleg van één van die andere signalen nodig zijn, dan zal die uitleg bij dat statement worden gegeven.

10.4 RS232C BASIC statements

De BASIC statements voor het besturen van de RS232C interface zitten allemaal in een ROM in de interface. De interface wordt in een extensie-slot van de MSX computer gestoken (terwijl de spanning van de computer uitgeschakeld is). Wordt hierna de computer aangeschakeld, dan kunnen de BASIC statements uit het RS232C ROM worden gebruikt. Alle statements beginnen met de CALL statement en worden gevolgd door het individuele RS232C-statement.

Het initialiseren van de RS232C interface gebeurt met het COMINI statement. Dit COMINI statement wordt als parameter meegegeven aan

het CALL statement (net als bij het u al bekende CALL FORMAT statement). Het volledige statement luidt dus: CALL COMINI.

Voordat we een RS232C poort openen, moeten we deze eerst initialiseren. We zullen nu eerst zien hoe we een poort kunnen initialiseren en vervolgens hoe we een poort openen.

Het initialiseren van een poort wordt met het CALL COMINI statement gedaan. Het volledige formaat van dit statement is als volgt:

```
CALL COMINI [([<string>][,[<RxBd>][,[<TxBd>][,[<Tout>]]]])]
```

string	diverse instellingen (zie hierna)
RxBd	baudrate voor ontvanger
TBd	baudrate voor zender
Tout	time out (waarna foutkonditie staat)

Baudrate:
Dit is de overdrachtsnelheid, de we eerder 'baudsnelheid' noemden. Deze snelheid mag zowel voor het zenden als voor het ontvangen worden opgegeven. Indien alleen de snelheid voor de ontvanger wordt gespecificeerd, dan zal de zender automatisch dezelfde snelheid krijgen. Indien helemaal geen baudrate wordt opgegeven, dan krijgen zowel de ontvanger als de zender automatisch de instelling van 1200 baud. Geeft u wel een snelheid op, dan is er keuze uit de volgende snelheden:

```
50  110   600  1800  2400  4800   9600
75  300  1200  2000  3600  7200  19200
```

Time out:
Wanneer de computer aan het modem te kennen heeft gegeven (door middel van RTS) dat deze iets wil versturen, en het modem geeft niet binnen de tijd die in Time out is aangegeven, het signaal CTS terug, dan wordt een foutkonditie gegenereerd. Het modem zou bijvoorbeeld uitgeschakeld kunnen zijn. Indien hier de waarde 0 wordt ingevuld, zal er geen Time out foutkonditie ontstaan. De computer blijft dan aanhoudend wachten, zonder dit te melden.

String:
De string wordt samengesteld uit een groot aantal parameters, en wel als volgt:

"[*n*:][D[P[S[X[C[ILF[SLF[SIO]]]]]]]]"

n: Poortnummer van de te initialiseren seriële poort. Wanneer deze parameter wordt weggelaten, wordt poortnummer 0 gekozen. U mag ook poort nummers 1 en 2 gebruiken. In dat geval dient uw computer natuurlijk wel meerdere poorten te hebben.

D Lengte (in bits) van de over te dragen data. Er is keuze uit 5, 6, 7 of 8 bits kodes.

P Pariteit. Er is keuze uit:
- E Even pariteit
- O Oneven pariteit
- I Niet op pariteit kontroleren, al zit er wel een pariteitsbit in de overgezonden kode
- N Geen pariteit

S Het aantal stopbits. De volgende keuzes kunnen worden gemaakt:
- 1 1 stopbit
- 2 1,5 stopbits
- 3 2 stopbits

X Geeft aan of er gebruik van het Xon/Xoff protokol moet worden gemaakt. De volgende keuzes zijn mogelijk:
- X Xon/Xoff protokol gebruiken
- N Xon/Xoff protokol niet gebruiken

C Geeft aan of het ready/busy protokol (RTS/CTS) moet worden gebruikt. De volgende mogelijkheden kunt u kiezen:
- H Handshake volgens RTS/CTS protokol wordt gebruikt
- N Er wordt geen handshake gebruikt

ILF Insert Line Feed na ontvangst van een Carriage Return kode. U heeft de volgende keuzemogelijkheden:
- A Er wordt een LF-kode achter een CR toegevoegd
- N Er wordt niets toegevoegd

SLF Verstuur een toegevoegde Line Feed kode na een verzonden Carriage Return kode. Uw keuzemogelijkheden zijn:
- A Er wordt een LF-kode toegevoegd na iedere CR
- N Er wordt niets toegevoegd

SIO Met Shift Out en Shift In kan in worden aangegeven dat kodes, die volgen op Shift Out een andere betekenis hebben dan hun standaard (ASCII) betekenis. Met Shift In wordt aangegeven dat alle daaropvolgende kodes weer hun standaard betekenis hebben. Op die manier kan de tekenset worden uitgebreid. Of van deze uitbreidingsmogelijkheid (geldt niet voor 8 bits kodes) wel of niet gebruik wordt gemaakt, kan met deze parameter worden aangegeven. De mogelijke instellingen zijn:
S SI/SO mogelijk
N SI/SO niet mogelijk

Wanneer we het CALL COMINI statement zonder enige parameters programmeren, dan worden automatisch voorkeurswaarden toegekend en wel als volgt:

```
CALL COMINI
```

is hetzelfde als:

```
CALL COMINI ("0:8E3XHNNN",1200,1200,0)
```

Dit wil zeggen dat RS232C-poort 0 met 8 bits kodes plus een Even pariteitsbit en twee stopbits wordt geinitialiseerd. Het Xon/Xoff protokol wordt gebruikt, CTS/RTS handshake wordt gebruikt, er worden geen LF-kodes toegevoegd bij ontvangen en zenden van CR-kodes en de SI/SO optie is niet mogelijk. Dit alles staat in de string. We zien bovendien, dat de snelheid van de overdracht voor zowel ontvangst als voor zenden 1200 baud is en dat er niet op een Time out konditie wordt gekontroleerd.

Stel nu, dat we diezelfde poort willen initialiseren met 7-bits tekenkodes in plaats van 8-bits en dat we het Xon/Xoff protokol niet willen gebruiken. Voor het overige moet alles blijven zoals in het vorige voorbeeld. Hoe ziet het statement waarmee die wijziging kan worden bewerkstelligd er dan uit?

```
CALL COMINI ("7E3N")
```

Alles wat we niet specificeren krijgt automatisch de voorkeurswaarden. Zouden we bijvoorbeeld alleen de zendsnelheid willen veranderen, dan zou het statement er als volgt uit zien:

```
CALL COMINI (,,1200)
```

U zou nu in staat moeten zijn iedere gewenste initiële instelling van de RS232C poort met verstand van zaken te verrichten. Houdt u daarbij echter vooral goed rekening met de instelling van de computer aan de andere kant van de lijn. Indien beide computers niet dezelfde instelling hebben, zal er van kommunikatie tussen beide computers niets terecht komen.

Weet u niet precies meer hoe het formaat van het COMINI statement is en welke parameters u allemaal mag specificeren, dan kunt u het statement invoeren:

```
CALL COMHELP [(<kanaalnummer>:)]
```

Hierop verschijnt op uw beeldscherm de volgende tekst:

```
Initialize statement options

CALL COMINI ("
<Device# {0,1,2...9}>:
<Character length {5,6,7,8}>
<Parity {E,O,I,N}>
<Stopbits {1,2,3}>
<XON/XOFF {X,N}>
<CTS handshaking {H,N}>
<Auto LF on receive {A,N}>
<Auto LF on transmit {A,N}>
<SI/SO {S,N}>"
,<Receiver baud rate>
,<Transmitter baud rate>
,<Time out count>
                          )
Default:
CALL COMINI("0:8N1XHNNN"
     ,1200,1200,0)
```

U kunt nu de toegestane waarden in de juiste volgorde volgens het juiste formaat intikken, waarna de poort zal worden geïnitialiseerd.

Is de RS232C poort eenmaal geïnitialiseerd, dan dient hij te worden geopend, voordat er gegevens over kunnen worden verstuurd of van

ontvangen. Het openen gaat op dezelfde manier als het openen van een bestand, met een OPEN statement. Het formaat van het OPEN statement, waarmee een RS232C poort wordt geopend is als volgt:

```
OPEN "COM[n]:" [FOR <mode>] AS [#]<filenummer>
```

Het enig nieuwe hierin is het te openen apparaat. Het apparaat is COM en mag worden gevolgd door een nummer (*n*). Indien geen nummer wordt ingevuld, zal nummer 0 worden aangenomen.

De mode kan worden weggelaten. In dat geval zal de mode I/O zijn. Er kan dan dus zowel worden gezonden als ontvangen. In deze mode wordt niet gekontroleerd op het bereiken van het einde van het over te sturen bestand. Hierdoor is deze mode zeer geschikt om uw computer met die van een kennis te laten kommuniceren, waarbij de te versturen tekens rechtstreeks van het toetsenbord naar de RS232C poort worden gestuurd en de ontvangen tekens rechtstreeks van de poort naar het beeldscherm.

Over een geïnitialiseerde en geopende poort kan worden gezonden en ontvangen. Zenden kan met de statements:

```
PRINT #<filenummer>, ...
PRINT #<filenummer> USING "<def>", ...
```

Ontvangen van gegevens via de RS232C poort kan met de statements:

```
INPUT #<filenummer>,<variabele>
LINE INPUT #<filenummer>,<string-variabele>
I$=INPUT$(<aantal>,#<filenummer>)
```

Komt *filenummer* overeen met het filenummer dat werd gebruikt in het OPEN statement waarmee de RS232C poort werd geopend, dan zullen de PRINT statements het zenden en de INPUT statements het ontvangen van gegevens tot gevolg hebben.

Het volgende voorbeeld laat zien hoe gegevens kunnen worden verzonden:

```
1000 CALL COMINI ("",300,300,0)
1010 OPEN "COM:" AS #1
1020 I$=INKEY$:IF I$="" THEN GOTO 1020
1030 PRINT #1,I$;
1040 GOTO 1020
```

U ziet dat de poort wordt geïnitialiseerd met snelheden voor zenden en ontvangen van 300 baud. Vervolgens wordt de poort geopend voor I/O (omdat er geen mode is gespecificeerd). Daarna wordt de eindeloze lus van regels 1020 t/m 1040 gestart. Daarin wordt steeds een teken opgevraagd met regel 1020, dat wordt verzonden met regel 1030. Hierna kan het volgende teken worden opgevraagd en verzonden.

Zolang we in ons programma bezig zijn gegevens te verzenden, kunnen we geen gegevens ontvangen. Immers onze computer kan slechts één statement tegelijk uitvoeren. Zou de computer aan de andere kant van de lijn, terwijl wij aan het zenden zijn, ook gegevens versturen, dan zal onze computer moeten overschakelen van zenden op ontvangen. Deze mogelijkheid wordt geboden door het statement CALL COMON. Het formaat van dit statement is:

```
CALL COMON ("[<kanaalnummer>:]")
```

Dit statement zorgt er voor dat het systeem na uitvoering van ieder BASIC statement even kontroleert of er een teken is ontvangen via de met kanaalnummer aangegeven seriële poort. Indien geen kanaalnummer wordt gespecificeerd, zal nummer 0 worden gekozen. Zodra er een teken is ontvangen zal er naar de subroutine worden gesprongen, die is aangegeven in het statement CALL COM. Het formaat van dit statement is:

```
CALL COM ([<kanaalnummer>:],GOSUB <regelnummer>)
```

De subroutine vanaf de met *regelnummer* aangegeven regel dient een ontvangstroutine te bevatten. Na ontvangst van het teken moet onmiddellijk worden teruggekeerd naar de zendroutine, waarna we weer een volgend teken kunnen intikken en naar de andere kant versturen. Het volgende voorbeeld laat zien hoe een programma er uitziet, dat zowel kan zenden als ontvangen.

```
1000 CALL COMINI ("",300,300,0)
1010 OPEN "COM:" AS #1
1020 CALL COM (,GOSUB 1100)
1030 CALL COMON ("")
1040 Z$=INKEY$: IF Z$="" THEN GOTO 1040
1050 PRINT #1,Z$;
1060 GOTO 1040
1100 O$=INPUT$(1,#1)
1110 PRINT O$;
1120 RETURN
```

Met regel 1020 wordt aangegeven naar welke subroutine moet worden gesprongen na detektie van een ontvangen teken. Met regel 1030 wordt ervoor gezorgd, dat het systeem de ontvangst van een teken langs de seriële poort detekteert. Met de regels 1040 t/m 1060 kunnen tekens worden ingetoetst en verzonden over de seriële poort. Mocht er op enig moment een teken van de andere kant worden ontvangen, dan wordt dit pas gedetekteerd nadat het statement, dat op dat moment in uitvoering is, wordt beëindigd. Pas dan wordt naar de in regel 1020 aangegeven subroutine gesprongen. In die subroutine wordt het teken van de seriële poort gelezen en op het beeldscherm afgedrukt, waarna wordt teruggekeerd naar de plaats waar het programma zojuist werd afgebroken.

Het is belangrijk om goed te beseffen dat het detekteren van de ontvangst van een teken pas wordt gekonstateerd nadat er een statement is uitgevoerd. Zou er op een gegeven moment een INPUT statement in uitvoering zijn, dan eindigt dat statement pas nadat we één of meer tekens plus de Return-toets hebben ingedrukt. Dat duurt (zelfs voor een snelle typist) zo lang, dat er al een hele reeks tekens op de seriële poort kunnen zijn ontvangen. Wanneer we dan eindelijk de seriële poort gaan uitlezen, kunnen we een aantal tekens missen. Er is weliswaar een speciaal buffer waarin de ontvangen tekens tijdelijk worden opgeslagen, maar dat buffer is slechts 255 tekens groot. Bij hoge transmissiesnelheden loopt dat buffer binnen een sekonde vol. Daarom is in het voorbeeld gekozen voor invoer met behulp van de INKEY-funktie. Deze wordt snel uitgevoerd, ook als er niets wordt ingetikt.

Er is echter nóg een goede reden om gebruik te maken van de INPUT$ funktie. Met deze funktie kunnen alle tekens worden gelezen, ook kontrole tekens. Een INPUT statement is het slechtst toepasbaar, omdat dit statement eindigt door ontvangst van een komma of een Carriage

Return teken. Ook het LINE INPUT statement is minder goed bruikbaar, omdat dat statement eindigt met ontvangst van een Carriage Return teken.

Net zoals het detekteren van de ontvangst van een teken via de seriële interface kan worden aangeschakeld met COMON, zo kan het detekteren weer worden uitgeschakeld met het statement CALL COMOFF. Het formaat van dit statement is:

```
CALL COMOFF ("[<kanaalnummer>:]")
```

Ook is het nog mogelijk om wel te blijven kontroleren of er een teken over de seriële interface is ontvangen, maar daar niet op te reageren. Dit wordt met het CALL COMSTOP statement gedaan. Het formaat is:

```
CALL COMSTOP ("[<kanaalnummer>:]")
```

Is na uitvoering van CALL COMSTOP een teken over de seriële interface ontvangen, dan gebeurt er niets. Zodra echter een CALL COMON statement wordt uitgevoerd, zal alsnog worden gesprongen naar de subroutine die in het CALL COM statement was aangegeven.

Wanneer de computer aan de andere kant van de lijn gegevens verstuurt en wij willen het zenden van de andere computer onderbreken, dan kunnen we een aantal *Break*-tekens sturen. Wanneer de computer aan de andere kant nu deze tekens ontvangt, en na ontvangst de status van zijn seriële poort kontroleert, zal deze ontdekken dat er Break-tekens zijn ontvangen en stoppen met het verzenden van nog meer tekens. Het statement waarmee wij Break-tekens kunnen versturen ziet er als volgt uit:

```
CALL COMBREAK (["<kanaalnummer>:"],<aantal>)
```

Hiermee wordt het opgegeven aantal Break-tekens verzonden over de met kanaalnummer aangegeven seriële poort. Dit statement werkt alleen wanneer de seriële poort is geopend.

Wanneer we een aantal tekens hebben ontvangen, willen we graag weten of die tekens allemaal korrekt zijn ontvangen. Daartoe kunnen we de status van de seriële poort opvragen met het volgende statement:

```
CALL COMSTAT (["<kanaalnummer>:"],<numerieke var.>)
```

Hierdoor wordt de status van de met kanaalnummer aangegeven seriële poort door de hardware in de gespecificeerde numerieke variabele gezet. Deze status staat in twee bytes ofwel 16 bits. Ieder bit heeft een bepaalde betekenis. Hierna volgt de betekenis van ieder bit.

bit	betekenis (indien bit = 1)
15	Buffer Overflow Error
14	Time Out Error
13	Framing Error
12	Over Run Error
11	Parity Error
10	Break toets ingedrukt
9	(gereserveerd)
8	(gereserveerd)
7	Clear To Send (CTS)
6	Timer/Counter Output
5	(gereserveerd)
4	(gereserveerd)
3	Data Set Ready (DSR)
2	Break teken gedetekteerd
1	Ring Indicator
0	Data Carrier Detected (DCD)

Een aantal van de bits die in de voorgaande theorie nog niet werd toegelicht, zullen we nu aan een nader onderzoek onderwerpen.

Buffer Overflow
Om te voorkomen, dat we tijdens het ontvangen van gegevens via de seriële poort tekens missen, worden alle ontvangen tekens in een buffertje gezet. Deze buffer is 255 tekens groot. Wanneer we met ons BASIC programma de tekens langzamer van de poort lezen dan ze door de computer naar onze seriële poort worden gezonden, dan zal de buffer langzaam maar zeker vollopen. Is de buffer vol en wordt er dan nóg een teken ontvangen, dan loopt de buffer over. Dit levert de conditie Buffer Overflow op.

Framing Error
De RS232C interface kontroleert bij ontvangst van een teken of dat teken uit het juiste aantal bits bestaat. Het geïnitialiseerde aantal start-

en stopbits en het eventuele pariteitsbit worden van het ontvangen teken afgekapt. De overblijvende bits moeten het vereiste aantal zijn. Is dit niet het geval dan wordt de konditie Framing Error gezet.

Data Set Ready (DSR)
Zoals in een voorgaande paragraaf al werd uitgelegd, wordt dit signaal door het modem gegenereerd als antwoord op het signaal DTR, dat door onze computer moet worden gegenereerd. Onze computer kan DTR aktiveren door het statement:

```
CALL COMDTR (["<kanaalnummer>:"],<uitdrukking>)
```

Indien de uitdrukking een waarde 0 oplevert, zal het DTR signaal gedéaktiveerd worden. Een waarde die ongelijk is aan 0 zal het DTR signaal aktiveren.

Ring Indicator
Dit is een signaal dat door een daartoe ingericht modem kan worden gegenereerd. Het modem zal dit signaal aktiveren op het moment dat de telefoonlijn waarop deze is aangesloten, wordt gebeld.

Het laatste nieuwe statement, dat we nog moeten behandelen is

```
CALL COMTERM [("<kanaalnummer>:")]
```

Door dit statement uit te voeren gaat onze computer zich als een terminal gedragen. Het aangegeven kanaal mag niet geopend zijn. Na uitvoering van dit statement krijgen een aantal funktietoetsen een speciale funktie, en wel als volgt:

F6 *Teken-mode aan/uit*
Na aanschakelen staat de tekenmode uit. Wanneer de tekenmode aan staat, worden kontrole tekens afgedrukt als een letter. Daartoe wordt bij de kode van het kontroleteken een waarde 64 opgeteld. De kode 02 zal dan resulteren in de kode 64+2=66. Dit is de kode voor de letter B. Deze letter zal dan ook worden afgedrukt.

F7 *Half/Full duplex*
Na aanschakelen staat full duplex ingesteld. In full duplex mode mag de computer zenden en ontvangen tegelijk. In half duplex mode mag de computer niet tegelijk ontvangen en zenden. In deze

mode worden alle tekens die naar de seriële interface worden gezonden tevens naar het beeldscherm gezonden.

F8 *Printer echo aan/uit*
Na aanschakelen staat de printer echo uit. Wanneer de printer echo aan staat, worden alle tekens die naar het beeldscherm worden gestuurd tevens naar de printer gestuurd.

Naast de tot nu toe behandelde nieuwe statements zijn er ook nog een aantal bestaande statements en funkties, die met de RS232C interface een meer uitgebreide of gewijzigde funktie hebben gekregen. Het gaat hier om de statements SAVE, LOAD, MERGE en RUN en de funkties EOF, LOC en LOF.

Wanneer u een BASIC programma in het geheugen van uw MSX computer heeft staan, en u geeft het direkte kommando

```
SAVE "COM[<kanaalnummer>:]"[,A]
```

dan wordt dat programma over de seriële interface verzonden als een ASCII-bestand. Het maakt daarbij niet uit of de optionele parameter ,A is opgegeven of niet. Het SAVE kommando is goed te gebruiken in kombinatie met het

```
LOAD "COM[<kanaalnummer>:]"[,R]
```

wanneer dit LOAD kommando op de computer aan de andere kant van de kommunikatielijn wordt gegeven. Het via de seriële poort geladen programma wordt in het geheugen gezet, een eventueel nog in dat geheugen staand programma wordt gewist en alle openstaande bestanden worden gesloten (dus eigenlijk precies hetzelfde als bij het laden van programma's van schijf). Wordt de ,R optie aan het statement toegevoegd, dan blijven alle bestanden geopend en wordt het programma na over de seriële poort te zijn ontvangen automatisch gestart. Een programma dat over de seriële poort wordt verzonden moet eindigen met de ^Z kode, omdat deze kode het einde van het programmabestand aangeeft voor het ontvangende station. Dit geldt ook voor het kommando

```
MERGE "COM[<kanaalnummer>:]"
```

Het over de seriële interface ontvangen programma, dat in ASCII formaat dient te zijn, wordt samengevoegd met het reeds in het geheugen van het ontvangende station staande programma. Komen gelijke regelnummers voor, dan wordt de in het geheugen staande programmaregel overschreven door de programmaregel die via de seriële interface wordt ontvangen.

Het RUN kommando heeft het volgende formaat:

```
RUN "COM[<kanaalnummer>:]"[,R]
```

Het programma dat over de seriële interface wordt ontvangen wordt direkt gestart. Indien de optie ,R wordt gespecificeerd, blijven alle openstaande bestanden gewoon geopend. Wordt ,R weggelaten, dan worden alle bestanden gesloten voordat het ingelezen programma wordt gestart.

De funkties EOF, LOC en LOF hebben een ongewijzigd formaat. Wanneer ze echter gebruikt worden voor een filenummer dat gekoppeld is aan een kommunikatiekanaal, dan hebben ze de hierna volgende funkties.

```
EOF (<filenummer>)
```

kijkt of het via de seriële interface ontvangen teken een *end-of-file* teken is. Deze funktie levert de waarde –1 op indien het end-of-file teken is ontvangen en blijft 0 indien dit niet het geval is.

Zoals eerder in deze paragraaf al werd opgemerkt, worden de via de seriële interface ontvangen tekens tijdelijk opgeslagen in het 255 bytes grote kommunikatiebuffer. De funktie

```
LOC (<filenummer>)
```

levert het aantal in het kommunikatiebuffer staande tekens op. Dit is het tegenovergestelde van het resultaat dat de funktie

```
LOF (<filenummer>)
```

oplevert. Met deze funktie wordt namelijk de grootte van het nog vrije deel van de kommunikatiebuffer uitgelezen.

Je zou kunnen beslissen, dat je, elke keer nadat je een teken naar de andere kant hebt gestuurd, even kijkt (met LOC of LOF) hoeveel

tekens er al in de kommunikatiebuffer zijn gezet door de andere kant. Komt dat aantal boven de 128, dan kun je de andere kant vragen tijdelijk te stoppen met het versturen van nog meer tekens, door het verzenden van het Xoff teken. Zodra het buffer is leeggelezen, verstuur je dan het Xon teken, waarna de computer aan de andere kant weer begint te zenden.

10.5 Een programmavoorbeeld

Een aantal van de zojuist besproken statements wordt in het programma aan het einde van deze paragraaf gebruikt. Het is mogelijk om met dit programma verbindingen te maken met een snelheid van 300 baud, zonder dat het kommunikatiebuffer volloopt. Voor hogere snelheden moet u het programma aanpassen. U moet dan bijvoorbeeld tijdelijk stoppen met het versturen van gegevens, om zodoende tijd vrij te maken voor het ontvangen van gegevens op hoge snelheid.

Het programma is geschreven voor gebruik in kombinatie met een *Hayes kompatibel modem*. Dit soort modems is in staat een aantal kommando's uit te voeren. Hierdoor is het bijvoorbeeld mogelijk een telefoonnummer van de computer naar het modem te sturen en het modem opdracht te geven dit nummer zelf te kiezen.

Hayes kompatibele modems kunnen, terwijl ze in de stand-by mode staan, een aantal kommando's interpreteren. Zoals u zich wellicht herinnert komt een modem in de stand-by mode door de spanning van het modem aan te schakelen en door vanuit de computer het signaal DTR te aktiveren. Zodra het modem stand-by is, aktiveert deze het signaal DSR. Vanaf dit moment mag de computer tekens over de "gegevenslijn" naar het modem sturen. Het modem zal de ontvangen gegevens als kommando's interpreteren. Gaat het modem van de stand-by mode naar de zend/ontvangst mode, dan worden tekens die over de gegevenslijn worden verstuurd door het modem gezien als te verzenden tekens.

Alle kommando's die het modem kan interpreteren, beginnen met de letters AT. Een aantal kommando's is speciaal voor het instellen van het modem. Normaal gesproken wordt de standaard instelling van het modem met behulp van de zogenaamde *dip-switches* gedaan. Met de AT-kommando's kunnen die instellingen echter worden gewijzigd.

Met de kommando's ATA en ATO wordt het modem van de stand-by mode naar de zend/ontvangst mode gebracht. Is het modem eenmaal in die stand, dan kan het geen verdere kommando's interpreteren. Alle gegevens die het dan nog ontvangt zullen over de telefoonlijn worden doorgestuurd. Toch is er natuurlijk een mogelijkheid om weer terug te komen in de stand-by mode. Daartoe moet eerst minimaal 1 sekonde niets worden verstuurd. Dan moeten binnen 1 sekonde drie plusjes (+++) worden verstuurd. Hierna moet weer gedurende minimaal 1 sekonde niets verzonden worden. Nu is het modem weer in stand-by mode en kan het weer kommando's interpreteren. Bij het doorlezen van de listing aan het einde van deze paragraaf zult u deze procedure tegenkomen. Het kommando dat daarna in dat programma als eerste wordt gegeven is het ATH kommando. Daarmee schakelt het modem zichzelf van de telefoonlijn af (dit komt overeen met het op de haak leggen van de telefoon).

Een ander kommando dat u in het programma zult tegenkomen is het kommando ATD. Dat kommando mag worden gevolgd door een telefoonnummer. Wanneer het modem dit kommando ontvangt zal deze het opgegeven telefoonnummer kiezen en de verbinding met de computer aan de andere kant opbouwen.

Ik hoop, dat voorgaande uitleg voldoende is om in te zien wat het volgende programma doet. Voor het schrijven van uw eigen kommunikatieprogramma zal het niet voldoende zijn. Dit onderwerp valt echter buiten het kader van dit boek en ik zal er dan ook niet verder op ingaan.

Het volgende programma laat voorbeelden zien van verschillende BASIC-kommando's uit de vorige paragraaf. Het programma werkt goed op transmissiesnelheden tot 300 baud. Ook het ontvangen van gegevens op een snelheid van 1200 baud ging nog goed, al weet ik niet wat er gebeurt wanneer er lange bestanden worden ontvangen. Ik ben bang dat dan de kommunikatiebuffer zal vollopen. Snelheden boven de 1200 baud heb ik het niet getest...

```
1000 '**********************
1010 '*   INITIALISEREN    *
1020 '**********************
1030 H=1
1040 DIM T$(9,1)
1050 RESTORE
```

```
1060 FOR I=0 TO 9
1070 READ T$(I,0):READ T$(I,1)
1080 NEXT I
1090 DATA Fido HCC MSX gg,072-610772,akkermans,123-45678
9,,,,,,,,,,,,,,,,
1100 DIM P$(4,10):RESTORE 1160
1110 FOR I=1 TO 10
1120 FOR J=1 TO 4
1130 READ P$(J,I)
1140 NEXT J
1150 NEXT I
1160 DATA 8,5,6,7,N,E,O,I,1,2,3,#,N,X,#,#,N,H,#,#,N,A,#,
#,N,A,#,#,N,S,#,#,75,300,600,1200,75,300,600,1200
1170 DIM P(1,10):RESTORE 1230
1180 FOR I=1 TO 10
1190 FOR J=0 TO 1
1200 READ P(J,I)
1210 NEXT J
1220 NEXT I
1230 DATA 4,1,4,1,3,1,2,1,2,1,2,1,2,1,2,1,4,2,4,2
1240 SCREEN 0: WIDTH 80: KEY OFF
1250 CALL COM(,GOSUB 1390)
1260 GOSUB 1810
1270 '
1280 '************************
1290 '*     zend routine     *
1300 '************************
1310 I$=INKEY$:IF I$="" THEN GOTO 1310
1320 IF I$=CHR$(24) THEN GOSUB 1450:'SELECT-KEY
1330 PRINT #1,I$;
1340 GOTO 1310
1350 '
1360 '************************
1370 '*   ontvangst routine  *
1380 '************************
1390 PRINT INPUT$(1,1);
1400 RETURN
1410 '
1420 '************************
1430 '*      hoofd menu      *
1440 '************************
1450 PRINT "Kies een funktie:"
1460 PRINT:PRINT "1 instelling van de poort"
```

```
1470 PRINT "2 status van de poort"
1480 PRINT "3 kies telefoonnummer"
1490 PRINT "4 einde kommunikatie"
1500 PRINT
1510 I$=INKEY$:IF I$="" THEN GOTO 1510
1520 IF I$="1" THEN GOTO 1810
1530 IF I$="2" THEN GOTO 1610
1540 IF I$="3" THEN GOTO 2350
1550 IF I$="4" THEN GOTO 2590
1560 GOTO 1510
1570 '
1580 '******************************
1590 '* Status van de RS232-poort  *
1600 '******************************
1610 CALL COMSTAT(,S)
1620 PRINT "Status van de RS232-poort is:"
1630 PRINT
1640 PRINT "Carrier Detect  = ";:IF S AND 1 THEN PRINT "
true" ELSE PRINT "false"
1650 PRINT "Ring Indicator  = ";:IF S AND 2 THEN PRINT "
true" ELSE PRINT "false"
1660 PRINT "Break           = ";:IF S AND 4 THEN PRINT "
detected" ELSE PRINT "not detected"
1670 PRINT "Data Set Ready  = ";:IF S AND 8 THEN PRINT "
true" ELSE PRINT "false"
1680 PRINT "T/C output-2    = ";:IF S AND 64 THEN PRINT
"asserted" ELSE PRINT "negated"
1690 PRINT "Clear To Send   = ";:IF S AND 128 THEN PRINT
"true" ELSE PRINT "false"
1700 PRINT "CTRL/BREAK-key  = ";:IF S AND 1024 THEN PRIN
T "pressed" ELSE PRINT "not pressed"
1710 PRINT "Parity error    = ";:IF S AND 2048 THEN PRIN
T "detected" ELSE PRINT "not detected"
1720 PRINT "Overrun error   = ";:IF S AND 4096 THEN PRIN
T "detected" ELSE PRINT "not detected"
1730 PRINT "Framing error   = ";:IF S AND 8192 THEN PRIN
T "detected" ELSE PRINT "not detected"
1740 PRINT "Time-out error  = ";:IF S AND 16384 THEN PRI
NT "occurred" ELSE PRINT "did not occur"
1750 PRINT "Buffer overflow = ";:IF S AND 16384 THEN PRI
NT "yes" ELSE PRINT "no"
1760 PRINT:PRINT
1770 PRINT "druk op een willekeurige toets om door te ga
```

```
an"
1780 I$=INKEY$:IF I$="" THEN GOTO 1780
1790 I$="":CLS:RETURN
1800 '
1810 '*********************************
1820 '* INSTELLEN VAN DE RS232-POORT *
1830 '*********************************
1840 CLOSE:CLS:PRINT
1850 PRINT "CHARACTER LENGTE       (aantal bits per teke
n)    =";P$(P(1,1),1)
1860 PRINT "PARITEIT              (None, Even, Odd, Ignor
e)    =";P$(P(1,2),2)
1870 PRINT "AANTAL STOP BITS            (1=1, 2=1.5, 3=
2)    =";P$(P(1,3),3)
1880 PRINT "XON/XOFF                 (X=enable, N=disabl
e)    =";P$(P(1,4),4)
1890 PRINT "CTS/RTS HANDSHAKE       (H=handshake, N= non
e)    =";P$(P(1,5),5)
1900 PRINT "AUTO LF ON RECEIVE      (A=added, N=not adde
d)    =";P$(P(1,6),6)
1910 PRINT "AUTO LF ON TRANSMIT    (A=send, N=do not sen
d)    =";P$(P(1,7),7)
1920 PRINT "SHIFT IN/OUT CONTROL     (S=enable, N=disabl
e)    =";P$(P(1,8),8)
1930 PRINT "BAUDRATE RECEIVER         (75, 300, 600, 120
0)    =";P$(P(1,9),9)
1940 PRINT "BAUDRATE TRANSMITTER      (75, 300, 600, 120
0)    =";P$(P(1,10),10)
1950 PRINT "TIME-OUT FOR CTS/XON  (seconds, 0=no time ou
t)    =";TT
1960 LOCATE 0,18:PRINT "     KIES  ITEM  MET CURSOR-OP/N
EER"
1970 PRINT "      KIES WAARDE MET RETURN-TOETS"
1980 PRINT "      ESC-TOETS IS EINDE INSTELLEN"
1990 LOCATE 52,H,1
2000 I$=INKEY$:IF I$="" THEN GOTO 2000
2010 IF I$=CHR$(27) THEN GOTO 2050
2020 IF I$=CHR$(13) THEN GOSUB 2140
2030 IF I$=CHR$(30) OR I$=CHR$(31) THEN GOSUB 2230
2040 GOTO 2000
2050 PA$=""
2060 FOR I=1 TO 8
2070 PA$=PA$+P$(P(1,I),I)
```

```
2080 NEXT I
2090 BR=VAL(P$(P(1,9),9)):BT=VAL(P$(P(1,10),10))
2100 CALL COMINI(PA$,BR,BT,TT)
2110 OPEN "COM:" AS #1
2120 CALL COMON("")
2130 CLS:RETURN
2140 '
2150 '*****************************
2160 '* parameter waarde wijzigen *
2170 '*****************************
2180 IF H=11THENINPUT TT:LOCATE 51,H:PRINT TT;"    ":GOT
O 2210
2190 IF P(0,H)>P(1,H) THEN P(1,H)=P(1,H)+1 ELSE P(1,H)=1
2200 PRINT P$(P(1,H),H);"   ";
2210 LOCATE 52,H,1
2220 RETURN
2230 '
2240 '*****************************
2250 '*  andere parameter kiezen  *
2260 '*****************************
2270 IF I$=CHR$(30) THEN IF H>1 THEN H=H-1 ELSE H=11
2280 IF I$=CHR$(31) THEN IF H<11 THEN H=H+1 ELSE H=1
2290 LOCATE 52,H,1
2300 RETURN
2310 '
2320 '*************************
2330 '* Telefoonnummer kiezen *
2340 '*************************
2350 CLS
2360 FOR I=0 TO 9
2370 PRINT T$(I,0);:PRINTTAB(50);T$(I,1)
2380 NEXT I
2390 PO=0:LOCATE 50,PO,1
2400 I$=INKEY$:IF I$="" THEN GOTO 2400
2410 IF I$=CHR$(13) THEN GOTO 2460
2420 IF I$=CHR$(30) THEN IF PO>0 THEN PO=PO-1 ELSE PO=9
2430 IF I$=CHR$(31) THEN IF PO<9 THEN PO=PO+1 ELSE PO=0
2440 LOCATE 50,PO,1
2450 GOTO 2400
2460 CLS:IF T$(PO,1)="" THEN GOTO 1310
2470 PRINT #1,"ATD";CHR$(&H2C);
2480 FOR I=1 TO LEN(T$(PO,1))
2490 T1$=MID$(T$(PO,1),I,1)
```

```
2500 IF T1$="-" THEN PRINT #1,CHR$(&H2C); ELSE PRINT #1,
T1$;
2510 NEXT I
2520 PRINT #1,
2530 FOR I=1 TO 1000: NEXT I
2540 GOTO 1310
2550 '
2560 '***************************
2570 '* Hoorn op de haak leggen *
2580 '***************************
2590 FOR I=1 TO 1000: NEXT I
2600 PRINT #1,"+++";
2610 FOR I=1 TO 1000: NEXT I
2620 PRINT #1,"ATH"
2630 CLOSE
2640 END
```

APPENDIX A

TYPLES

Het in deze appendix opgenomen programma "Typles" laat goed zien wat de MSX2 grafische mogelijkheden zijn. Aan het plaatje dat op het beeldscherm verschijnt (een gekleurde afbeelding van het toetsenbord) ziet u dat ieder pixel afzonderlijk kan worden gekleurd. Ook ziet u dat de PAINT-kleur verschillend mag zijn van de kleur van de lijnen, die het in te kleuren vlak omgeven.

Het programma leert u volgens een erkende methode typen volgens het tienvingersysteem. Wanneer u alle door het programma opgegeven oefeningen dagelijks doet, dan zult u na enkele weken in staat zijn zo'n 100 aanslagen per minuut te tikken met alle tien uw vingers. Wie zich van het begin af aanwent om niet naar zijn vingers te kijken tijdens het oefenen, die zal na enkele weken zelfs "blind" kunnen typen.

Hoe wordt het programma bediend? Na het opstarten van het programma wordt u naar uw naam gevraagd. Gebruik steeds dezelfde naam. Wanneer u namelijk een oefening met goed gevolg heeft afgewerkt, en u stopt daarna met oefenen, dan wordt uw tot dan toe bereikte resultaat onder uw naam naar schijf geschreven. De volgende keer dat u het programma start, wordt eerst gekeken hoever u was gekomen. U krijgt dan de eerstvolgende oefening.

Op het scherm ziet u het toetsenbord van uw MSX-computer. De toetsen hebben verschillende kleuren. Onder het toetsenbord staat een verklaring van de kleuren. Iedere kleur behoort bij een bepaalde vinger. Went u zich van het begin af aan de toetsen met de aangegeven vingers in te drukken. Alleen dan zult u met tien vingers leren tikken.

Boven het toetsen bord verschijnt een wit vakje. Even later glijden er letters van rechts naar links. De letters verdwijnen in het witte vakje. Steeds wanneer de letters een positie naar links verschuiven hoort u een korte piep. Het is de bedoeling, dat u op elke piep een letter intoetst, en wel de letter die op dat moment in het vakje staat.

Als u een typfout maakt, dan kunt u deze niet korrigeren. U bent aan het typen, hetgeen wil zeggen dat iedere ingetikte letter op papier staat en dus niet meer kan worden hersteld. Dit is geheel in tegenstelling tot tekstverwerking, waarbij iedere typfout kan worden hersteld. Ga na het maken van een fout gewoon door met het intikken van de volgende letters.

Het programma registreert hoeveel fouten u tikt, maar ook hoe snel u typt. Wanneer u precies op iedere piep een letter intikt, zult u een snelheid van iets minder dan 60 aanslagen per minuut halen. Deze snelheid blijft gehandhaafd totdat u alle toetsen heeft leren vinden. Daarna gaat de snelheid, indien u de oefeningen foutloos en binnen de tijd uitvoert, langzaam omhoog.

Maakt u enkele typfouten, dan zult u de oefening nog eens moeten doen. Maakt u te veel typfouten, dan wordt teruggegaan naar de vorige oefening. Hetzelfde geld voor de snelheid. Bent u te langzaam, dan zal de vorige oefening weer worden opgegeven.

Voor iedere oefening wordt getoond welke toetsen bij de oefening zullen worden betrokken. De betrokken toetsen worden even van een witte rand voorzien. Na iedere oefening krijgt u uw score te zien. U ziet dan hoeveel fouten u heeft gemaakt en hoe snel u heeft getypt.

Hier volgt het programma.

```
10 CLEAR 1500
20 ON ERROR GOTO 3630
30 DIM OE$(30)
40 OE$(1)="fdsag"
50 OE$(2)="jkl;h"
60 OE$(3)="fjdksla;gh"
70 OE$(4)="frgt"
80 OE$(5)="juyh"
90 OE$(6)="de"
100 OE$(7)="ki"
110 OE$(8)="sw"
120 OE$(9)="lo"
130 OE$(10)="aq"
140 OE$(11)=";p"
150 OE$(12)="qwert"
160 OE$(13)="poiuy"
```

```
170 OE$(14)="rueiwoqpty"
180 OE$(15)="fvgb"
190 OE$(16)="jmhn"
200 OE$(17)="dc"
210 OE$(18)="k,"
220 OE$(19)="sx"
230 OE$(20)="l."
240 OE$(21)="az"
250 OE$(22)=";/"
260 OE$(23)="zxcvb"
270 OE$(24)="/.,mn"
280 OE$(25)="vmc,x.z/bn"
290 OE$(26)="qwertyuiop;lkjhgfdsazxcvbnm,./"
300 OE$(27)="YUIOPHJKLNM"
310 OE$(28)="QWERTASDFGZXCVB"
320 OE$(29)="12345"
330 OE$(30)="67890"
340 DIM CP(256,2)
350 COLOR 1,7,10
360 SCREEN 1:KEY OFF:WIDTH 30:CLS
370 LOCATE 2,8:PRINT "Hoe heet je";:INPUT N$
380 LOCATE 0,22:PRINT "Is je naam goed gespeld (J/N)?";
390 COLOR 15:LOCATE 0,20:PRINT;:FOR I=1 TO 200:NEXT I
400 COLOR 1:PRINT;:FOR I=1 TO 200:NEXT I
410 I$=INKEY$:IF I$="" THEN GOTO 390
420 IF I$<>"j" AND I$<>"J" AND I$<>"n" AND I$<>"N" THEN
PRINT "Alleen N of J antwoorden":GOTO 410
430 IF I$="j" OR I$="J" THEN CLS:LOCATE 5,22:PRINT"Even
geduld s.v.p":OPEN "a:"+N$+".typ" FOR INPUT AS #1:INPUT#
1,SC:CLOSE#1
440 IF I$="n" OR I$="N" THEN GOTO360
450 CLS:PRINT:PRINT
460 LOCATE 2,3:PRINT N$;","
470 LOCATE 2,6:PRINT "Je was tot les";SC;"gekomen."
480 LOCATE 2,9:PRINT"Met de volgende oefeningen    kun j
e je typ-vaardigheid    verder verhogen."
490 LOCATE 2,22:PRINT"RETURN-toets   =   starten"
500 IF INKEY$<>CHR$(13) THEN GOTO 500
510 OPEN "grp:" AS #1
520 SCREEN 5,2
530 DATA 255,255,192,192,192,192,192,192
540 DATA 192,192,192,192,192,192,255,255
550 DATA 255,255,3,3,3,3,3,3
```

```
560 DATA 3,3,3,3,3,3,255,255
570 SP$=""
580 RESTORE 530
590 FOR I=1 TO 32
600 READ A:LET SP$=SP$+CHR$(A)
610 NEXT I
620 SPRITE$(0)=SP$
630 GOSUB 2000
640 '************************
650 '* Genereer oefentekst *
660 '************************
670 '
680 '***** Bepaal de inhoud
690 TE$=""
700 IF SC>3 THEN GOTO 720
710 O$=OE$(1):L=3:OL=SC+2:GOTO 1140
720 IF SC>7 THEN GOTO 740
730 O$=OE$(2):L=3:OL=SC-2:GOTO 1140
740 IF SC>16 THEN GOTO 760
750 O$=OE$(3):L=4:OL=SC-6:GOTO 1140
760 IF SC>19 THEN GOTO 780
770 O$=OE$(4):L=4:OL=SC-15:GOTO 1140
780 IF SC>22 THEN GOTO 800
790 O$=OE$(5):L=4:OL=SC-18:GOTO 1140
800 IF SC>28 THEN GOTO 820
810 O$=OE$(SC-17):L=3:OL=2:GOTO 1140
820 IF SC>33 THEN GOTO 840
830 O$=OE$(1)+MID$(OE$(12),SC-28,1):L=5:OL=6:GOTO 1140
840 IF SC>38 THEN GOTO 860
850 O$=OE$(2)+MID$(OE$(13),SC-33,1):L=5:OL=6:GOTO 1140
860 IF SC>48 THEN GOTO 880
870 O$=OE$(1)+OE$(2)+MID$(OE$(14),SC-38,1):L=5:OL=11:GOT
O 1140
880 IF SC=49 THEN O$=OE$(1)+OE$(2)+OE$(12):L=5:OL=15:GOT
O 1140
890 IF SC=50 THEN O$=OE$(1)+OE$(2)+OE$(13):L=5:OL=15:GOT
O 1140
900 IF SC=51 THEN O$=OE$(1)+OE$(2)+OE$(12)+OE$(13):L=5:O
L=20:GOTO 1140
910 IF SC>54 THEN GOTO 930
920 O$=OE$(15):L=3:OL=SC-50:GOTO 1140
930 IF SC>57 THEN GOTO 950
940 O$=OE$(16):L=3:OL=SC-53:GOTO 1140
```

```
950 IF SC>63 THEN GOTO 970
960 O$=OE$(SC-41):L=3:OL=2:GOTO1140
970 IFSC>68THENGOTO990
980 O$=OE$(1)+MID$(OE$(23),SC-63,1):L=5:OL=6:GOTO1140
990 IFSC>73THENGOTO1010
1000 O$=OE$(2)+MID$(OE$(24),SC-68,1):L=5:OL=6:GOTO1140
1010 IFSC>83THENGOTO1030
1020 O$=OE$(1)+OE$(2)+MID$(OE$(25),SC-73,1):L=5:OL=11:GO
TO1140
1030 IFSC=84THENO$=OE$(1)+OE$(2)+OE$(23):L=5:OL=15:GOTO1
140
1040 IFSC=85THENO$=OE$(1)+OE$(2)+OE$(24):L=5:OL=15:GOTO1
140
1050 IFSC=86THENO$=OE$(1)+OE$(2)+OE$(23)+OE$(24):L=5:OL=
20:GOTO1140
1060 IFSC=87THENO$=OE$(14)+OE$(3)+OE$(25):L=5:OL=30:GOTO
1140
1070 IFSC=88THENO$=OE$(26)+OE$(27):L=5:OL=41:GOTO1140
1080 IFSC=89THENO$=OE$(26)+OE$(28):L=5:OL=45:GOTO1140
1090 IFSC=90THENO$=OE$(26)+OE$(27)+OE$(28):L=5:OL=56:GOT
O1140
1100 IFSC=91THENO$=OE$(26)+OE$(29):L=5:OL=35:GOTO 1140
1110 IFSC=92THENO$=OE$(26)+OE$(30):L=5:OL=35:GOTO 1140
1120 IFSC=93THENO$=OE$(26)+OE$(29)+OE$(30):L=5:OL=40:GOT
O1140
1130 IFSC=94THENO$=OE$(26)+OE$(27)+OE$(28)+OE$(29)+OE$(3
0):L=5:OL=66:GOTO 1140
1140 FOR J=1 TO 3
1150 FOR I=1 TO LEN(O$)
1160 X=CP(ASC(MID$(O$,I,1)),1)
1170 Y=CP(ASC(MID$(O$,I,1)),2)
1180 FOR K=1 TO 30:NEXT K
1190 PUT SPRITE0,(X+8,Y+56),15
1200 NEXT I
1210 NEXT J
1220 FOR J=1 TO 10
1230 FOR I=1 TO L
1240 TE$=TE$+MID$(O$,INT(RND(1)*OL+1),1)
1250 NEXT I
1260 TE$=TE$+" "
1270 NEXT J
1280 TE$=TE$+STRING$(7," ")
1290 '**************
```

```
1300 '* Start test *
1310 '**************
1320 '
1330 TIME=0
1340 TY$=""
1350 LINE (92,15)-(165,31),1,B
1360 PLAY "S9M1000T7006"
1370 P$="CCCCCCCCCCCCCCCCCCCC"
1380 FOR I=1 TO LEN(TE$)-8
1390 LINE (93,16)-(102,30),15,BF
1400 LINE (103,16)-(164,30),7,BF
1410 DRAW "BM96,20"
1420 PRINT #1,LEFT$(TE$,8);
1430 GOSUB 1510
1440 NEXT I
1450 LINE (93,16)-(164,30),7,BF
1460 GOTO 1610
1470 '*******************
1480 '* Letter intikken *
1490 '*******************
1500 '
1510 I$=INKEY$
1520 IF PLAY(1)=0 THEN PLAY"XP$;"
1530 IF I$="" GOTO 1510
1540 X=CP(ASC(I$),1):Y=CP(ASC(I$),2)
1550 PUT SPRITE0,(X+8,Y+56),15
1560 'IF I$=CHR$(8) AND LEN(TY$)=>1 THEN TY$=LEFT$(TY$,L
EN(TY$)-1):GOTO 540
1570 TY$=TY$+I$
1580 TE$=RIGHT$(TE$,LEN(TE$)-1)+LEFT$(TE$,1)
1590 RETURN
1600 '
1610 FO=0
1620 TE$=RIGHT$(TE$,LEN(TE$)-8)
1630 FOR I=1 TO LEN(TE$)
1640 IF MID$(TE$,I,1)<>MID$(TY$,I,1) THEN FO=FO+1:FO$=FO
$+MID$(TE$,I,1)+"-"+MID$(TY$,I,1)
1650 NEXT I
1660 IF FO=0 THEN SC=SC+1
1670 IF FO=0 AND INT(((LEN(TE$)-8)*60)/(INT(TIME/50)))<5
0 THEN SC=SC-1
1680 IF FO>5 AND SC>0 THEN SC=SC-1
1690 LINE (8,150)-(248,211),7,BF
```

```
1700 T=INT(TIME/50)
1710 DRAW "BM8,150"
1720 PRINT #1, "Tijd    :";T;"sekonden"
1730 DRAW "BM8,160"
1740 PRINT #1, "Snelheid:";INT(((LEN(TE$)-8)*60)/T);"aan
slagen/minuut"
1750 PRINT #1,
1760 DRAW "BM8,170"
1770 PRINT #1,"Fouten  :";FO
1780 DRAW "BM8,180"
1790 PRINT #1,"RETURN  = doorgaan"
1800 DRAW "BM8,190"
1810 PRINT #1,"ESC     = ophouden"
1820 I$=INKEY$:IF I$="" THEN GOTO 1820
1830 IF I$=CHR$(27) THEN GOTO 3520
1840 IF I$=CHR$(24) THEN GOTO 3590
1850 IF I$<>CHR$(13) THEN GOTO 1820
1860 GOSUB 1880
1870 GOTO 690
1880 LINE (8,150)-(248,211),7,BF
1890 LINE (78,150)-(85,157),3,BF
1900 DRAW "BM90,151":PRINT#1,"PINK"
1910 LINE (78,160)-(85,167),5,BF
1920 DRAW "BM90,161":PRINT#1,"RINGVINGER"
1930 LINE (78,170)-(85,177),13,BF
1940 DRAW "BM90,171":PRINT#1,"MIDDENVINGER"
1950 LINE (78,180)-(85,187),8,BF
1960 DRAW "BM90,181":PRINT#1,"WIJSVINGER"
1970 LINE (78,190)-(85,197),11,BF
1980 DRAW "BM90,191":PRINT#1,"DUIM"
1990 RETURN
2000 '************************
2010 '* TOETSENBORD TEKENEN *
2020 '************************
2030 '
2040 FOR I=8 TO 232 STEP 16
2050 LINE (I,56)-(I+16,72),1,B
2060 NEXT I
2070 LINE (8,72)-(32,88),1,B
2080 FOR I=32 TO 208 STEP 16
2090 LINE (I,72)-(I+16,88),1,B
2100 NEXT I
2110 LINE (8,88)-(36,104),1,B
```

```
2120 FOR I=36 TO 212 STEP 16
2130 LINE (I,88)-(I+16,104),1,B
2140 NEXT I
2150 LINE (8,104)-(44,120),1,B
2160 FOR I=44 TO 188 STEP 16
2170 LINE (I,104)-(I+16,120),1,B
2180 NEXT I
2190 LINE (204,104)-(232,120),1,B
2200 LINE (232,104)-(248,120),1,B
2210 LINE (44,120)-(60,136),1,B
2220 LINE (60,120)-(76,136),1,B
2230 LINE (76,120)-(204,136),1,B
2240 LINE (204,120)-(220,136),1,B
2250 LINE (248,72)-(248,104),1
2260 DATA E,0,0,3
2270 DATA 1,16,0,3
2280 DATA 2,32,0,5
2290 DATA 3,48,0,13
2300 DATA 4,64,0,8
2310 DATA 5,80,0,8
2320 DATA 6,96,0,8
2330 DATA 7,112,0,8
2340 DATA 8,128,0,13
2350 DATA 9,144,0,5
2360 DATA 0,160,0,3
2370 DATA -,176,0,3
2380 DATA =,192,0,3
2390 DATA \,206,0,3
2400 DATA <,222,0,3
2410 DATA TB,0,16,3
2420 DATA Q,24,16,3
2430 DATA W,40,16,5
2440 DATA E,56,16,13
2450 DATA R,72,16,8
2460 DATA T,88,16,8
2470 DATA Y,104,16,8
2480 DATA U,120,16,8
2490 DATA I,136,16,13
2500 DATA O,152,16,5
2510 DATA P,168,16,3
2520 DATA [,184,16,3
2530 DATA ],200,16,3
2540 DATA [,216,16,3
```

```
2550 DATA CTL,0,32,3
2560 DATA A,28,32,3
2570 DATA " ",120,64,11
2580 DATA D,60,32,13
2590 DATA F,76,32,8
2600 DATA G,92,32,8
2610 DATA H,108,32,8
2620 DATA J,124,32,8
2630 DATA K,140,32,13
2640 DATA L,156,32,5
2650 DATA ;,172,32,3
2660 DATA ',188,32,3
2670 DATA 204,32,3
2680 DATA SFT,0,48,3
2690 DATA Z,36,48,3
2700 DATA X,52,48,5
2710 DATA C,68,48,13
2720 DATA V,84,48,8
2730 DATA B,100,48,8
2740 DATA N,116,48,8
2750 DATA M,132,48,8
2760 DATA ",",148,48,13
2770 DATA .,164,48,5
2780 DATA /,180,48,3
2790 DATA SFT,196,48,3
2800 DATA S,44,32,5
2810 RESTORE 2260
2820 FOR I=1 TO 55
2830 READ I$,X,Y,K
2840 CP(ASC(I$),1)=X: CP(ASC(I$),2)=Y
2850 PSET(X+12,Y+60):PRESET(X+12,Y+60)
2860 PRINT #1,I$
2870 PAINT(X+11,Y+60),K,1
2880 NEXT I
2890 LINE(104,40)-(104,72),15
2900 LINE -(112,72),15
2910 LINE -(112,88),15
2920 LINE -(116,88),15
2930 LINE -(116,104),15
2940 LINE -(124,104),15
2950 LINE -(124,120),15
2960 DRAW "BM8,44":PRINT#1,"LINKER HAND"
2970 DRAW "BM152,44":PRINT#1,"RECHTER HAND"
```

```
2980 GOSUB 1880
2990 RESTORE 3050
3000 FOR I=1 TO 47
3010 READ I$,X,Y
3020 CP(ASC(I$),1)=X: CP(ASC(I$),2)=Y
3030 NEXT I
3040 RETURN
3050 DATA !,16,0
3060 DATA @,32,0
3070 DATA #,48,0
3080 DATA $,64,0
3090 DATA %,80,0
3100 DATA ^,96,0
3110 DATA &,112,0
3120 DATA *,128,0
3130 DATA (,144,0
3140 DATA ),160,0
3150 DATA _,176,0
3160 DATA +,192,0
3170 DATA |,206,0
3180 DATA q,24,16
3190 DATA w,40,16
3200 DATA e,56,16
3210 DATA r,72,16
3220 DATA t,88,16
3230 DATA y,104,16
3240 DATA u,120,16
3250 DATA i,136,16
3260 DATA o,152,16
3270 DATA p,168,16
3280 DATA {,184,16
3290 DATA },200,16
3300 DATA a,28,32
3310 DATA d,60,32
3320 DATA f,76,32
3330 DATA g,92,32
3340 DATA h,108,32
3350 DATA j,124,32
3360 DATA k,140,32
3370 DATA l,156,32
3380 DATA ":",172,32
3390 DATA chr$(34),188,32
3400 DATA ~,204,32
```

```
3410 DATA z,36,48
3420 DATA x,52,48
3430 DATA c,68,48
3440 DATA v,84,48
3450 DATA b,100,48
3460 DATA n,116,48
3470 DATA m,132,48
3480 DATA <,148,48
3490 DATA >,164,48
3500 DATA ?,180,48
3510 DATA s,44,32
3520 CLOSE #1
3530 OPEN "a:"+N$+".typ" FOR OUTPUT AS #1
3540 PRINT #1,SC
3550 CLOSE #1
3560 SCREEN 0
3570 PRINT "Je mag de computer nu uitschakelen.  Als je
de volgende keer weer oefent en je gebruikt dezelfde sch
ijf, dan  zul je op niveau";SC;"beginnen."
3580 END
3590 '***** Fouten-analyse
3600 GOTO 1820
3610 '
3620 '
3630 IF ERR=53 AND (I$="j" OR I$="J") THEN SC=0: RESUME
450
3640 RESUME
```

APPENDIX B

TREFWOORDEN

A

B

C

D

E

F

G

H

I

K

L

M

N

T

V

W

X

Z

Bij dezelfde uitgever verschenen:

Nederlandse boeken en software voor MSX- en MSX2-gebruikers:

Titel	ISBN-nummer	prijs
MSX boeken		
MSX Basic handboek	90 6398 100 7	*f 49,95*
MSX Disk handboek	90 6398 407 3	*f 29,80*
MSX DOS handboek	90 6398 674 2	*f 26,75*
MSX Quick disk handboek	90 6398 254 2	*f 23,70*
MSX Machinetaalhandboek	90 6398 735 8	*f 34,80*
MSX(2) en machinetaal - de afstand overbrugd	90 6398 669 6	*f 32,50*
MSX Basic voor kinderen *deel 1*	90 6398 084 1	*f 19,70*
MSX Basic voor kinderen *deel 2*	90 6398 304 2	*f 24,75*
MSX Basic leerboek *deel 1*	90 6398 649 1	*f 24,75*
MSX Basic leerboek *deel 2*	90 6398 769 2	*f 24,75*
MSX DOS leerboek *deel 3*	90 6398 519 3	*f 24,75*
MSX2 Leerboek *deel 4*	90 6398 737 4	*f 24,75*
MSX Basic met vpoke en sprite toepassingen	90 6398 372 7	*f 27,50*
Computer en modem voor homecomputers	90 6398 938 5	*f 36,75*
MSX Computer en printers aansluiten en gebruiken	90 6398 415 4	*f 27,75*
MSX Verder uitgediept	90 6398 447 2	*f 24,10*
MSX Praktijkprogramma's	90 6398 437 5	*f 24,75*
MSX/MSX2 mogelijkheden	90 6398 606 8	*f 29,80*
MSX computers en de buitenwereld	90 6398 740 4	*f 39,85*
MSX Truuks en Tips *deel 1*	90 6398 900 8	*f 25,15*
MSX Truuks en Tips *deel 2*	90 6398 340 9	*f 25,15*
MSX Truuks en Tips *deel 3*	90 6398 910 5	*f 25,15*
MSX Truuks en Tips *deel 4*	90 6398 897 4	*f 25,15*
MSX Truuks en Tips *deel 5*	90 6398 745 5	*f 25,15*
MSX Truuks en Tips *deel 6*	90 6398 879 6	*f 25,15*
MSX Truuks en Tips *deel 7*	90 6398 789 7	*f 25,15*
MSX Truuks en Tips *deel 8*	90 6398 850 8	*f 25,15*
MSX2 boeken		
MSX2 Basic handboek	90 6398 221 6	*f 57,05*
MSX2 Disk/DOS handboek	90 6398 222 4	*f 37,85*
MSX2 Utility-handboek	90 6398 223 2	*f 30,05*
MSX2 Zakboekje	90 6398 224 0	*f 27,75*
MSX2 Machinetaalhandboek	90 6398 915 6	*f 42,90*
MSX/MSX2 software		
FISTAN/MSX2 administratiepakket	90 6398 819 2	*f 300,50*
FASTAN/MSX2 factureringspakket	90 6398 889 3	*f 300,50*
KASTAN/MSX2 kaartenbakprogramma	90 6398 940 7	*f 149,-*
FLASH assembler/disassembler	90 6398 721 8	*f 119,-*
Lesmaker	90 6398 170 8	*f 495,-*
SuperKasboek	90 6398 230 5	*f 149,-*
SnelFaktuur/MSX2	90 6398 826 5	*f 149,-*

LEERBOEK MSX2 deel 4

De serie MSX Leerboeken geeft een komplete kursus programmeren in MSX-BASIC. De eerste twee delen behandelen de volledige MSX-BASIC taal, in het derde deel komt het MSX-DOS aan bod en in het vierde deel worden alle extra's van MSX2 uitgediept.

De MSX Leerboeken zijn gericht op de beginnende programmeur. De moeilijkheidsgraad van de leerstof wordt dan ook slechts geleidelijk hoger. De gebruikte voorbeelden zijn zo gekozen, dat al in een vroeg stadium bruikbare programma's kunnen worden gemaakt. Dit zal de lezer/leerling er toe aansporen om verder te gaan. Aan het eind van ieder deel is een groot voorbeeldprogramma opgenomen. Dit programma laat zien waartoe de lezer/leerling na het bestuderen van het boek in staat zal zijn.

Bij ieder leerboek is een afzonderlijk "Opdrachten en uitwerkingen" boekje verkrijgbaar. In deze boekjes staan, in volgorde van de hoofdstukken uit het Leerboek, vragen en opdrachten met antwoorden en uitwerkingen. Zowel voor gebruik op school als voor individueel gebruik zullen deze boekjes erg nuttig zijn.

ISBN 90 6398 737 4