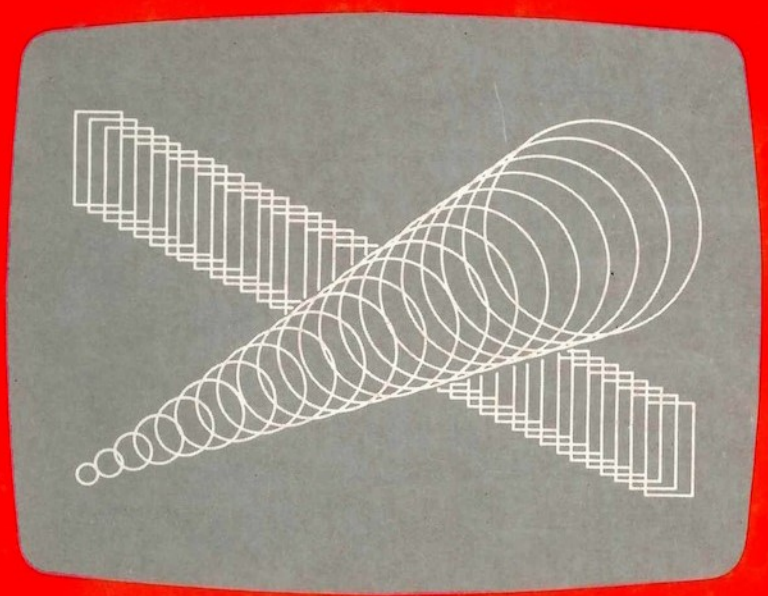
MSX BASIC

LEREN PROGRAMMEREN



M.B. IMMERZEEL

DE MUIDERKRING

HOME COMPUTER

M. B. IMMERZEEL - MSX-BASIC - LEREN PROGRAMMEREN



Beste lezer,

Dit MSX boek is afkomstig uit de nalatenschap van Wammes Witkop - hoofdredacteur MSX Computer Magazine 1985 - 1992.

MSX-BASIC

© 1985 De Muiderkring BV - Bussum - Nederland

Niets uit deze uitgave mag worden verveelvoudigd en/of openbaar gemaakt door middel van druk, fotocopie, microfilm of op welke andere wijze ook zonder voorafgaande toestemming van de uitgever.

ISBN 90 6082 259 5

M. B. IMMERZEEL

MSX BASIC

LEREN PROGRAMMEREN

DE MUIDERKRING B.V. – BUSSUM
UITGEVERIJ VAN TECHNISCHE BOEKEN EN TIJDSCHRIFTEN

Inhoud

Voorwoord 7
1. Inleiding 8
   1.1. Het programma 8
   1.2. De opbouw van de computer 8
   1.3. Programmeertalen 8
   1.4. De indeling van de geheugenruimte 9
2. Het gebruik van het toetsenbord 11
   2.1. Het beeldscherm 11
   2.2. De cursorbewegingen 11
   2.3. De kleur van het scherm en de tekens 12
   2.4. Kleine letters en hoofdletters 13
   2.5. De grafische karakterset 13
   2.6. Het corrigeren van teksten 14
   2.7. De 40 en 32 kolommode 14
3. De MSX computer als rekenmachine, de direct mode 16
   3.1. Het PRINT commando 16
   3.2. De rekenkundige bewerkingen 17
4. Programmeren in BASIC, program mode 19
   4.1. Het eerste programma 19
   4.2. Benamingen en variabelen 23
   4.3. Basisconstructies 24
5. Het invoeren van de gegevens (data) 28
   5.1. Het INPUT statement 28
   5.2. Het READ statement 31
6. Variaties met variabelen 33
   6.2. Het definiëren van variabelen 34
   6.3. Bewerkingen met variabelen 34
   6.4. Het printen van variabelen 35
7. Het werken met het cassettedek 38
   7.1. Inleiding 38
   7.2. Het SAVEN van programma's 38
   7.3. Het samenvoegen van programma's van de tape 39
   7.4. De behandeling van een ''FILE'' 40
8. De ASCII-code 43
   8.1. Het buffergeheugen 43
   8.2. De codegetallen 43
   8.3. De timer 46
   8.4. De functietoetsen 46
   8.5. De interval 47

9. Het veranderen van de inhoud van geheugenplaatsen 49
9.1. Het lezen en het schrijven van geheugenplaatsen 49
9.2. Positieve binaire getallen 49
9.3. Negatieve binaire getallen 50
9.4. Hexadecimale en octale getallen 51
9.5. Logische bewerkingen met binaire getallen 52
9.4. Het werken met een masker 55
10. Het toevalsgetal 57
10.1. De RND(X) functie 57
10.2. Twee toepassingen 58
10.3. Het voorkomen van foutmeldingen 63
10.4. De iteratiemethode 64
11. De geluidsgenerator 66
11.1. Inleiding 66
11.2. Het invoeren van de gegevens met het PLAY statement 68
11.3. Het invoeren van de gegevens met het SOUND statement 71
11.4. Ruis 74
12. Grafische functies, de veertigkolommode 76
12.1. De indeling van het scherm 76
12.2. Bewegende beelden 77
12.3. De dobbelsteen 79
12.4. De opbouw van de karakters 81
12.5. Het zelf karakters maken 83
12.6. Het verplaatsen van het schermgeheugen 85
13. Grafische functies, de tweeendertigkolommode 86
13.1. De indeling van het scherm 86
13.2. Het verplaatsen van het kleurgeheugen 87
13.3. Het gebruik van sprites 88
14. Grafische functies, hoge resolutie 93
14.1. De indeling van het scherm 93
14.2. Het tekenen van rechte lijnen en cirkels 95
14.3. Het DRAW statement 96
14.4. Grafieken 98
14.5. Het printen van tekst 99
15. Grafische functies, multi color mode 100
15.1. De indeling van het scherm 100
15.2. Twee toepassingen 102

Voorwoord

De bedoeling van dit boek is dat het een beschrijving geeft van de Microsoft Extendid BASIC programmeertaal die wordt gebruikt bij alle computers die zijn voorzien van de MSX BASIC version 1.0.
Nu is de MSX BASIC een zeer uitgebreide BASIC. Dat houdt in dat voor allerlei functies - en dat zijn er bij deze computer nogal wat - er instructies zijn die het programmeren moeten vereenvoudigen. Met dit boek kunt u stap voor stap inzicht in uw computer krijgen door het invoeren van programma's die speciaal zijn ontwikkeld om u de functies van uw computer te leren kennen. Daarbij komt u, tot het laatste programma toe, steeds weer nieuwe instructies tegen waarvan de werking duidelijk wordt verklaard. Deze programma's zijn in het begin zeer eenvoudig opgebouwd en worden later uitgebreider, zodat vanzelf uw inzicht in het programmeren meegroeit. Het leren in dit boek betekent vaak dat u aan de resultaten op het beeldscherm de werking van het programma en de opbouw van de computer leert kennen. U leert door onderzoeken van de mogelijkheden.

Bij de samenstelling van dit boek is er vanuit gegaan dat u, behalve een MSX computer, ook een cassetterecorder (eventueel een speciale DATA recorder) en een kleurentelevisie bezit, niet meer en niet minder.

Ede, voorjaar 1985

Bij het werken met computers is het nodig enig inzicht in binaire getalstelsels te hebben. Moeilijk is dat niet, maar iemand die nog niet daarmee in aanraking is gekomen staat er vreemd tegenover. Dat kan voor u geen hindernis meer zijn want hieraan is in dit boek precies die aandacht besteed die nodig is voor het opdoen van de nodige kennis.

Helaas heeft ieder boek zijn tekortkomingen, de een meer en de ander minder. De bedoeling is dat dit boek hoort bij de laatste categorie, want het behoort tot een serie waarvan het de doelstelling is dat het veel informatie geeft voor zo weinig mogelijk geld. Als er dan ook een instructie niet direct is beschreven dan is het er een die u na het lezen van dit boek ook zonder verklaring kunt toepassen.

Vaardigheid in programmeren kunt u krijgen door het veel te doen, maar ook door veel programma's van anderen te bestuderen. Om daarvoor in aanraking te komen met anderen die dezelfde hobby uitoefenen kunt u zich het beste aanmelden bij een zogenaamde gebruikersgroep. Een boek kan niet compleet zijn zonder u het adres van deze groep te geven:
De MSX gebruikersgroep voor de Benelux,
Batterijlaan 39,
1402 SM Bussum.

M. B. Immerzeel

1. Inleiding

1.1. Het programma

Van zeer jongs afaan hebben we ondervonden dat bepaalde handelingen slechts in een bepaalde volgorde kunnen of moeten worden verricht. Zo hebben we al heel vroeg geleerd dat eerst de sokken en daarna pas de schoenen moeten worden aangetrokken. Het ''leren'' is al begonnen toen we voor het eerst belangstelling voor ons zelf en onze omgeving gingen krijgen en zal op alle terreinen blijven doorgaan, zolang ons leven duurt. Dit leren houdt in dat ons moet worden bijgebracht hoe en in welke volgorde bepaalde handelingen moeten worden gedaan om tot een resultaat te komen. Zo is ons op school verteld hoe bij het vermenigvuldigen een aantal optellingen in een zekere volgorde moeten worden uitgevoerd om het produkt van twee getallen te vinden. Anders gezegd: er moeten een aantal *handelingen* of *operaties* worden uitgevoerd in een bepaalde volgorde - volgens een programma - met een paar getallen, de *operanden.*
Gelukkig hoeft ons niet alles te worden bijgebracht. Soms kunnen we ook zelf wel iets leren, dus ons bepaalde programma's eigen maken, zonder directe hulp van anderen. Deze mogelijkheid hebben de ''personal computers'' waartoe de MSX computers behoren, nog niet. Deze computers moet alles worden ''geleerd''. Daarom moet behalve de operanden - die later anders zullen worden genoemd - ook een programma worden toegediend. Dit programma moet helemaal gericht zijn op de mogelijkheden die de computer heeft, dat wil zeggen: gericht zijn op de handelingen die de computer kan uitvoeren. Zo moet de Commodore 64 anders worden geprogrammeerd dan de MSX computer omdat de fabrikanten verschillende mogelijkheden in de computer hebben ingebouwd.

1.2. De opbouw van de computer

Hoe verschillend de computers ook kunnen zijn, de basisopbouw is in principe gelijk. Er is een *geheugen* waarin de operanden en het programma worden opgeslagen en er is een centrale eenheid, de centrale *processor,* die de handelingen, voorgeschreven door het programma, één voor één verricht. Hoe dat in zijn werk gaat kunt u lezen in andere boeken. Voor ons is het belangrijk te weten dat de handelingen die door de processor moeten worden verricht, aan hem moeten worden medegedeeld door een codegetal, de *operatiecode* en een programma is daarom een aaneenschakeling van operatiecodes en operanden.
Een operatiecode kan een getal zijn van 0 tot en met 255. Met 0 mee zijn er dus 256 mogelijkheden waarvan slechts een gedeelte wordt gebruikt. Ook de operanden zijn gecodeerd. Daardoor zijn niet alleen cijfers maar ook letters en andere tekens te coderen. Elke operand en elke operatiecode wordt in één geheugenplaats ondergebracht. Een programma bestaande uit 500 operatiecodes en operanden heeft daarom ook 500 geheugenplaatsen nodig. Een MSX computer heeft, zonder speciale voorzieningen, een *geheugenruimte* van 65536 geheugenplaatsen en elke geheugenplaats kan een (code)getal bevatten dat niet groter is dan 255. Elke geheugenplaats krijgt zijn eigen nummer, het adres: de MSX computer heeft dus 65536 adressen. Dit wordt een 64 K geheugenruimte genoemd. Hoe dit te rijmen valt met 65536 geheugenplaatsen laat zich als volgt verklaren.
Bij de computer wordt steeds uitgegaan van machten van 2 (2^0, 2^1, 2^2, 2^3 enz.). Zo is er een geheugen van 1024 (2^{10}) geheugenplaatsen. Dit wordt voor het gemak een 1 K-geheugen genoemd.
De MSX computer heeft een geheugenruimte van $64 \times 1024 = 65536$ geheugenplaatsen, vandaar! Overigens, $256 = 2^8$.

1.3. Programmeertalen

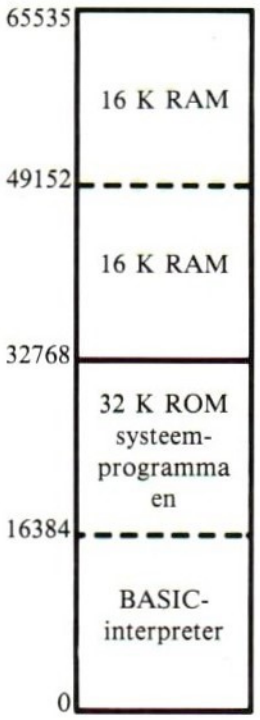
Hoewel het aantal operaties dat de computer kan uitvoeren, beperkt is en de operaties zelf vaak erg eenvoudig zijn (de rekenkundige operaties zijn bijvoorbeeld niets meer dan optellen en aftrekken), zijn de toepassingsmogelijkheden van de computer bijna onbeperkt. Er moeten hem dan wel de juiste gegevens en programma's kunnen worden toege-

diend en de computer moet ook de resultaten kunnen bekendmaken.
Er is dan ook een soort van communicatie nodig tussen de computer en de gebruiker. Het inbrengen van de gegevens en de programma's kan plaatsvinden met het toetsenbord, magneetband (cassette) of magneetschijf (floppy disk) en het uitvoeren van gegevens kan gebeuren met een beeldscherm (tv), cassetteband, floppy disk of een printer.
Deze hulpmiddelen moeten dan wel alle handelingen opgedragen krijgen die nodig zijn voor het lezen van de toetsen, de cassetteband en de floppy disk en voor het schrijven naar de diverse randapparaten. In de computer is daarvoor al een uitgebreid programma aanwezig, het *systeemprogramma*. Dit programma is samengesteld uit operaties en operatiecodes en wordt een *machinetaalprogramma* genoemd. De codes zijn als woorden van een taal op te vatten en slechts *die* taal wordt door de centrale processor verstaan. Ook wij zelf kunnen programma's in machinetaal in de computer brengen. Dit wordt "programmeren in machinetaal" genoemd.
Meestal is dit echter een vrij langdurige aangelegenheid omdat vaak een groot aantal operaties elkaar moeten opvolgen voor een bepaalde handeling.
Een ingewikkelde vermenigvuldiging zal bijvoorbeeld bestaan uit een zeer groot aantal optellingen. Er zijn daarom een aantal "hogere" programmeertalen ontwikkeld en daar is BASIC er één van. BASIC is een afkorting van Beginners Allpurpose Symbolic Instruction Code en is een programmeertaal die zeer populair is. De bedoeling zal duidelijk zijn: één enkele instructie moet een groot aantal operatiecodes vervangen. Er is dan wel een zogenaamde BASIC interpreter nodig die de in de BASIC taal geschreven programma's vertaalt naar een machinetaalprogramma, want dat is uiteindelijk de enige taal die de computer verstaat. Ook de BASIC interpreter kan als een systeemprogramma worden beschouwd. Met dit programma wordt een instructie in de BASIC taal omgezet in een reeks van operatiecodes, nodig om de computer de juiste handelingen te laten verrichten.

Ofschoon een programma dat in de BASIC taal is geschreven eenvoudiger is, duurt het - in verband met het vertalen - zeer veel langer om zo'n programma door de computer te laten uitvoeren, dan een machinetaalprogramma dat dezelfde werking heeft. Waar snelheid is gewenst, moet wel eens een machinetaalprogramma in het BASIC programma worden gevoegd.

1.4. De indeling van de geheugenruimte

De adressen in de geheugenruimte zijn genummerd van 0 tot en met 65535. Op elk adres *kan* zich een geheugenelement (een elektronische schakeling) bevinden dat een codegetal kan bevatten. Bij de MSX computers kan dit getal niet groter zijn dan 255. Dit wordt ook wel een "byte" genoemd.
Voor het invoeren van een programma en de operanden zijn geheugenelementen nodig die de code getallen kunnen opnemen, *het inschrijven* naar de geheugenelementen. Later moet het getal weer op te halen zijn, het *lezen* van de geheugenplaatsen. Dit zijn de zogenaamde RAM geheugenelementen (Random Access Memory), Deze geheugenelementen verliezen hun informatie als de spanning op de computer wordt uitgeschakeld.



afb. 1

Zij kunnen daarom niet worden gebruikt voor het systeemprogramma en de BASIC interpreter. Deze *moeten* altijd in de computer aanwezig zijn, anders zou hij na het inschakelen van de spanning niet

eens te bedienen zijn. Voor deze programma's worden geheugenelementen gebruikt die niet zijn in te schrijven, maar alleen zijn uit te lezen, ROM geheugenelementen (Read Only Memory). Zij worden door de fabrikant van informatie voorzien die ze nooit meer kwijtraken.
De indeling van de geheugenruimte geeft afb. 1. De 64 Kbyte geheugenruimte (dat wil zeggen: er is plaats voor $64 \times 1024 = 65536$ geheugenelementen die elk 1 byte kunnen bevatten) die in de normale situatie wordt gebruikt (bij BASIC programmeren) is verdeeld in vier blokken van 16 Kbyte. De adressen zijn genummerd van 0 tot en met 65535. De onderste twee blokken van 16 Kbyte, op de adressen van 0 tot en met 32767, zijn gevuld met ROM geheugenelementen voor het systeemprogramma en de BASIC interpreter. De andere twee blokken op de adressen 32768 tot en met 65535 zijn geheel of gedeeltelijk (afhankelijk van de uitvoering van uw computer) gevuld met RAM geheugenelementen voor het programma en de gegevens.

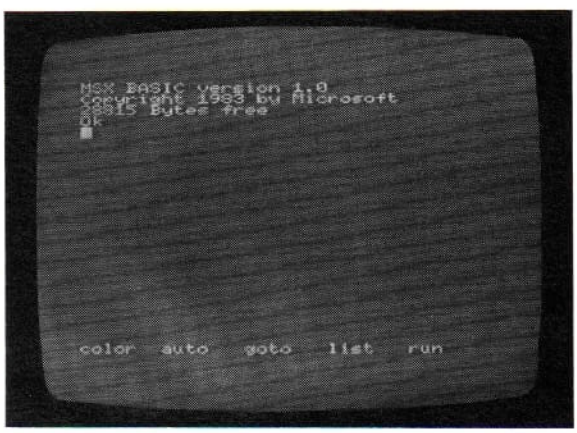
In het begin van dit hoofdstuk is geschreven over de centrale processor die uw computer bezit. De MSX computer bevat nog een tweede processor, de Video Display Processor (VDP). Deze wordt gebruikt voor alles wat nodig is om de tekst op uw beeldscherm te krijgen. Ook deze tekst moet in een geheugen worden opgeslagen en men heeft daarvoor bij deze processor een RAM geheugen van 16 Kbyte aangebracht. Deze geheugenruimte valt dus buiten die in afb. 1.

2. Het gebruik van het toetsenbord

2.1. Het beeldscherm

Aangenomen wordt dat u, eventueel met behulp van de handleiding of de verkoper, de computer hebt kunnen aansluiten op een tv en een cassettedeck. Deze laatste heeft u beslist nodig. De computer heeft een normaal "QWERTY" toetsenbord, dat wil zeggen; een toetsenbord dat veel overeenkomst vertoont met dat van een schrijfmachine. Er zijn ook een aantal afwijkende toetsen terwijl de toetsen meerdere functies hebben. Niet alle toetsen worden in dit hoofdstuk behandeld. Een toets die hierin niet is behandeld komt in een later hoofdstuk ter sprake.
Na het inschakelen van de computer duurt het enige seconden voordat er iets op het scherm zichtbaar wordt. Is er in uw computer al software "ingebouwd" (afhankelijk van het type en het fabrikaat) dan wordt een zogenaamd "MENU" op het scherm getoond. Dit MENU geeft aan welke functies u kunt kiezen en op welke manier. Omdat dit boek het programmeren in BASIC betreft kiest u uiteraard BASIC. Het MENU verdwijnt dan van uw scherm. Dit zal nu in het algemeen de volgende mededeling tonen:

MSX BASIC version 1,0
Copyright 1983 by Microsoft.
. . . . . Bytes free.
Ok



Het laatste Ok geeft aan dat de computer gereed is om de opdrachten te ontvangen.
Is in uw computer geen software ingebouwd dan komt hij na het inschakelen gelijk al in de "BASIC mode". Ook dan zal in het algemeen bovenstaande tekst te lezen zijn.
Daarbij ziet u aan de linkerkant van het scherm een vierkantje. Dit wordt een *cursor* genoemd en geeft de plaats aan waar het eerst volgende teken zal worden geschreven. Nu staat de cursor nog aan het begin van een regel maar hij verplaatst zich langs die regel als u met het toetsenbord een tekst intikt (doen!). Er kunnen hooguit zevenendertig tekens op een regel (inclusief de spaties). Is een regel vol dan wordt automatisch overgegaan naar het begin van de volgende regel. Gebruik hiervoor *niet* de RETURN toets. Indien u niet willekeurig in een woord wilt afbreken, maak dan de resterende plaatsen op de regel vol met spaties. Met al dat gene wat u hebt ingetikt doet de computer overigens niets.
Typt u eens door totdat het gehele beeldscherm is gevuld. Typt u nu nog meer tekst dan ziet u alle regels op het scherm naar boven opschuiven terwijl de nieuwe tekst op de vrije onderste regel verschijnt. Dit wordt "verticale scrolling" genoemd. Het zal u zijn opgevallen dat de onderste regel van het scherm al tekst bevat en dat deze regel door u niet kan worden beschreven. De tekst op deze regel geeft de functie van de functietoetsen f1 tot en met f5. Als u op de SHIFT toets drukt dan ziet u de functies van de toetsen f6 tot en met f10.

2.2. De cursorbewegingen

Drukt u, nadat u het een en ander aan tekst op het scherm hebt gebracht, eens op de toets HOME. De cursor zal nu geheel links bovenaan het veld verschijnen, op de eerste regel.
Drukt u op de SHIFT toets en gelijktijdig op de HOME toets (HOME in de SHIFT mode) dan zal niet alleen de cursor geheel links bovenaan het scherm verschijnen maar zal ook alle tekst van het

scherm verdwijnen; het scherm is "schoon gemaakt" of "gewist".
De cursor kan met vier toetsen op elke plaats van het scherm worden geplaatst. De pijlen op de toetsen geven aan in welke richting de cursor kan worden verplaatst. Deze toetsen zijn *repeterend,* dat wil zeggen dat de cursor zich blijft verplaatsen zolang de betreffende toets ingedrukt wordt gehouden. Na het indrukken zult u daarop wel heel even moeten wachten. Als u twee van deze toetsen gelijktijdig ingedrukt houdt (uiteraard niet de toetsen die een tegenovergestelde cursorbeweging ten doel hebben) dan ziet u de cursor schuin over het scherm gaan.
Alle toetsen, inclusief de spatiebalk, zijn overigens repeterend. Houdt maar eens een willekeurige toets enige tijd ingedrukt! Als u het geluid van uw tv hard genoeg hebt afgesteld dan zult u steeds als de werking van een toets wordt uitgevoerd (het plaatsen van een letter op het scherm of het geven van een spatie) een korte tik uit de luidspreker van uw tv kunnen horen.

2.3. De kleur van het scherm en de tekens

In principe is het beeldscherm in twee gedeelten ingedeeld. Het eerste gedeelte is het *veld.* Dat is het gedeelte van het scherm dat de tekst kan bevatten. De rand hier omheen is het *kader.* Omdat dit kader dezelfde kleur heeft als het veld is het hiervan niet te onderscheiden. Zowel de kleur van het veld en het kader als die van de tekens op het veld kan worden veranderd. Hiervoor kan gekozen worden uit vijftien kleuren. Elke kleur wordt met een kleurnummer aangegeven.

code	kleur	code	kleur
0	transparant	8	rood
1	zwart	9	lichtrood
2	middelgroen	10	donkergeel
3	lichtgroen	11	lichtgeel
4	donkerblauw	12	donkergroen
5	lichtblauw	13	magenta
6	donkerrood	14	grijs
7	hemelsblauw	15	wit

Lijst 1

In lijst 1 kunt u zien welke kleuren mogelijk zijn en welk nummer (van 1 tot en met 15) bij de kleur hoort. Het nummer 0 hoort bij *transparant.* Dat betekent dat er wel tekens op het scherm aanwezig zijn maar dat deze niet waargenomen kunnen worden (een "verborgen" tekst dus). Hetzelfde effect wordt verkregen als het veld en de tekens dezelfde kleur hebben.

De kleur kan worden bepaald door het invoeren van het commando COLOR gevolgd door 1, 2 of 3 kleurnummers ("code" in tabel 1), gescheiden door komma's. Zet de cursor aan het begin van een nieuwe regel (een regel zonder tekst) en toets eens in:

```
color 15,6
```

Druk nadat u de 6 hebt ingetoetst op de toets RETURN. U krijgt dan witte letters op een rode achtergrond. Nu had u het uzelf ook makkelijk kunnen maken door de f1 toets in te drukken. Het woord color had dan op uw scherm gestaan. Zelf had u verder alleen nog maar de nummers 15 (voor de kleur van de tekens, wit) en het nummer 6 (voor de veldkleur, donkerrood) hoeven in te tikken. Vergeet vooral de komma niet! Zorg er ook voor dat het commando op een "schone" regel komt te staan. Dit geldt voor alle commando's die moeten worden gegeven. Toets nu eens in:

```
color 6
```

Doe dat eventueel met behulp van de f1 toets, dus: f1 toets indrukken en dan toets 6. Druk daarna op de RETURN toets. Dit is belangrijk. Pas als de RETURN toets is ingedrukt weet de computer dat de ingevoerde regel een commando is die hij moet uitvoeren. U ziet dat alles wat op het scherm stond geschreven is verdwenen, althans *lijkt* verdwenen. In werkelijkheid staan de tekens nog op het scherm maar zijn niet te zien omdat ze dezelfde kleur hebben als de achtergrond. Het scherm is dus niet "gewist".
Druk nu op de SHIFT toets en tegelijk op de f1 toets. Dit geeft de functie f6. Nu worden de kleuren op uw scherm weer zoals die waren na het inschakelen van de computer en zijn alle tekens weer zichtbaar. De functie f6 heeft hetzelfde effect als

```
color 15,4,4 (RETURN)
```

en deze regel ziet u dan ook op het scherm geschreven (behalve RETURN). Het gebruik van de func-

tietoets heeft dan ook als doel om een bepaalde zin (of een enkel woord) op het scherm te schrijven, al of niet gevolgd door een RETURN.
Het beste leert u uw computer kennen door veel te experimenteren. Probeer dan ook eens een aantal keren de kleuren te veranderen met het color commando, gevolgd door twee kleurnummers. Het derde nummer achter color is voor het kader en dat heeft voorlopig nog geen werking. Probeer ook eens:

color 0,4 (RETURN)

en

color 0,6 (RETURN)

U ziet dan wat met ''transparant'' (kleurnummr 0) wordt bedoeld.

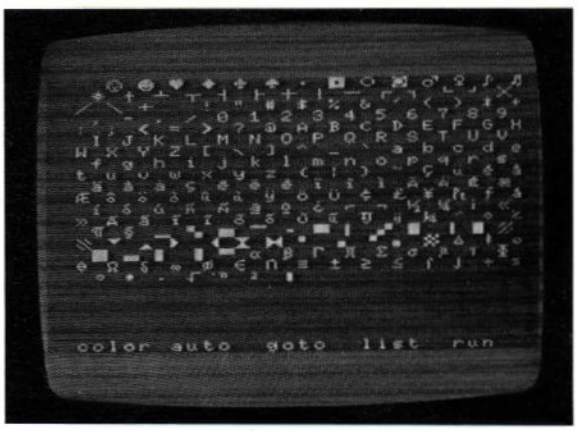
2.4. Kleine letters en hoofdletters

Het zal u opgevallen zijn dat de tekens die u met de toetsen op het scherm kunt schrijven, wat de letters betreft, kleine letters zijn.
Als u niets bijzonders doet dan staat de computer in de *onderkast* mode (engels: lower case). De tekens die u op het scherm ziet, staan ook op de toetsen. Omdat bij sommige MSX computers ook nog wel andere tekens op de toetsen zijn aangegeven zullen we dit de ''voornaamste tekens'' noemen. Op bepaalde toetsen staan twee van deze tekens boven elkaar. Hierbij zijn ook de leestekens (, : . ; en dergelijke) en hiervan wordt steeds het onderste teken op het scherm ''geprint''. Gebruikt u bij het intoetsen de SHIFT toets dan krijgt u hoofdletters op het scherm (kapitaal, engels: upper case) en van de toetsen waarop twee ''voornaamste'' tekens staan, het bovenste teken.
Op uw toetsenbord is een toets met het opschrift ''CAP'' of ''CAPS''. Drukt u deze toets in dan staat de computer in de *kapitaal* mode en worden alle letters als hoofdletters op het scherm geplaatst. Van de leestekens kunt u dan weer de onderste op de toets of de bovenste kiezen, door al of niet gebruik te maken van de SHIFT toets. U kunt de kapitaal mode verlaten door opnieuw de toets CAP (CAPS) in te drukken.

Behalve deze tekens kunt u nog veel meer karakters uit uw MSX computer halen. Hiervoor dient de toets ''CODE''. Drukt u deze gelijk met een toets in dan verschijnt, afhankelijk van de gebruikte toets, een frans, duits, deens, zweeds, italiaans of spaans karakter. In bepaalde gevallen kunt u hiervan ook de hoofdletters op het scherm laten afdrukken. Dan moet gelijk met de toets CODE ook de SHIFT toets worden ingedrukt. Staan deze karakters op de toetsen van uw computer dan zijn ze aan de linkerkant daarvan afgebeeld. Probeer ook eens wat er gebeurt als u daarbij de computer in de kapitaal mode schakelt.

2.5. De grafische karakterset

Behalve een aantal karakters (tekens) waarmee u een bepaalde tekst op het scherm kunt schrijven zijn er in de MSX computer ook nog een aantal karakters ter beschikking waarmee u op het scherm kunt tekenen.
Hiervoor dient u de computer in de *grafische* mode te plaatsen met de toets die gemerkt is met GRAPH. Afhankelijk van het type computer dient u deze toets ingedrukt te houden dan wel eenmalig in te drukken.
In het laatste geval kunt u de grafische mode weer verlaten door de toets nogmaals in te drukken. Elke toets waarmee u in de normale mode letters en cijfers op het scherm kunt plaatsen zal in de grafische mode ook grafische tekens op het scherm brengen.



Breng de cursor aan het begin van een lege regel en druk in de grafische mode na elkaar de toetsen r, t en y in, in deze volgorde. Breng de cursor naar het begin van de volgende regel en gebruik nu na elkaar de toetsen v, b en n. U ziet vanzelf wat het

resultaat is. Ook in de SHIFT mode zullen de meeste toetsen een karakter op het scherm printen. Heeft u geen kaart met een afbeelding van alle functies van de toetsen of zijn de karakters niet op de toetsen afgebeeld, dan kunt u door proberen er achter komen welke karakters door een bepaalde toets op het scherm worden geplaatst.
Druk in de grafische mode eens toets p in. U krijgt dan een vlakje gelijk aan de cursor te zien. U kunt in de SHIFT mode eenzelfde vakje op het scherm krijgen met vier toetsen. Probeer deze toetsen eens te vinden.

2.6. Het corrigeren van teksten

Het is ondenkbaar dat in teksten die worden ingetoetst geen typefouten zullen worden gemaakt. Uiteraard is het nodig dat deze op eenvoudige wijze kunnen worden gecorrigeerd. Het veranderen van een enkele letter of cijfer kan door stand komen door de cursor op dit teken te plaatsen en de juiste toets in te drukken. We kunnen echter ook karakters invoegen en verwijderen. We zullen dat demonstreren met de zin

MSX personal home computter.

Toets deze zin daadwerkelijk in! We zullen deze veranderen in:

De MSX home computer.

Eerst de t die teveel in "computter" staat. We plaatsen de cursor op deze t en drukken op de toets DEL (van delete, schrappen). Deze t verdwijnt dan en de overige letters schuiven op om het gat dat zou ontstaan te vullen. Nu plaatsen we de cursor op de h van "home" en drukken op de toets BS (Back Space). U ziet de woorden "home computer" opschuiven naar links. Houdt deze toets ingedrukt totdat het woord "personal" is verdwenen (voorzichtig, het repeteren gaat snel!). Het resultaat is:

MSX home computer.

Nu nog het woord "De" invoeren. Daartoe wordt de cursor op de M van "MSX" geplaatst en de toets INS (van insert, invoegen) ingedrukt. Als u het woord "De", gevolgd door een spatie, intoetst, dan ziet u de hele zin naar rechts opschuiven bij elke letter die wordt ingetoetst.

De *insert* mode kan worden verlaten door opnieuw de toets INS in te drukken.

2.7. De 40 en 32 kolommode

Het gedeelte van het scherm waarop u met de toetsen karakters kunt plaatsen hebben we het veld genoemd. Met de onderste regel, waarop de functies van de functietoetsen staan vermeld meegerekend, is het veld 24 regels hoog. Horizontaal kunnen we 37 tekens op een regel plaatsen. Omdat de tekens op de regels steeds recht boven elkaar staan spreken we ook wel van (verticale) kolommen. De werkelijke grootte van het veld kunnen we echter niet zien omdat het kader en het veld steeds dezelfde kleur hebben. In werkelijkheid is het veld breder dan het gedeelte dat we kunnen beschrijven. Het veld is opgebouwd uit 40 verticale kolommen en 24 regels. Dat wil zeggen, dat is de situatie na het inschakelen van de computer. Toets nu eens in:

screen 1 (RETURN)

Dus eerst screen 1 intypen en daarna de RETURN toets indrukken. U ziet nu duidelijk de situatie op het scherm veranderen. Ten eerste is het scherm "gewist". Ten tweede kunt u nu slechts 29 karakters op een regel plaatsen. Het aantal regels is echter gelijk gebleven, in totaal 24. Toets nu in:

color 15,5,15 (RETURN)

Nu kunt u duidelijk het veld en het kader van elkaar onderscheiden.

Het derde kleurnummer achter COLOR heeft in deze mode dan ook inderdaad effect. Het hangt van de afstelling en van de centrering van uw televisie af of u de linker- en/of rechterzijkant van het veld kunt zien. In elk geval kunt u zien dat de breedte van het veld groter is dan het gedeelte dat u kunt beschrijven. In de breedte gaan nu 32 kolommen, het scherm is in de *32 kolommode*. Toets eens in:

screen 0 (RETURN)

Opnieuw is het scherm gewist. De situatie is weer zoals we die gewend waren. Het scherm is terug in de *40 kolommode*.
Het is overigens heel goed mogelijk om in de 40 ko-

lommode ook 40 karakters op één regel op het scherm te krijgen. Toets maar eens in:

width (40) (RETURN)

Dit kan uiteraard alleen maar als het scherm ook in de 40 kolommode staat. U kunt nu het scherm over de gehele breedte beschrijven. Met het commando WIDTH(X) kunt u de breedte van het *beschrijfbare* gedeelte van het scherm op elke waarde tussen 1 en 40 brengen door voor X een getal tussen 1 en 40 in te vullen. Probeer ook eens:

screen 1: width (32) (RETURN)

U ziet dat u ook nu het scherm over het gehele gedeelte kan beschrijven.

3. De MSX computer als rekenmachine, de direct mode

3.1. Het PRINT commando

Het is uiteraard de bedoeling dat de computer met hetgeen wat wordt ingetoetst en op het scherm zichtbaar is, ook iets doet. Daarbij moet op zijn minst aan twee voorwaarden worden voldaan: de opdracht die wordt gegeven moet aan de computer bekend zijn (moet tot zijn mogelijkheden behoren) zodat hij hem ook kan uitvoeren en aan de computer moet worden bekend gemaakt of hij de ingetoetste zin in zijn geheugen moet gaan plaatsen en eventueel moet gaan uitvoeren. Voor het tweede gedeelte dient de RETURN toets. Wordt op een bepaalde regel een opdracht ingetoetst dan wordt deze opdracht ingevoerd in de computer op het moment dat de RETURN toets wordt ingedrukt. Wordt de computer gebruikt in de DIRECT mode, zoals in dit hoofdstuk de bedoeling is, dan voert de computer de opdracht ook meteen uit. In elk geval komt de cursor aan het begin van een nieuwe regel te staan. We hebben met de direct mode al in het vorige hoofdstuk kennis gemaakt toen we de computer de opdrachten COLOR en SCREEN gaven. De eerste opdracht in dit hoofdstuk is

```
let A=3
```

Dit wil het volgende zeggen: reserveer een aantal geheugenplaatsen en noem deze A; plaats het getal 3 in deze geheugenplaatsen. De letter A wordt hier een variabele genoemd. Aan deze variabele kan elke willekeurige waarde worden gegeven, in dit geval wordt aan de variabele A de waarde 3 toegekend. Druk nu de toets RETURN in. Op het scherm verschijnt nu de tekst Ok. Dit houdt in dat de computer de opdracht heeft uitgevoerd en een nieuwe opdracht kan ontvangen. Toets nu eens in:

```
let A=10: print A (RETURN)
```

Hier worden twee opdrachten op één regel gegeven, LET en PRINT.
Deze *moeten* daarom worden gescheiden door het teken ":".
De PRINT opdracht houdt in dat de computer iets op het scherm moet laten zien, in dit geval de *waarde* die aan de variabele is gegeven. De tekst op het scherm ziet er zo uit:

```
let A=10: print A
 10
Ok
```

Probeer nu eens:

```
let A=15: print"A" (RETURN)
```

U ziet, nu wordt niet de *inhoud* van A op het scherm geschreven, maar gewoon de letter A. De computer zal alles wat na een PRINT opdracht tussen aanhalingstekens staat letterlijk op het scherm schrijven. Nu de volgende regel:

```
let A=5: print"A=",A (RETURN)
```

Let op de komma in de PRINT opdracht. Het resultaat is:

```
A=           5
```

Vervang de komma door ";"

```
let A=5: print"A=";A (RETURN)
```

Er is nu slechts één spatie tussen A= en 5.
In de tekst zullen we in het vervolg het (RETURN) weglaten. U kunt deze toets toch niet vergeten te gebruiken, omdat de computer gewoon niets doet als u hem niet indrukt. Het volgende gaan we proberen:

```
a=5:?"A=";a
```

De computer reageert hierop hetzelfde als op het vorige voorbeeld. Het woord LET kan dus worden weggelaten en voor PRINT kan een vraagteken worden gebruikt. Dit is een sterk verkorte manier voor het invoeren van deze opdrachten. In dit boek

zal verder LET steeds worden weggelaten, maar voor PRINT zal immer de onverkorte tekst worden toegepast.

3.2. De rekenkundige bewerkingen

Gaat u eens na wat het resultaat is van de volgende regel.

print 8+2; 8−2; 8*2; 8/2; 8^2; 8^(1/3)

Het resultaat laat niet alleen zien dat na een PRINT opdracht een berekening kan volgen die door de computer wordt uitgevoerd, maar ook dat meerdere berekeningen op één regel kunnen worden geplaatst die ook worden uitgevoerd, en wel in de volgorde waarin ze op de regel voorkomen. Op deze regel ziet u ook de beschikbare bewerkingen:

+	optellen
−	aftrekken
*	vermenigvuldigen (asterisk)
/	delen
X^Y	machtsverheffen

Als u niet gewend bent om te rekenen met de computer zult u misschien het teken ''*'' als vermenigvuldigingsteken wel wat vreemd vinden.

Probeer ook eens

print 8:2

U ziet dat het wat meer vertrouwde teken '':'' voor een deling hier niet gebruikt kan worden. De computer gebruikt dit teken als scheidingsteken tussen de opdrachten op één regel. Als deelteken dient u dus ''/'' te gebruiken. De machtsverheffing 8^2 wordt als volgt ingevoerd: 8^2. Voor het trekken van de (derdemachts)wortel $\sqrt[3]{8}$ wordt gebruik gemaakt van de eigenschap

$$\sqrt[3]{8} = 8^{1/3}$$

Dat haakjes zijn gebruikt in 8^(1/3) houdt verband met de eigenschap dat ook hier een bepaalde volgorde wordt aangehouden bij het uitwerken van bepaalde berekeningen, indien daarin een aantal verschillende bewerkingen worden toegepast. Deze volgorde luidt:

1. Machtsverheffen, van links naar rechts:
   2^3^4 = 8^4 = 4096

2. Vermenigvuldigen en delen, ook van links naar rechts:
   5*8/2*4 = 40/2*4 = 20*4 = 80

3. Optellen en aftrekken, ook al van links naar rechts:
   40 − 8 + 30 − 10 = 32 + 30 − 10 = 62 − 10 = 52

Hierbij gaan de bewerkingen tussen haakjes voor op andere bewerkingen. Probeer eens:

print 3*5^8/2; (3*5)^(8/2)

Er zijn in de computer een aantal functies ingebouwd die belangrijk zijn voor het rekenen.

SIN(X)	Sinus
COS(X)	Cosinus
TAN(X)	Tangens
ATN(X)	Arctangens
EXP(X)	e^x
LOG(X)	ln X
SQR(X)	$\sqrt{X}$

Voor de goniometrische functies sinus, cosinus en tangens moet de waarde van X worden ingevoerd in radialen. Wilt u de goniometrische waarden weten van een hoek van 60° dan moet hiervoor PI/3 worden gebruikt, waarin PI ongeveer 3.14 is. Nu kunt u de waarde van PI met een zeer grote nauwkeurigheid door de computer laten bepalen. Toets daarvoor eens in:

print 4*atn(1)

U ziet dat er nog al wat cijfers achter de komma worden gegeven! De computer zelf rekent met getallen die maximaal nog één cijfer meer achter de komma hebben dan hij op het scherm laat zien.

Toets de volgende regel eens in:

pi = 4*atn(1): a = sin(pi/3): b = cos(pi/3):
print pi/3; a; b; tan(pi/3); atn(a/b)

U zult zien dat als de waarde van de tangens van een hoek is gegeven (A/B), de functie ATN(X) de grootte van de hoek uitdrukt in radialen (PI/3).

Dat hier haakjes zijn gebruikt heeft geen betrekking op de volgorde van de bewerking, maar simpelweg omdat bij functies *altijd* haakjes moeten worden toegepast waartussen de variabelen zijn geplaatst.

```
x = log(5):
print x; exp(x); exp(log(4)); exp(1)
```

Voor het gemak zijn steeds eenvoudige getallen ingevoerd. Uiteraard is het ook mogelijk grotere en gebroken getallen in te voeren. Probeer:

```
print 0.00015; 150000000000000
```

Het zal zo langzamerhand wel geen nieuws voor u zijn dat bij gebroken getallen inplaats van een komma, voor het aangeven van de decimalen een punt wordt gebruikt.
Zoals u ziet gaat de computer over op de ''wetenschappelijke notatie'':

```
1.5e-4   1.5e+14
```

Dit is de schrijfwijze voor $1.5*10^{-4}$ en $1.5*10^{14}$.

Per variabele wordt een maximum aan geheugenplaatsen gereserveerd zodat ook aan de grootte van een getal grenzen moeten worden gesteld.
Bij gewone notatie:
kleinste getal 0.01; grootste getal
99 999 999 999 999.
Bij wetenschappelijke notatie:
kleinste getal 9.9999999999999e-64, grootste getal 9.9999999999999e+62.
Dit zijn positieve getallen. Ook bij de negatieve getallen vinden we deze uitersten.

De nauwkeurigheid waarmee gerekend wordt is zeer groot. De computer rekent met maximaal 15 cijfers achter de komma en laat daarvan maximaal 14 cijfers zien. Dit wordt de *dubbele precisie* genoemd. Wilt u geheugenruimte besparen dan kunt u ook in de *enkele precisie* werken. Daarvoor dient de opdracht CSNG(X):

```
print 1/7: print csng(1/7)
```

U ziet dat in de enkele precisie niet meer dan 6 decimalen achter de komma worden getoond. De computer rekent echter met maximum 7 decimalen.

4. Programmeren in BASIC, program mode

4.1. Het eerste programma

Zoals u hebt gemerkt, hoeft u niets bijzonders te doen om de computer in de direct mode te laten werken. U hoeft ook niets bijzonders te doen om de computer in de program mode te laten werken. In dat geval wordt een programma afgewerkt. In welke mode de computer gaat werken, wordt bepaald door de wijze waarop de opdrachten worden ingevoerd. Toetst u de volgende regel eens in:

new (RETURN)
10 a=a+1 (RETURN)
20 print a (RETURN)
30 goto 10 (RETURN)
run (RETURN)

Na het laatste RETURN ziet u op het scherm een reeks van getallen verschijnen, onder elkaar geschreven en steeds met 1 oplopend. De computer is nu bezig een programma te ''doorlopen'' waar geen eind aan schijnt te komen. Slechts als de waarde van A de grootte heeft bereikt die hij maximaal kan aannemen, stopt de computer en meldt ''OVERFLOW''. U zult wel geen zin hebben om daarop te blijven wachten. We laten daarom het programma stoppen. Nu kan dat door de computer stomweg uit te schakelen en later weer aan te zetten. Als we het zo doen dan kunnen we er zeker van zijn dat het programma niet meer in het geheugen aanwezig is. Een andere methode is die waarbij we gebruik maken van een RESET knop (indien op de computer aanwezig).

Ook dan zijn we echter het programma uit het geheugen kwijt en de RESET knop mag dan ook alleen maar als laatste redmiddel worden gebruikt (indien de computer zodanig in de war is dat hij door niets meer te stoppen is). Om het programma echt te laten stoppen hebben we de STOP toets. Druk deze in en u ziet dat het programma inderdaad is gestopt. U bent echter nog ''in'' het programma. U kunt met het toetsenbord geen commando's geven. Het toetsenbord is nog uitgeschakeld, zoals dat steeds bij het afwerken van een programma het geval is. Drukt u opnieuw op de STOP toets dan gaat het programma weer verder op de plaats waar het gestopt is (probeer alles zelf ook uit!). Om ''uit'' het programma te komen moet de CTRL toets en tegelijk daarmee de STOP toets worden ingedrukt.

Van elke regel van het ingevoerde programma zal een verklaring worden gegeven.

new (RETURN)

Hiermee wordt aangegeven dat een nieuw programma wordt ingevoerd. Dit heeft tot gevolg dat een programma dat al eerder in de computer is ingevoerd, wordt ''gewist''. Wordt een nieuw programma ingevoerd ''over'' een oud programma dat nog in het geheugen aanwezig is dan kan dat vreemde resultaten geven. Voor het eerste programma na het inschakelen van de computer is NEW dus eigenlijk niet nodig. Het beste is echter er een gewoonte van te maken steeds NEW in te toetsen. In dit boek zal deze opdracht in het algemeen niet meer worden gebruikt.

10 a=a+1 (RETURN)

Dit is de eerste programmaregel. Een programmaregel *moet* altijd voorzien zijn van een regelnummer, hier 10. De computer maakt hieruit op dat hij in de program mode moet werken. Het RETURN aan het eind heeft dan als resultaat dat de computer de regel opslaat in zijn geheugen maar hem *niet* verwerkt. De regelnummers kunnen hele getallen zijn, te beginnen bij 0 en oplopend met 1.

Dit is niet gebruikelijk. Door de regelnummers bijvoorbeeld met 10 te laten oplopen, zoals hier, kunnen later, indien nodig, nog enige regels worden tussengevoegd. De volgorde waarin de regels worden ingetoetst is onbelangrijk. De computer plaatst ze zelf in de goede volgorde, probeer maar (als het programma nog loopt moet u het eerst stoppen):

```
new
30 goto 10
10 a=a+1
20 print a
run
```

Zoals ook in het vorige hoofdstuk is afgesproken zullen we verder in het algemeen het (RETURN) achter de programmaregels weglaten. U mag het echter nooit vergeten. De regel wordt pas ingevoerd door het indrukken van de RETURN toets. Als u op deze toets drukt hoeft de cursor niet per se op het eind van de programmaregel te staan. De cursor mag zelfs aan het begin van die regel staan. Alle tekst die op de schermregel staat waarop de cursor is wordt als tekst voor de programmaregel beschouwd, soms zelfs ook de tekst op de volgende schermregel. Daarom moet een programmaregel steeds op een *lege* schermregel worden geschreven. Als een programmaregel is ingetoetst wordt hij ingevoerd met de RETURN toets, waar de cursor zich op deze regel dan ook mag bevinden. Pas daarna mag de volgende programmaregel worden ingetoetst op de schermregel waar de cursor na RETURN is gekomen.
Het zal u misschien opgevallen zijn dat de opdrachten en de variabelen (zoals A) steeds met kleine letters worden ingetoetst. Dit is eigenlijk niet juist. De opdrachten en de variabele namen *moeten* met hoofdletters worden geschreven. Worden ze echter met kleine letters ingevoerd dan maakt de computer zelf daar hoofdletters van.
In de tekst zal dan ook steeds hiervoor (als het goed is) hoofdletters worden gebruikt.

Regel 10 van het programma is een LET opdracht en alles wat hierover onder de direct mode is gezegd, geldt ook hier. In deze regel wordt de waarde van de variabele A bij elke doorgang met 1 opgehoogd. Dan moet deze variabele om te beginnen dus al een waarde hebben. Zodra in een programma een variabele wordt benoemd (hier met A) wordt hieraan door de computer bij RUN de waarde 0 toegekend.

```
20 print a (RETURN)
```

Alles wat bij de direct mode is verteld over de opdracht PRINT is ook hier van toepassing.

```
30 goto 10 (RETURN)
```

Deze opdracht wil alleen maar zeggen: vervolg het programma met regel 10.
Dit is een sprongopdracht. Er kan naar elke willekeurige regel in het programma worden gesprongen, zodat elke willekeurige regelnummer (als die tenminste in het programma aanwezig is) na "goto" kan worden ingevoerd. Deze (achterwaartse) sprong heeft als resultaat dat A steeds met 1 wordt opgehoogd en wordt geprint. Het is dus deze opdracht die de oorzaak is dat het programma zich steeds maar herhaald tot een "OVERFLOW" het laat stoppen. Probeer dit eens (geen NEW, stop eventueel het programma):

```
30 goto 20
run
```

Dit is een fout die bij het programmeren nogal eens voorkomt, het springen naar een verkeerde regel!

Wat u zal opvallen is dat de oorspronkelijke regel 30 is vervangen door een regel met een nieuwe inhoud. Het veranderen van een regel kan tot stand komen door eenvoudig de regel opnieuw in te toetsen met een nieuwe tekst. Een regel kan ook eenvoudig uit het programma worden verwijderd. Probeer dit maar eens:

```
30 (RETURN)
run
```

Let op het verschijnen van "Ok". De computer geeft hiermee aan dat hij klaar is met het afwerken van het programma en klaar staat voor het ontvangen van nieuwe opdrachten. Maak het programma weer compleet met

```
30 goto 10
```

(Er kan niets fout gaan als u elke aanwijzing in dit hoofdstuk nauwgezet opvolgt). We hebben nu al een paar keer het woord "goto" moeten intoetsen, zodat het tijd wordt eens te wijzen op het gemak van toets f3. Als u eerst het regelnummer "30" intoetst en daarna de toets f3 dan staat "30 goto" al op het scherm en hoeft u verder alleen nog maar "10" in te toetsen.

De laatste regel die nog besproken moet worden hoort eigenlijk niet tot het programma en heeft dan ook geen regelnummer:

run (RETURN)

Hiermee wordt het programma gestart, dat wil zeggen: de computer krijgt de opdracht het programma te verwerken (uit te voeren), te beginnen met de regel met het laagste nummer. Daarna wordt het programma regel voor regel afgewerkt, in volgorde van het regelnummer. Het programma kan met elke willekeurige regel worden gestart, probeer maar:

run 20 (RETURN)

en ook

run 30 (RETURN)

De computer gaat nu het programma afwerken, te beginnen met de regel die na RUN is vermeld. Tracht eens aan de hand van het programma na te gaan wat het eerste getal is dat de computer laat zien. Het zal lastig zijn dit op het scherm te controleren. We zullen het programma wat moeten vertragen. Voer daarvoor de volgende regels in (geen NEW):

```
22 b=0
24 b=b+1
26 if b<50 then 24
```

RUN het programma en zie het resultaat. Probeer nu ook RUN 20, RUN 30, RUN 22 en RUN 24. Merk op dat de laatste twee geen verschil te zien geven. Dit komt omdat na RUN B 0 is gemaakt door de computer.
Hoe is nu de werking van de regels 22 tot en met 26? Het nieuwe hieraan is regel 26. Ook dit is een sprongopdracht, maar er is een verschil met GOTO. GOTO is een *onvoorwaardelijke* sprongopdracht. In regel 26 is een *voorwaardelijke* sprongopdracht gegeven (IF . . . THEN) die als volgt vertaald kan worden: als B kleiner is dan 50 spring dan naar regel 24. De sprong naar regel 24 wordt alleen uitgevoerd als aan de voorwaarde B<50 wordt voldaan. Wordt niet aan de voorwaarde voldaan (B wordt op een gegeven moment 50) dan wordt het programma vervolgd met de eerstvolgende regel, in dit geval regel 30. In de regels 24 en 26 is een teller geconstrueerd en de vertraging wordt verkregen omdat elke handeling die de computer moet verrichten, tijd kost.

Hier moeten dezelfde handelingen vijftig keer worden verricht. Een langere of kortere vertraging wordt verkregen door het getal 50 in regel 26 groter of kleiner te maken.
Steeds als het programma opnieuw moet worden doorlopen (regel 30 GOTO 10) moet de teller B op nul worden gezet. De computer doet dat maar één keer, bij RUN, vandaar dat regel 22 is toegevoegd. Verwijder regel 22 door

22 (RETURN)

en RUN het programma nogmaals. U ziet het resultaat hiervan. Verander regel 26 eens in

26 if b<50 then 22

(voeg regel 22 weer toe) en u ziet het resultaat van een veel gemaakte fout.
Voor de voorwaardelijke sprongopdracht IF . . . THEN zijn een aantal voorwaarden mogelijk:

X<Y	X is kleiner dan Y
X>Y	X is groter dan Y
X<=Y	X is kleiner dan of gelijk aan Y
X>=Y	X is groter dan of gelijk aan Y
X<>Y	Xis ongelijk aan Y

X en Y kunnen variabelen zijn, maar er kan in de voorwaarde ook een berekening voorkomen, bijvoorbeeld:

IF X^(½)<EXP(Y) THEN 200

Voor de teller in de regels 22 tot en met 26 is er een veel eenvoudiger opdracht:

26 for b=0 to 50: next b

Verwijder regel 22 en 24 en RUN het programma. Doe dat deze keer eens door eenvoudig de toets f5 in te drukken!
Het resultaat van dit programma zal ongeveer gelijk zijn aan hiervoor. Ook hier is een variabele B waarvan de inhoud steeds met 1 wordt opgehoogd (door NEXT B) en die telt van 0 tot en met 50. De vertraging is echter kleiner dan in het voorgaande voorbeeld.
Verander regel 25 daarom in:

26 for b=0 to 200: next b

Let op de dubbele punt voor NEXT. Dit is het scheidingsteken tussen de FOR . . . TO opdracht en NEXT. RUN het programma (met de f5 toets). Probeer nu eens:

```
26 for b=0 to 200 step .5: next
```

De vertraging is nu twee keer zo groot omdat de teller nu met 0.5 wordt opgehoogd in plaats van met 1. Wordt de grootte van de stap niet vermeld dan wordt de teller steeds met 1 opgehoogd. Als er geen verwarring mogelijk is (zoals dat zou kunnen gebeuren bij meerdere tellers) dan kan de variabele bij NEXT worden weggelaten. In plaats van NEXT B dus gewoon NEXT. Er zijn ook nog andere mogelijkheden:

```
26 for b=200 to 0 step -.5: next
```

Hier hebben we te maken met een *aflopende* teller.

We kunnen nu nog een paar variaties maken met de opdracht PRINT. Voer daartoe de volgende regel in:

```
22 PRINT
```

en RUN het programma. Er wordt nu steeds een regel overgeslagen!
Als u nu zo langzamerhand niet meer weet wat voor programma in de printer is opgeslagen dan stopt u het programma en typt

```
list (RETURN)
```

Het programma, zoals dat in de computer is opgeslagen, wordt netjes op het scherm uitgetypt.
Verwijder regel 22 en RUN het programma eerst met

```
20 print a,
```

en daarna met

```
20 print a;
```

Het verschil zal u duidelijk zijn als u zich herinnert wat in hoofdstuk 3 over de tekens "," en ";" is geschreven. Deze tekens worden "separators" (scheidingstekens) genoemd. De volgende samenvatting kan hierover worden gegeven:

a. Wordt een PRINT opdracht niet afgesloten met een scheidingsteken dan wordt de volgende PRINT opdracht begonnen aan het begin van de volgende regel. Dat wil zeggen dat de nu onzichtbare cursor naar het begin van de nieuwe regel gaat.
b. Wordt een PRINT opdracht afgesloten met een separator dan wordt de volgende PRINT opdracht op dezelfde regel geschreven (voorzover de ruimte op de regel dit toelaat) en wordt er een ruimte open gelaten (aantal spaties) afhankelijk van de soort separator.
c. Een dubbele punt geeft het eind van een opdracht aan, dus ook het einde van een PRINT opdracht.
d. De komma verdeelt de regels in twee kolommen voor getallen die korter zijn dan elf karakters, inclusief de decimaal punt. Zijn de getallen langer dan elf karakters dan worden ze onder elkaar geschreven. De punt-komma zorgt voor één spatie vóór en één spatie achter een getal, zodat zich tussen twee getallen steeds twee spaties bevinden.

RUN het programma nogmaals en stop met de CTRL en STOP toetsen, (de computer meldt: "break in . . ."). Type nu

```
cont (RETURN)
```

Het programma gaat nu verder waar het was gestopt! Om volledig te zijn betreffende de LIST opdracht, probeer eens het volgende:

```
list -20    (RETURN)
list 20-    (RETURN)
list 20     (RETURN)
list 20-26  (RETURN)
```

MSX BASIC kent nog een andere LIST opdracht:

```
list.
```

Let op de punt achter LIST. Het resultaat hiervan is dat alleen de laatste regel van het programmadeel dat al eerder op het scherm is getoond, wordt gelist. Ook met de LIST opdracht kunt u het zich makkelijk maken. Als u de toets f4 indrukt dan wordt het woord "list" op het scherm geschreven dat u eventueel nog kan aanvullen. Ook voor "LIST." is een toets, f9. Hetzelfde geldt voor

RUN (RETURN) waarvoor u de toets f5 of f10 kunt gebruiken. Ten slotte kan de toets f8 worden gebruikt voor CONT (RETURN).

4.2. Benamingen en variabelen

Er zijn nu in kort bestek een aantal regels opgesomd die belangrijk zijn voor het samenstellen van programma's, waarbij is gesproken van ''opdrachten''. We zullen deze opdrachten op de juiste manier benoemen.
De opdrachten zijn in drie categorieën onder te brengen.

a. Commando's
   Commando's hebben betrekking op een programma maar zijn er zelf geen onderdeel van. Voorbeelden: NEW, LIST, CONT.
b. Statements
   Dit zijn de opdrachten die de computer bepaalde bewerkingen laat uitvoeren en daardoor het verloop van het programma bepalen. Ze hebben betrekking op variabelen, functies en regelnummers. Eventueel kunnen ze de inhoud van een variabele veranderen. Voorbeelden: LET, PRINT, GOTO, IF . . . THEN, FOR . . . TO en NEXT.
c. Functies
   Het resultaat van een functie is een waarde die afhankelijk is van de inhoud van een variabele. De inhoud van de variabele zelf verandert niet. In de functie wordt de variabelenaam steeds tussen haakjes geplaatst. Voorbeelden: SIN(X), COS(X), TAN(X) en ATN(X).

Bij de inhoud van variabelen hebben we ons alleen beziggehouden met getallen. Omdat in de geheugenplaatsen die voor een variabele gereserveerd worden steeds codegetallen worden geschreven (hoofdstuk 1) zijn er verschillende variabelen mogelijk.

a. Floating point variabelen
   Deze kunnen alle soorten van getallen als inhoud hebben, dus ook gebroken getallen. Ze worden benoemd met een letter (A, B, . . . Z), twee letters (AA, AB, . . . ZZ) of een letter en een cijfer (A0, A1, . . . Z9). U kunt geen combinatie van twee letters gebruiken die gelijk is aan de eerste twee letters van een commando, statement of functie. De computer denkt dan met zo'n opdracht te doen te hebben, in plaats van een variabele.
b. Integer variabelen
   Deze kunnen slechts gehele getallen bevatten. Ze worden benoemd als de floating point variabelen gevolgd door het teken % (A%, AC%, D5%).
c. Stringvariabelen
   Deze kunnen een reeks van karakters bevatten, zoals letters, cijfers, grafische karakters enz. Ze worden benoemd als de floating point variabelen gevolgd door het teken $ (X$, PX$, F5$). Voorbeeld: A$ = ''MSX computer''. Hier zijn dus aanhalingstekens nodig.

Het is wel eens makkelijk om in een programma de variabelen een volledige naam te geven. In een programma voor een huishoudboekje kunt u dan een variabele ''INKOMEN'' noemen en een andere ''UITGAVEN'' De computer accepteerd dat, maar leest alleen de eerste twee letters.
Als u een tweede variabele met ''INKOPEN'' benoemt dan zal dat moelijkheden geven met ''INKOMEN'' omdat de computer daar geen verschil in ziet.
Probeer de volgende programma's

```
10 A%=5.15
20 B%=3
30 C=A%*B%
40 PRINT C
```

Dit gaat niet helemaal goed, de waarde in A% is 5 in plaats van 5.15.

```
10 A=5.15
20 A%=3
30 C=A*A%
40 PRINT "A*A%=";C
```

Let in dit programma op de spatie tussen A*A% = en het getal dat voor C wordt geprint. Achter een tekst geeft de computer bij een puntkomma als scheidingsteken geen spatie en vóór een getal slechts één spatie. De spatie achter het getal is niet waarneembaar.

```
10 AA$="com"
20 B5$="puter MSX "
30 B$="pleet"
40 PRINT AA$;B5$;AA$+B$
```

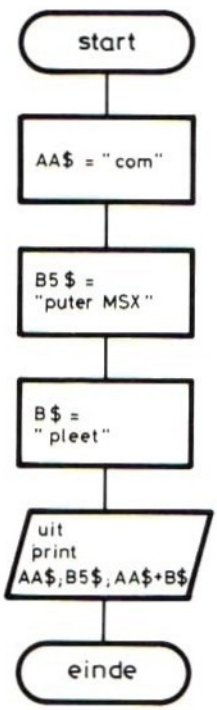
Vergeet bij het intoetsen van dit programma vooral de aanhalingstekens niet. In regel 40 heeft het ''+'' teken hetzelfde effect als het scheidingsteken '';''. In regel 20 is de spatie achter MSX ingepro-

grammeerd. Dit is nodig omdat bij het printen van karakterstrings de computer geen spatie invoert, ook niet als er in de string getallen voorkomen (tussen de aanhalingstekens).

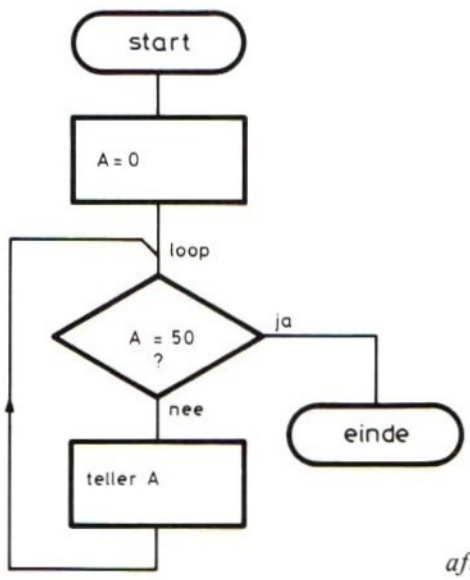
4.3. Basisconstructies

Ons eerste programma stelde goed beschouwd niet zoveel voor. Een groot bezwaar was wel dat het maar (bijna) onbeperkt doorging, "er kwam geen einde aan". De hoofdoorzaak was hier het statement GOTO dat het programma steeds maar weer opnieuw liet beginnen. Dit soort slordigheden kunnen we voorkomen als we ons naar een aantal grondregels richten. Om deze beter te kunnen begrijpen maken we gebruik van een *blokschema*. In afb. 2 ziet u het blokschema van het laatste programma van paragraaf 4.2. Hierin is elke programmaregel door een blok weergegeven. De "blokken" "START" en "EINDE" dienen slechts om het begin en het eind van het programma aan te geven. U ziet dat in de afbeelding de blokken verschillend van vorm zijn. Met uitzondering van die voor "START" en "EINDE" is de algemene vorm vaak een rechthoek, behalve als het een uitvoer betreft (zoals hier naar het beeldscherm) of een invoer. Dan heeft het blok de vorm van een parallellogram. De constructie van deze afbeelding wordt een *"sequentie"* genoemd.
Het zal duidelijk zijn dat de volgorde waarin het programma wordt doorlopen van boven naar beneden is. Deze *stroomrichting* zou kunnen worden aangegeven door middel van pijlen in de verbindingslijnen tussen de blokken. Bij deze stroomrichting is dat echter niet nodig. Is de stroomrichting van beneden naar boven dan zal hij steeds door pijlen worden aangegeven. Een blokschema van een programma wordt ook wel een *"stroomdiagram"* genoemd.

Een wat uitgebreider programma kan vaak in gedeelten worden gesplitst die elk weer als een klein programmaatje zijn op te vatten. De werking van elk zo'n programmablok wordt dan in een symbool (rechthoek, parallellogram) geplaatst in de volgorde zoals ze in het programma voorkomen. Dat betekent dus dat elk blok in een stroomdiagram niet slechts één commando of statement van een programma hoeft te omvatten, maar ook wel betrekking kan hebben op een groot aantal opdrachten.
Voor het volgende programma keren we weer terug naar ons eerste programma van de vorige paragraaf.
Dit wordt "teller A" genoemd. Laten we aannemen dat deze teller tot 50 moet tellen en niet verder. We kunnen dan de constructie in afb. 3 toepassen. We komen hierin een nieuw symbool te-



afb. 2



afb. 3

gen, de ruit. Deze wordt gebruikt als een voorwaarde moet worden gesteld. De voorwaarde is hier "A = 50". Dit programmadeel heeft de *label* "LOOP" meegekregen. Dit kan worden beschouwd als een adresaanduiding.
De werking is als volgt: eerst wordt de variabele A nul gemaakt. In het volgende blok wordt bepaald of aan de voorwaarde A = 50 wordt voldaan. Is dat niet het geval (zoals nu, A = 0) dan wordt het programma voortgezet in de richting waar "NEE" bij staat. In het volgende programmadeel (teller A) wordt A met 1 opgehoogd en zijn inhoud (nu 1) wordt op het scherm getoond). We gaan nu weer terug (*onvoorwaardelijke* sprong) naar LOOP. Daar wordt weer bepaald of aan de voorwaarde is voldaan, enzovoorts, tot A inderdaad 50 is geworden. Nu wordt de lus in het programma verlaten (in de richting van JA) met een *voorwaardelijke* sprong naar einde.
Dit stroomdiagram levert het volgende programma op:

```
10 A=0
20 IF A=50 THEN 60
30 A=A+1
40 PRINT A
50 GOTO 20
60 END
```

Als het zeker is dat dit programma steeds met RUN wordt gestart dan kan uiteraard regel 10 worden weggelaten. Daarna wordt met

IF A = 50 THEN 60

alleen een voorwaardelijke sprong naar regel 60 gemaakt als aan de voorwaarde A = 50 is voldaan. Anders wordt doorgegaan met regel 30. De GOTO in regel 50 zorgt er voor dat het geheel zich herhaalt tot A = 50. Dan wordt de sprong naar regel 60 gemaakt waar we een nieuw statement tegenkomen: END. Daarmee wordt het programma verlaten en krijgen we Ok op het scherm geprint. Het kan ook op deze manier:

```
20 IF A=50 THEN END
```

Breng deze regel in het programma en zie het resultaat!

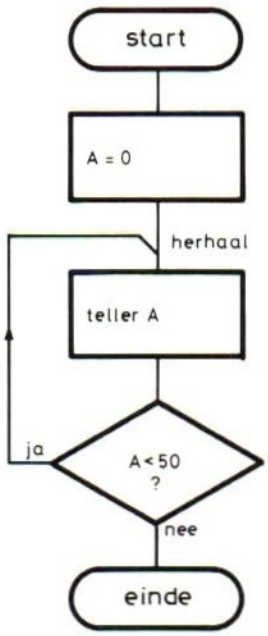
Afb. 4 geeft een tweede methode om het probleem op te lossen. Hier heeft het programmadeel "TELLER A" de label "HERHAAL" gekregen.

Eerst wordt dit doorlopen en daarna wordt bepaald of aan de voorwaarde A < 50 wordt voldaan. Zolang dit het geval is wordt met een voorwaardelijke sprong het programmadeel herhaald tot niet meer aan de voorwaarde wordt voldaan. In dat geval is de laatste werking van het programma het uitvoeren van de statement END.

Het volgende programma is hiervan het resultaat:

```
10 A=0
20 A=A+1
30 PRINT A
40 IF A<50 THEN 20
50 END
```

Merk op dat de volgorde van de programmaregels geheel in overeenstemming is met het verloop van het stroomdiagram. Dit programma is korter dan het voorafgaande en het statement GOTO komt hierin niet meer voor.



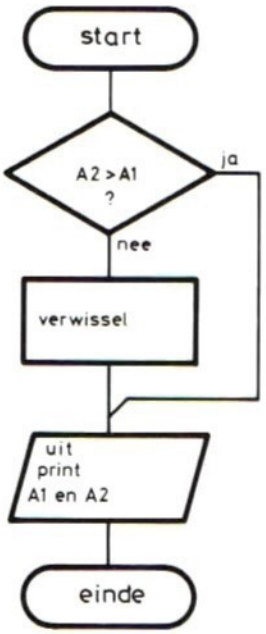
afb. 4

In het volgende voorbeeld worden twee getallen, indien nodig, op de juiste volgorde geplaatst. Het kan gebruikt worden als onderdeel van een sorteerprogramma voor een reeks van getallen die willekeurig door elkaar staan. In dit geval moet de inhoud van de variabele A1 kleiner zijn dan die van A2. Eerst wordt de voorwaarde gesteld A2 > A1 (afb. 5). Is dat niet het geval dan worden A1 en A2

verwisseld en daarna op het scherm geprint. Is dat wel het geval dan wordt naar het programmadeel gesprongen dat het printen verzorgt. Het volgende programma is de uitvoering hiervan:

```
1000 A1=4:A2=3
1010 IF A2>A1 THEN 1030
1020 A=A1: A1=A2: A2=A
1030 PRINT A1,A2
1040 END
```

Zoals u ziet zijn we hier niet met regelnummer 10 begonnen maar met regelnummer 1000. Hoge regelnummers zijn dus ook mogelijk, hoewel niet onbeperkt hoog. Het hoogste regelnummer dat mogelijk is, is 65529.



afb. 5

In regel 1000 wordt de inhoud van A1 4 en A2 3. In deze regel zijn dus twee statements (LET) achter elkaar geplaatst, gescheiden door ":". Het is dan ook heel goed mogelijk om meerdere - verschillende - statements na elkaar te plaatsen op een regel, mits ze steeds worden gescheiden door het teken ":". Indien de programma's daardoor minder overzichtelijk worden kan het beter worden nagelaten. Aan de voorwaarde in regel 1010 wordt niet voldaan zodat wordt doorgegaan naar regel 1020. In deze regel wordt verwisseld. Hiervoor is de "hulpvariabele" A nodig. Deze krijgt de waarde van A1, A1 krijgt de waarde van A2 en deze op zijn beurt de waarde van A. In regel 1030 worden A1 en A2 op het scherm geprint. Voer nu de volgende regel in:

```
1000 A1=3:A2=4
```

Nu is het verwisselen niet nodig en wordt door "THEN 1030" naar regel 1030 gegaan voor het printen (als u het programma ook "RUNT" tenminste). Als u werkelijk wilt weten of in regel 1010 ook inderdaad de sprong wordt gemaakt voer dan de volgende regel in:

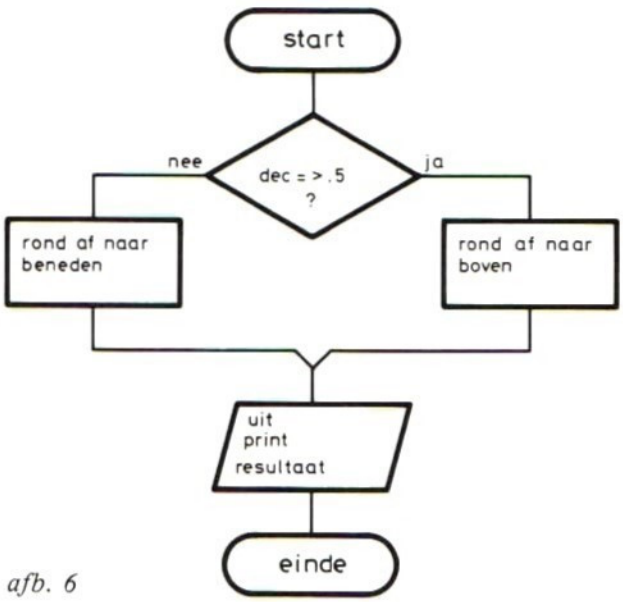
```
1010 IF A2>A1 THEN STOP
```

Na RUN meldt de computer: "Break in 1010". Hieraan kunt u zien dat het statement THEN inderdaad is uitgevoerd! Toets nu in:

cont (RETURN)

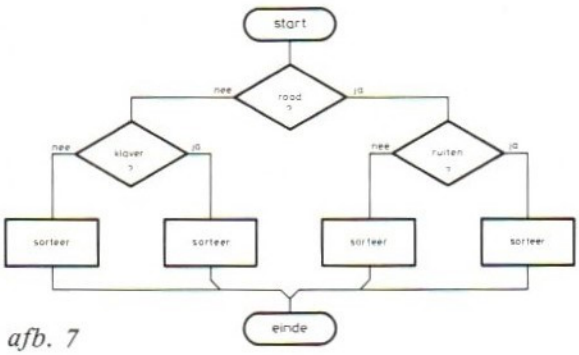
Het programma wordt nu vervolgd en de variabelen worden op het scherm geprint, nu echter in de *verkeerde* volgorde. Dat komt omdat door STOP het programma wordt voortgezet met regel 1020 waar nu een verwisseling plaats heeft die niet plaats had moeten hebben.

In het laatste voorbeeld wordt een voorwaarde gesteld waarna altijd een sprong tot stand komt, echter naar verschillende programmadelen (afb.



afb. 6

6). Deze vorm lijkt zeer veel op de voorgaande maar het verschil zal u duidelijk worden als na de eerste selectie opnieuw een voorwaarde moet worden gesteld. Dit komt bijvoorbeeld voor bij het sorteren van speelkaarten, zie afb. 7. Hier wordt eerst naar rood en zwart gesorteerd. De zwarte kaarten worden daarna in schoppen en klaver gesorteerd en de rode in harten en ruiten. Dan worden de vier stapeltjes naar waarde gesorteerd.



afb. 7

Terug naar afbeelding 6. Hier gaat het om het afronden van een getal. Als het getal achter de komma gelijk is aan 0.5 of groter is dan 0.5 dan moet naar boven worden afgerond, anders moet naar beneden worden afgerond. Het volgende programma maakt dat mogelijk.

```
10 A=12.4
20 I=INT(A): D=A-I
30 IF D=>.5 THEN 100 ELSE 200
100 P=I+1: GOTO 300
200 P=I: GOTO 300
300 PRINT P
310 END
```

De eerste programmaregel vermeldt dat het getal (A) gelijk is aan 12.4. In regel 20 wordt met INT(A) het gedeelte vóór de komma bepaald. We weten dat een integer variabele slechts een geheel getal kan zijn. De INT functie zorgt er voor dat de variabele I slechts met het gedeelte van vóór de komma van A wordt "geladen". Later zal nog op deze functie worden teruggekomen.
Door het verschil van A en I te berekenen (D = A – I) is ook het getal achter de komma bekend. In regel 30 wordt geselecteerd. Is het getal achter de komma gelijk aan of groter dan .5 dan wordt naar regel 100 gesprongen (THEN), anders (ELSE) naar regel 200. In regel 100 vinden we in P het naar boven afgeronde getal, in regel 200 het naar beneden afgeronde getal. In regel 300 komen we weer samen zodat in beide gevallen P wordt geprint.
RUN het programma. Voer daarna de volgende regel in en RUN het programma opnieuw.

```
10 A=12.6
```

5. Het invoeren van de gegevens (data)

5.1. Het INPUT statement

Tot nu toe zijn de gegevens, ook wel DATA genoemd, direct in het programma ingevoerd door het benoemen van de variabelen (A = 5.15, A% = 3, AA$ = "COM"). Dit is één manier voor het invoeren van de data. Er zijn nog twee andere manieren. Voor beiden wordt een apart statement gebruikt. Om te beginnen het INPUT statement. Dit statement wordt toegepast als men buiten het programma data wilt invoeren. Het programma kan dan steeds weer worden gebruikt, zonodig met andere data. Veronderstel dat u bij uw computer nog een floppy disk drive en een printer wilt kopen maar u heeft geen geld genoeg. U besluit een lening af te sluiten en wel een persoonlijke lening waarvoor u maandelijks een rente van 1.2% moet betalen over het bedrag dat u op dat moment nog schuldig bent. De aflossing zal *f* 150,— per maand bedragen. Over een bedrag kleiner dan *f* 25,— behoeft u geen rente te betalen. U wilt weten in hoeveel maanden u het bedrag kunt aflossen, wat het laatste aflossingsbedrag is en hoe groot de totale kosten zijn. De grootte van de bedragen wilt u zelf kunnen invoeren om het programma dat u hiervoor gebruikt ook later nog te kunnen benutten met andere bedragen.

```
10 INPUT BEDRAG:LEEN=BEDRAG
20 INPUT RENTE
30 INPUT AFLOS
40 LEEN=INT(LEEN*(100+RENTE))/100-AFLOS
50 X=X+1
60 IF LEEN>AFLOS THEN 40
70 IF LEEN<25 THEN X=X+1: GOTO 120
80 LEEN=INT(LEEN*(100+RENTE))/100-AFLOS
90 X=X+1
100 IF SGN(LEEN)=1 THEN 70
110 LEEN=LEEN+AFLOS
120 PRINT"laatste aflossing";LEEN
130 PRINT"de totale kosten zijn";
140 PRINT ((X-1)*AFLOS+LEEN)-BEDRAG
150 PRINT "het totaal aantal maanden is";X
```

Het bovenstaande programma kunt u hiervoor gebruiken. U toetst het in en laat het "lopen", of wel u RUNT het programma (denk aan de toets f5). We gaan regel voor regel na wat er gebeurt.
De computer komt niet erg ver. Bij de eerste regel al vraagt hij om data door een vraagteken af te drukken. Dit is de werking van het INPUT statement in deze regel. Nu moet u de grootte van het te lenen bedrag invoeren. In dit voorbeeld toetst u daarom 3000 in (RETURN). Ook nu wordt het ingetoetste pas in de computer gevoerd door RETURN.
De computer schrijft dit bedrag in de geheugenplaatsen voor "BEDRAG" en "LEEN" (INPUT BEDRAG: LEEN = BEDRAG). U ziet dat we de variabelen hier een naam hebben gegeven. De computer zal echter slechts de eerste twee letters van de naam lezen. Nu wordt regel 20 uitgevoerd waarin de computer opnieuw wacht op het invoeren van data, nu de hoogte van de rente per maand. Hiervoor voert u 1.2 in. Op dezelfde manier vraagt de computer in de volgende regel om data. Als u hier 150 invoert dan wordt daarmee de geheugenplaatsen voor AFLOS gevuld. Met deze gegevens gaat de computer aan het werk. In regel 40 berekent de computer de grootte van het bedrag dat u per maand nog verschuldigd bent. Dit moet een bedrag zijn dat wordt afgekort op hele centen. Eerst wordt het bedrag in centen uitgerekend:
LEEN*(100 + RENT). Dit kan een gebroken getal zijn, vandaar dat met de integer functie INT een afkorting plaats heeft. De INT functie kort een gebroken getal namelijk af tot een geheel getal door afronding *naar beneden*. Delen door 100 geeft het maandelijks verschuldigde bedrag in guldens.
Van dit bedrag moet nog de maandelijks te betalen aflossing worden afgetrokken (– AFLOS). In regel 50 wordt het aantal maanden geteld.
Door regel 60 wordt deze berekening herhaald tot het geleende bedrag kleiner is dan de aflossing. Dan wordt eerst nagegaan of nog rente verschuldigd is (regel 70). Is dat het geval dan wordt deze rente berekend. Het bedrag dat nu moet worden afgelost kan daardoor weer groter zijn geworden dan de aflossing. In dat geval wordt in regel 80 een positief getal gevonden voor LEEN. Hoewel zeer onwaarschijnlijk, het bedrag dat in deze regel wordt berekend zou ook nul kunnen zijn.

Dit wordt onderzocht door de functie SGN in regel 100. Deze geeft bij een positief getal in LEEN als resultaat +1, bij LEEN = 0 een 0 en bij een negatief getal in LEEN −1. Regel 110 corrigeert als LEEN gelijk is aan of kleiner is dan de aflossing terwijl in de volgende regels het resultaat wordt afgedrukt.

Na het proberen van dit programma zult u waarschijnlijk nog wat beter met de functies INT en SGN willen kennismaken. Dit kan in hoofdstuk 6.

Het hiervoor gegeven programma is eigenlijk niet zo fraai. Ten eerste maakt de computer niet bekend welk gegeven hij van u verwacht en ten tweede wordt twee keer een eender programmadeel doorlopen (regels 40-50 en 80-90). We veranderen het daarom als volgt:

```
10 INPUT"te lenen bedrag";BEDRAG
20 LEEN=BEDRAG
30 INPUT"relatieve rente";RENTE
40 INPUT"aflossingsbedrag";AFLOS
50 GOSUB 1000
60 IF LEEN>AFLOS THEN 50
70 IF LEEN<25 THEN X=X+1:GOTO 110
80 GOSUB 1000
90 IF SGN(LEEN)=1 THEN 70
100 LEEN=LEEN+AFLOS
110 PRINT"laatste aflossing=";LEEN
120 PRINT"de totale kosten zijn";
130 PRINT ((X-1)*AFLOS+LEEN)-BEDRAG
140 PRINT"het totaal aantal maanden is";X
150 END
1000 LEEN=INT(LEEN*(100+RENTE))/100-AFLOS
1010 X=X+1
1020 RETURN
```

Heel makkelijk bij het invoeren van een programma is de toets f2 (AUTO). Voor u aan het intoetsen begint drukt u deze toets in en daarna RETURN. U ziet dan de eerste regelnummer 10 op het scherm geprint. U kunt nu hierachter de tekst van de regel intoetsen. Na RETURN verschijnt automatisch het volgende regelnummer. Steeds na het invoeren van een regel (door RETURN) verschijnt het volgende regelnummer. Na de laatste regelnummer (150) verschijnt regelnummer 160. U kunt de automode nu verlaten door gelijktijdig de CTRL toets en de toets c in te drukken. De regelnummers 1000 tot en met 1020 toetst u gewoon zelf in.

De eerste verandering in het programma vinden we bij het INPUT statement in de regels, 10, 30 en 40. Net als bij het PRINT statement wordt alles wat hierachter tussen aanhalingstekens is vermeld, letterlijk op het scherm geschreven, nu echter gevolgd door een vraagteken, hetzelfde vraagteken dat ook wordt geschreven als geen tekst achter INPUT is vermeld. En dus vraagt de computer na RUN

te lenen bedrag?

Nu weet u welk bedrag moet worden ingevoerd. De tweede verandering is het GOSUB statement in regel 50. Dit is een onvoorwaardelijke sprongopdracht naar een subroutine, een programmadeel dat meerdere keren in een programma wordt doorlopen, hier begint op regel 1000 en eindigt met RETURN op regel 1020. Een subroutine (ook wel een "onderprogramma" genoemd) moet *altijd* worden afgesloten met dit statement zodat de computer weet wanneer de subroutine is beëindigd zodat hij kan doorgaan met de regel van het "hoofdprogramma" die volgt op de regel waarin hij naar de subroutine werd verwezen. In dit geval gaat hij dus verder met regel 60. Zoals u ziet wordt de subroutine twee maal "aangeroepen" (in de regels 50 en 80). Nadat de subroutine voor de eerste keer is doorlopen wordt dus teruggesprongen naar regel 60. Het programma wordt vervolgd met de volgende regels tot in regel 80 opnieuw naar de subroutine wordt gesprongen. Nadat deze opnieuw is doorlopen zorgt RETURN er voor dat het programma wordt voortgezet met regel 90 van het hoofdprogramma. Het verloop van het gehele programma is daarmee gelijk aan het voorafgaande. U ziet op regel 150 het END statement. Het zal duidelijk zijn dat dit het einde van het programma aangeeft. U zult zich echter afvragen waarom dit niet eerder is toegepast. Bij de vorige programma's volgde na de laatste regel van het programma geen volgende regel meer, zodat de computer zelf wel kon besluiten dat het programma ten einde was. Nu volgen na de laatste regel (regel 140) nog de regels van de subroutine en zonder END zou de computer na regel 140 gewoon doorgegaan zijn met de regels van deze subroutine. Verwijder regel 150 maar eens (150 RETURN) en RUN het programma, dan ziet u het effect.

In programma's worden veel fouten gemaakt (zeer zeker ook bij het gewone intoetsen). Voor het opsporen van deze fouten (debuggen) geeft de computer zelf al de nodige foutmeldingen. Deze zult u beslist nog wel tegenkomen. Voor de fouten

die de computer niet ziet is het STOP statement wel handig. Voer de volgende regel in:

```
65 STOP
```

Na RUN en het invoeren van de data krijgt u de melding BREAK IN 65. Het programma is dan doorlopen tot aan deze regel en of dat goed is gebeurt kunt u vaststellen met

print leen; x

in de direct mode. U onderzoekt dan of de tot aan deze regel gebruikte variabelen de juiste inhoud hebben. Met het commando CONT kunt u het programma laten vervolgen vanaf de regel waar het gestopt was (regel 70). Dat kan niet als u het programma na BREAK hebt gewijzigd. Toets na de melding BREAK IN 65 maar eens in:

```
55 PRINT
```

en geef het commando CONT. U ziet wel hoe de computer hierop reageert. Voor het commando CONT kunt u overigens de toets f8 gebruiken. Na een BREAK kunt u het programma ook opnieuw laten beginnen met RUN. Vergeet dan niet vooraf regel 65 te verwijderen.

Het kan ook zijn dat u in plaats van te lenen, liever wilt sparen. Daarvoor gebruiken we het volgende programma, dat bestemd is voor een spaarrekening waarop u een vast bedrag per maand wilt sparen en waarop eens per jaar rente wordt uitgekeerd. De rente wordt maandelijks berekend en op het eind van het jaar op de rekening bijgeschreven. Als u de bedragen en het aantal jaren dat u van plan bent om te sparen intoetst, dan geeft het programma voor elk jaar het absolute bedrag van de rente en het totaal gespaarde bedrag.

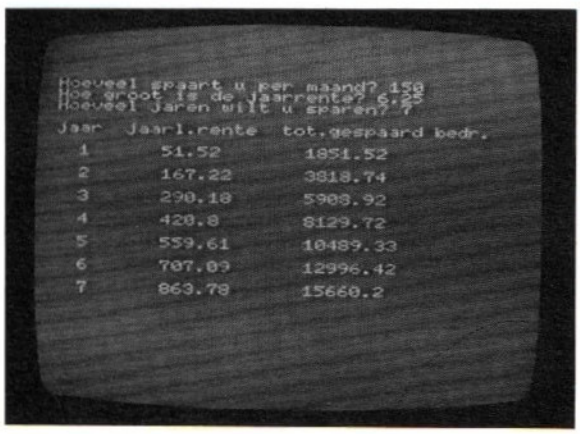
```
10 PRINT "Hoeveel spaart u per maand";
20 INPUT SPAAR
30 PRINT "Hoe groot is de jaarrente";
40 INPUT RENTE
50 PRINT "Hoeveel jaren wilt u sparen";
60 INPUT JAAR: PRINT
70 PRINT "jaar"TAB(6)"jaarl.rente";
80 PRINT TAB(19)"tot.gespaard bedr."
90 PRINT
100 FOR X=1 TO JAAR: JRLRENTE=0
110 FOR Y=1 TO 12
120 JRLRENTE=JRLRENTE+INT(BEDR*RENTE/12)/100
130 BEDR=BEDR+SPAAR
140 NEXT Y
150 BEDR=BEDR+JRLRENTE
160 PRINT TAB(1) X TAB(8) JRLRENTE;
170 PRINT TAB(20) BEDR: PRINT
180 NEXT X
```

Het nieuwe aan dit programma is dat de bedragen in tabelvorm op het scherm moeten komen. Hiervoor wordt de functie TAB te hulp geroepen. Deze functie plaatst de cursor op de plaats die het getal dat bij de functie behoort, aangeeft, te rekenen vanaf het begin van de regel waarop de computer tekst aan het printen is. TAB(6) houdt dus in dat de cursor op plaats 6 op de regel wordt geplaatst (de meest linkse plaats heeft nummer 0). In de regels 70 tot en met 90 wordt de "kop" van de tabel gevormd. De regels 120 tot en met 140 berekenen het gespaarde bedrag en de absolute waarde van de rente. Deze bedragen worden elke maand hoger en als het jaar om is worden ze in de betreffende kolommen geplaatst door de regels 160 en 170.
In dit programma zijn twee lussen toegepast, elk met een teller. Lus 1 wordt gevormd door de regels 110 tot en met 140 en lus 2 door de regels 100 tot en met 180. Lus 1 valt binnen lus 2. Dit hoort "regel" te zijn. De lussen mogen niet "door elkaar" lopen, een foutieve programmering is:

```
100 FOR Y=1 TO 12
110 FOR X=1 TO JAAR: JRLRENTE=0
```

Overigens is ook hier voor de regels 140 en 180 alleen NEXT voldoende (dus zonder de X of de Y). Storend bij dit programma is dat behalve de tekst die voor u belangrijk is ook nog andere zaken op het scherm staan. Voeg de volgende regels in:

```
5 CLS
65 CLS
```



Het statement CLS zorgt er voor dat uw scherm wordt gewist en de cursor aan het begin van de eerste regel wordt geplaatst. Hier wordt dus hetzelfde resultaat verkregen als met de SHIFT-HOME toetsen. RUN nu het programma.

5.2. Het READ statement

Met al dat gedoe over geld kennen we nog steeds de derde manier om data in te voeren, niet. Hierbij wordt de data in een bepaalde regel in het programma ingevoerd en kan daarna niet meer worden veranderd.
Deze data wordt dan gelezen door het READ statement. Als voorbeeld dient het volgende programma.
Veronderstel er zijn zes kinderen die om de beurt op één fiets een blokje rond mogen. U doet net of u de computer de volgorde laat bepalen en voert het volgende programma in:

```
10 CLS
20 FOR X=1 TO 6
30 READ A
40 ON A GOTO 50,60,70,80,90,100
50 PRINT "Jan";: GOTO 110
60 PRINT "Piet";: GOTO 110
70 PRINT "Sonja";: GOTO 110
80 PRINT "Hans";: GOTO 110
90 PRINT "Marit";: GOTO 110
100 PRINT"Ria";
110 PRINT " is ";X: PRINT: NEXT X
120 DATA 1,6,5,3,4,2
```

In hoofdstuk 10 zult u zien dat de volgorde inderdaad door de computer door loting kan worden bepaald. In dit programma is een ''lus'' die zes keer wordt doorlopen en elke keer bij het passeren van regel 30 leest de computer de data in regel 120 en geeft de variabele A de waarde die is gelezen. In dit programma wordt in de eerste doorgang ''1'' gelezen, bij de tweede doorgang ''6'', de derde doorgang ''5'' enz. Nu hadden we na regel 30 zes keer het IF . . . THEN statement kunnen gebruiken, zoals: IF A = 1 THEN PRINT ''Jan'';: GOTO 110 enz. Dit is vervangen door het ON . . . GOTO statement op regel 40. Deze laat de computer springen naar de regel achter GOTO waarvan de plaats door de waarde van A wordt aangegeven, dus A = 1 GOTO 50, A = 6 GOTO 100 enz. Omdat zes keer een ander getal is gelezen wordt ook zes keer een andere naam op het beeldscherm geprint. Voeg nu de volgende regels eens in en RUN het programma daarna.

```
15 FOR Y=0 TO 1
115 NEXT
```

Er had nu twee keer hetzelfde op het scherm moeten worden geprint, maar in plaats daarvan volgt een foutmelding. Er gebeurt dus iets wat niet klopt. Als u na een wijziging eens wilt weten hoe het programma in de computer is opgeslagen, LIST het dan eens (toets f4)! Voeg nu ook de volgende regel toe en RUN:

```
112 RESTORE
```

U ziet, RESTORE geeft de mogelijkheid de data van voren af aan te laten lezen. Met het volgende programma kunt u ook de volgorde kiezen:

```
10 CLS
20 FOR X=1 TO 6
30 READ A$
40 PRINT A$;" is";X: PRINT: NEXT
50 DATA Jan,Piet,Sonja,Hans,Marit,Ria
```

Zoals u ziet kan het READ statement voor elke soort van variabele worden gebruikt en steeds als er wordt gelezen (bij elke doorgang van de lus) wordt het volgende getal of de karakterstring die *tussen* de komma's staat gelezen. Probeer eens het volgende programma:

```
10 FOR X=0 TO 10
20 READ A(X)
30 PRINT "A(";X;")=";A(X)
40 NEXT
50 DATA 10,9,8,7,6,5,4,3,2,1,0
```

Eerst wordt het getal 10 gelezen. In deze eerste doorgang is X 0 zodat het getal 10 in de variabele A(0) wordt geplaatst. Daarna wordt netjes op het scherm geprint:

A(0) = 10

Daarna wordt het getal 9 gelezen, in de variabele A(1) geplaatst (X = 1) en op het scherm geprint. Dit gaat zo door tot alle data is gelezen.
Verander nu de volgende regels:

```
10 FOR X=0 TO 11
50 DATA 11,10,9,8,7,6,5,4,3,2,1,0
```

Doe dat eens zo: LIST regel 10 en *verander* dan met gebruikmaking van de cursor de 0 van de 10 in

FOR X=0 TO 10 in een 1. LIST daarna regel 50 en plaats de cursor op de 1 van het getal 10 in deze regel. Druk nu op de INS toets en voeg ''11,'' in. Na een LIST kunt u er zich van overtuigen dat de regels 10 en 50 inderdaad met de nieuwe inhoud in het programma zijn opgenomen.
Dit programma zal overigens niet ''lopen'', de computer geeft een foutmelding. Tot nu toe werd elke variabele in een programma benoemd. De computer kan dan de benodigde geheugenruimte reserveren. Hier is dat niet het geval. De computer ziet dat een aantal variabelen zal worden gebruikt (A(X)) maar weet nog niet hoeveel. Hij reserveert vast geheugenruimte voor elf variabelen (van A(0) tot en met A(10)) zodat het eerste voorbeeld goed werkte, het hoogste subscribt van A(X) is 10.
Bij het daarop volgende voorbeeld zijn er twaalf variabelen A(X) en is er dus te weinig geheugenruimte gereserveerd. Voer de volgende regel eens in het programma en RUN:

```
5 DIM A(11)
```

Nu werkt het programma wel omdat het DIM statement aangeeft wat de variabele met het hoogste subscribt (11) is en dus de benodigde geheugenruimte kan worden gereserveerd. Een dergelijke reeks variabelen (van A(0)) tot en met A(11)) wordt een ARRAY genoemd en zo zijn er floting point arrays, integer arrays en string arrays. Bij tabellen zijn er X verticale kolommen en Y horizontale regels (zie het ''spaar'' programma). Eén element in zo'n tabel zou dan kunnen worden benoemd met bijvoorbeeld A(X,Y), voor de derde kolom en de tweede regel: A(2,1). Zijn er zeven kolommen en acht regels dan is A(6,7) het hoogste element en moet de dimensionering luiden:

dim a(6,7)

Dit is een twee dimensionaal array.

6. Variaties met variabelen

6.1. De diverse soorten variabelen

U hebt intussen al kennis gemaakt met verschillende soorten van variabelen die bij de MSX computer mogelijk zijn. Al deze variabelen moeten worden benoemd. Hiervoor kunt u kiezen voor één letter (A, B, C, . . . Z), twee letters (AA, AB, AC, . . . ZZ) of een letter en een cijfer van 0 tot en met 9, A0, A1, . . . A9, . . . Z9). De letters zijn steeds hoofdletters. Gebruikt u bij het intoetsen de kleine letters dan maakt de computer daar zelf hoofdletters van. Probeer maar eens:

a = 5: B = 6: print A: PRINT b

De variabelen zijn in twee groepen in te delen, de stringvariabelen en de numerieke variabelen. Voor deze variabelen wordt door de computer geheugenruimte gereserveerd en daarom kan de inhoud van de variabele nooit een bepaalde grootte overschrijden. Zo kan een stringvariabele niet meer dan 255 karakters bevatten. Dit kunnen dan wel alle soorten van karakters zijn die met het toetsenbord kunnen worden voortgebracht, dus ook de grafische karakters (met de toets GRAPH) en de speciale karakters (met de toets CODE). Zowel de hoofd- als de kleine letters worden in de ingebrachte vorm in de string bewaard.
Een stringvariabele wordt benoemd door achter de ''naam'' (A, ZB, C1 enz.) het teken ''$'' te plaatsen: A$, ZB$, C1$ enz.
De numerieke variabelen zijn onderverdeeld in integer variabelen en floating point variabelen. De integer variabelen kunnen getallen bevatten *tussen* −32769 en +32768. Bij positieve getallen wordt het ''+'' teken weggelaten. Het grootste positieve getal dat een integer variabele kan bevatten is dus 32767. De integer variabele wordt benoemd door achter de naam het teken ''%'' te plaatsen.
De floating point variabelen zijn onderverdeeld in variabelen voor lage nauwkeurigheid en voor hoge nauwkeurigheid. Met die lage nauwkeurigheid valt het overigens nog al mee. Van de getallen die meer dan zes cijfers groot zijn worden de zevende en volgende cijfers volgens de rekenkundige methode afgerond. De variabele voor lage nauwkeurigheid worden benoemd door achter de naam het teken ''!'' te plaatsen. Probeer eens de volgende regels:

a = 1/7: a! = 1/7: print a: print a!
a = 1/6: a! = 1/6: print a: print a!

en ook:

a! = 100000000/7: print a!
a! = 100000000/6: print a!

Het grootste getal dat in een variabele voor lage nauwkeurigheid kan worden ingevoerd is 9.999994E + 62. Dat wordt dan afgerond tot 9.99999E + 62. Hier wordt de absolute waarde van een getal mee bedoeld, het kan dus een positief getal zijn maar ook een negatief (het kleinste negatieve getal). Het kleinste positieve getal dat kan worden ingevoerd is 9.999995E − 65. Dit wordt afgerond tot 1E − 64. Elk kleiner positieve getal dat wordt ingevoerd wordt afgerond tot 0. Het grootste negatieve getal dat kan worden ingevoerd is −9.999995E − 64.
Bij de variabelen voor hoge nauwkeurigheid worden van de getallen die meer dan veertien cijfers bevatten het vijftiende en de volgende cijfers op de bekende wijze afgerond. Ze kunnen benoemd worden door achter de naam het teken ''#'' te plaatsen. Dat hoeft niet, elke numerieke variabele zonder toevoeging achter de naam wordt als een variabele voor hoge nauwkeurigheid beschouwd. Probeer de volgende regel (let op de afronding):

a = 1/6: a# = 1/6; print a: print a#

In absolute waarde is het grootste getal dat een variabele voor hoge nauwkeurigheid kan bevatten gelijk aan 9.99999999999994E + 62. Dit wordt afgerond tot 9.9999999999999E + 62.
Het kleinste getal dat ingevoerd kan worden is 9.99999999999995E − 65. Dit wordt afgerond tot 1E − 64.

6.2. Het definiëren van variabelen

Indien u meent slechts weinig geheugenruimte ter beschikking te hebben voor de variabelen, of indien veel variabelen moeten worden toegepast dan is het uiteraard nodig de numerieke variabelen op de juiste wijze te benoemen. Van variabelen met hoge nauwkeurigheid moet dan zo weinig mogelijk gebruik worden gemaakt. Integer variabelen vragen de minste geheugenruimte, daarna komen de variabelen voor lage nauwkeurigheid en dan die voor hoge nauwkeurigheid. Hoeveel geheugenruimte u op een gegeven moment nog ter beschikking heeft voor een BASIC programma met al zijn variabelen kunt u vaststellen met

```
print fre(0)
```

Op het scherm wordt dan geprint hoeveel bytes nog vrij zijn. Het kan dan zijn dat u per stringvariabele 255 karakters kunt invoeren, in totaal staan u slechts 200 bytes van de geheugenruimte ter beschikking voor deze variabele. Hoeveel hiervan nog vrij zijn kunt u vaststellen met

```
print fre(" ")
```

Probeer dit eens en voer dan de volgende regel in:

```
a$ = "MSX computer": print fre(" ")
```

U zult zien dat u voor A$ twaalf geheugenplaatsen nodig hebt gehad (met de spatie mee twaalf karakters).
In principe betekent het plaatsen van het teken "!" achter de variabelenaam A het *definiëren* van de variabele A als floating point variabele voor lage nauwkeurigheid. Dit definiëren kunt u ook voor een aantal variabelen tegelijk doen zodat de tekens achter de variabelenaam achterwege kunnen blijven. Ten eerste het volgende statement:

```
defint a-c
```

Voer dit in de direct mode uit en probeer dan:

```
a = 5.15: ad = 3.25: a1 = 10/3: b = 385.73 c = 23.9:
d = 102.75
print a; ad; a1; b; c; d
```

Uit het resultaat volgt dat alle variabelen waarvan de naam begint met een A, B of een C na DEFINT integer variabelen zijn. Probeer ook eens:

```
ad! = 1/7: print ad!
```

U ziet dat variabelen die door een teken achter de naam anders worden gedefinieerd hiervan uitgesloten zijn.

```
defsng d-e: defdbl f-k: defstr z
```

Voor deze statements geldt hetzelfde als bij de voorgaande, nu echter voor respectievelijk floating point lage nauwkeurigheid, floating point hoge nauwkeurigheid en stringvariabelen.

6.3. Bewerkingen met variabelen

Van de bewerkingen die met stringvariabelen zijn uit te voeren heeft u al in de vorige hoofdstukken een enkel voorbeeld gezien. Het aantal bewerkingen hiermee is echter zo groot dat we voorbeelden hiermee over de volgende hoofdstukken zullen verdelen. In dit hoofdstuk zal het verder in hoofdzaak over numerieke variabelen gaan. Aangenomen wordt dat geen variabelen gedefinieerd zijn. Is dat wel het geval (in verband met het voorafgaande) voer dan in:

```
clear
```

Dit statement laat alle variabelen uit het geheugen verdwijnen. De reserveringen van de geheugenplaatsen en ook de definities gaan verloren.
Alle berekeningen die de computer moet uitvoeren voor de diverse functies worden met de hoge nauwkeurigheid uitgevoerd. Probeer als voorbeeld daarvan maar eens:

```
a = exp(1): print a
```

Dat geldt ook voor de rekenkundige bewerkingen. Probeer ook:

```
a1! = 1/7: a2! = 1/3: print a1!; a2!; a1!*a2!
```

U ziet een vermenigvuldiging van twee getallen met zes cijfers achter de decimaalpunt levert een getal op van twaalf cijfers achter de decimaalpunt dat dus *niet* is afgerond tot een getal van zes cijfers achter de decimaalpunt.

Voor het bepalen van het getal vóór de decimaalpunt van een gebroken getal zijn verschillende mogelijkheden:

```
a1 = -14.2: a1% = a1: a2 = cint(a1): a3 = fix(a1)
print a1; a1%; a2; a3
```

Hierin komt u twee nieuwe functies tegen. Eerst CINT. In tegenstelling met de INT functie waardoor *alle* getallen naar beneden worden afgerond tot een geheel getal, ook de negatieve, geeft de CINT functie steeds het juiste getal voor de decimaalpunt. Dit werkt alleen voor getallen *tussen* -32769 en 32768. De FIX functie doet hetzelfde, echter ook voor getallen die buiten de genoemde grenzen vallen. Probeer eens:

```
print int(536891,123); fix(536891,123);
int(-536891,123)
```

De volgende functie waarmee we al kennis hebben gemaakt is de SGN functie. Met deze functie kunt u het teken (positief of negatief) van het resultaat van een bewerking bepalen. Het volgende demonstreert de werking:

```
a = 15: b = 10: c% = 10,15
print sgn(a - b); sgn(b - a); sgn(b - c%)
```

Voor de verandering proberen we weer eens een programmaatje. Dit berekent de grootste gemene deler van twee getallen, door ze bij voortduring op elkaar te delen. Als voorbeeld dient het volgende sommetje:

```
81/51 = 1 rest 34
51/34 = 1 rest 17
34/17 = 2 rest 0,   GGD = 17
```

Hier zijn de twee getallen 81 en 51. Deze worden op elkaar gedeeld. De rest van deze deling wordt op het kleinste getal gedeeld enz. Van de laatste deling is de rest 0. De deler hiervan is de gezochte GGD.

```
10 CLS
20 INPUT"Getal 1";G1
30 INPUT"Getal 2";G2
40 PRINT
50 IF G1<G2 THEN SWAP G1,G2
60 R=G1 MOD G2
70 IF R=0 THEN 90
80 G1=G2: G2=R: GOTO 60
90 PRINT "De GGD is";G2
```

De eerste regel van dit programma maakt het scherm schoon. De volgende twee regels zijn voor het invoeren van de twee getallen. Deze kunnen in een willekeurige volgorde worden ingevoerd. Omdat echter steeds het grootste getal door het kleinste moet worden gedeeld moeten we nagaan of G1 het grootste getal is (regel 50). Is dat niet zo dan moeten we de getallen onderling verwisselen. Dit kan met het SWAP statement. Dit verwisselt de inhouden van de variabelen G1 en G2. In regel 60 wordt op een bijzondere manier de deling G1/G2 uitgevoerd. Niet het resultaat van de deling is belangrijk, maar de rest en die vinden we in de variabele R. Hiervoor is de bewerking MOD verantwoordelijk. Als deze rest 0 is dan is de GGD gevonden (G2), anders moet opnieuw een deling worden uitgevoerd met de juiste getallen. Deze worden in regel 80 in G1 en G2 geladen.

6.4. Het printen van variabelen

Als laatste voor wat betreft de numerieke variabelen gaan we na hoe we de getallen op het scherm geprint kunnen krijgen. We hebben al gezien hoe de gehele inhoud van een variabele op een bepaalde plaats op het scherm (met TAB(X)) kunnen laten printen. We kunnen deze inhoud echter ook op het scherm laten plaatsen met een vooraf bepaald aantal decimalen achter de decimaalpunt waarbij de eventuele lagere decimalen worden afgerond. Probeer het volgende programmaatje eens:

```
10 A=256.158: B=8145.6
20 PRINT TAB(10) USING "####.##";A
30 PRINT TAB(10) USING "####.##";B
```

U ziet dat de twee getallen keurig onder elkaar op het scherm staan met de decimaalpunten op één lijn. Uiteraard is dit het resultaat van het statement USING. Vóór de decimaalpunt zijn vier plaatsen gereserveerd. Komt het getal één of meer cijfers te kort dan blijven de voorste plaatsen leeg. Achter de decimaalpunt zijn in dit geval twee plaatsen gereserveerd. Daarom worden de getallen getoond met twee decimalen na de decimaalpunt, eventueel afgerond of anders met nullen aangevuld. Het aantal plaatsen bepaalt u met het aantal tekens "#". Het kan zijn dat een getal te groot uitvalt. Verander regel 10 in:

```
10 A=256.158: B=18145.65#
```

U ziet, hoewel het getal niet in de gereserveerde ruimte past wordt het toch in zijn geheel geprint, echter met de aanduiding "%" er voor en niet meer goed op zijn plaats.
Andere toepassingen van het PRINT USING statement ziet u als u de volgende regels in de direct mode probeert:

```
print using "# . # #^^^^";354.1245
print using "# #. # #^^^^";9876.5432
```

Het PRINT USING statement kan ook worden gebruikt bij het printen van stringvariabelen. Zie hiervoor het volgende programma.

```
10 SCREEN 0: WIDTH (40)
20 CLEAR 2000
30 DIM A$(50),N$(50),P$(50),T$(50),W$(50)
40 PRINT "Wilt u een nieuw adres invoeren,"
50 INPUT" ja of nee";J$
60 IF J$="nee" THEN 160
70 IF J$<>"ja" THEN 40
80 INPUT "Naam";N$(X)
90 INPUT "Adres";A$(X)
100 A$(X)= N$(X)+"    "+A$(X)
110 INPUT "Postcode";P$(X)
120 INPUT "Woonplaats";W$(X)
130 P$(X)= P$(X)+"    "+W$(X)
140 INPUT "telefoonnr.";T$(X)
150 X=X+1: PRINT: GOTO 40
160 INPUT "wat is de naam";S$
170 FOR N=0 TO X
180 I=INSTR(1,A$(N),S$)
190 IF I=0 AND N<>X THEN NEXT N
200 IF I=0 THEN 250
210 CLS: PRINT A$(N): PRINT
220 PRINT P$(N): PRINT
230 PRINT USING"Tel.    &";T$(N)
240 PRINT: PRINT: PRINT: GOTO 40
250 PRINT"Naam niet in bestand."
260 GOTO 40
```

Met dit programma kunt u een zogenaamd "bestand" van adressen met telefoonnummers maken. Hier volgt de verklaring. In regel 10 wordt er voor gezorgd dat er zoveel mogelijk ruimte op het scherm ter beschikking is voor de tekst.
Als u bepaalde instructies niet meer kent (alle "nieuwe" instructies worden toegelicht) is het uiteraard het beste als u de werking daarvan even naleest. In dit programma wordt bijna uitsluitend met stringvariabelen gewerkt. Nu mag het zo zijn dat er maximaal 255 tekens in een string gaan, dat wil nog niet zeggen dat hiervoor geheugenruimte aanwezig is. Bij het inschakelen is geheugenruimte voor maximaal 200 tekens bestemd. Dit kunt u vaststellen als u het volgende intoetst:

```
print fre(" ")
```

Dit is niet genoeg voor ons doel. We kunnen echter de hoeveelheid geheugenplaatsen zelf bestemmen. In regel 20 is met CLEAR 2000 gezorgd voor een ruimte van 2000 geheugenplaatsen. Dit kan ook in de direct mode:

```
clear 2000: print fre(" ")
```

Gerekend is op 50 namen (kaarten) met gegevens en 40 tekens per kaart. Dat betekent ook dat de stringvariabelen met "subscribt" (getal tussen haakjes) moeten worden gedimensioneerd (regel 30). Als u op de vraag in regel 40 te kennen geeft dat u geen nieuw adres wilt invoeren dan kan dat alleen maar betekenen dat u de inhoud van een kaart op het scherm wilt zien.
Regel 70 zorgt er voor dat alleen maar wordt gereageerd op "ja" of "nee". Heeft u met "ja" geantwoord dan komt u bij regel 80. Eerst wordt de naam en het adres ingevoerd. Deze worden in regel 100 tot één string A$(X) samengevoegd. Tussen de naam en het adres worden drie spaties ingevoegd. Dezelfde bewerking ondergaan de postcode en de woonplaats. Als laatste wordt het telefoonnummer ingevoerd. In regel 150 wordt de teller X verhoogd (X is na RUN steeds 0). De teller X geeft daarmee het kaartnummer aan. Hierna wordt naar regel 40 teruggegaan zodat u opnieuw voor de keus "ja" of "nee" wordt gesteld. Heeft u met "nee" gekozen voor het printen van de informatie dan komt u bij regel 160. Hier wordt om de naam gevraagd waarvan u de informatie wilt weten. Met deze naam moet de betreffende kaart geselecteerd worden uit de aanwezige kaarten waarvan het hoogste kaartnummer X is, vandaar: FOR N = 0 TO X. Nu volgt een nieuwe functie. Met INSTR(. .) wordt namelijk gezocht in de string A$(N) (naam plus adres) of de naam die in S$ is opgeslagen (zo juist ingevoerd) aanwezig is. Dat gebeurt vanaf het eerste karakter. Hiervoor is namelijk het getal 1 ingevoerd. Dit getal geeft het karakter aan vanwaar af in de string gezocht wordt. Dit getal kan niet kleiner dan 1 en niet groter dan 255 zijn. Wordt de naam niet gevonden dan is I gelijk 0. Wordt de naam wel gevonden dan bevat I het nummer van het karakter, gerekend vanaf het meest linkse. In regel 190 wordt nagegaan of de kaart met de naam gevonden is. Is dat niet zo (I = 0) en alle kaarten zijn nog niet aan de beurt geweest (N < > X) dan wordt de volgende

kaart geprobeerd (NEXT N). Zijn alle kaarten beproefd dan is de naam niet in het bestand aanwezig. Dat wordt u door regel 250 meegedeeld. Is de naam wel gevonden dan wordt de inhoud van de kaart op het scherm geprint. Overigens, als het goed is kunt u de regels 190 en 200 verbeteren! Eerst worden de naam en het adres op het scherm geprint. Daarna de postcode en de woonplaats. Nu nog het telefoonnummer. We moeten zuinig zijn met de geheugenruimte en het is dus niet juist om op elke kaart hetzelfde woord te schrijven: ''Telefoonnr.''. De oplossing geeft PRINT USING. In regel 230 wordt eerst ''Tel. '' op het scherm geprint (let op de spaties) en daarna op de plaats van het teken ''&'' de inhoud van T$(N). Waar het teken ''&'' ook staat in de string achter USING (tussen de '' '' tekens), steeds zal de inhoud van de string die is genoteerd achter deze string (hier dus T$(N)) de plaats van & innemen. De rest van dit programma is eenvoudig te volgen. Probeer zelf eens na te gaan of voor het vinden van de kaart eigenlijk wel de volledige naam ingevoerd moet worden!
Op welke manier het bestand (de inhoud van de kaarten) kan worden bewaard lezen we in het volgende hoofdstuk.

Er zijn nog twee mogelijkheden met PRINT USING:

```
10 A$="BASIC"
20 B$="Programma"
30 PRINT USING "!";A$,B$
40 PRINT USING "\   \";B$
```

In regel 30 wordt door het teken ''!'' zowel van de inhoud van A$ als van B$ slechts het eerste karakter geprint. Door regel 40 wordt de lengte van de string die geprint wordt (de inhoud van B$ verandert niet) beperkt. Tussen de schuine strepen achter USING zijn twee spaties. Daarom worden de eerste *vier* karakters van B$ geprint (Prog), meer niet. Het aantal karakters wat geprint wordt (te beginnen met het meest linkse karakter) is steeds gelijk aan het aantal spaties *plus* twee.

Om het niet meer af te leren volgt hier nog een kort programma:

```
10 DEF FNI(X)=3.1416*X^2/4:CLS
20 INPUT"Wat is de lengte";L
30 INPUT"Wat is de buitendiameter";BD
40 INPUT"Wat is de binnendiameter";ID
50 INPUT"Wat is de soortelijke massa";SM
60 MASSA=SM*L*(FNI(BD)-FNI(ID))
70 PRINT:PRINT:PRINT"De massa is ";
80 PRINT USING"##.####^^^^";MASSA;
90 PRINT" kg"
```

Het is een programma voor het berekenen van een massa van een pijp waarvan u de lengte, de buiten- en de binnendiameter en de soortelijke massa kunt invoeren. Nu zal het u mogelijk een zorg zijn hoe groot de massa van een pijp is maar het gaat er maar om de FN . .(X) functie te introduceren. Op de plaats van de puntjes kunnen één of twee letters of een letter en een cijfer worden ingevuld voor de naam van de functie. Hier is gekozen voor I: FNI. Deze functie wordt gebruikt als in een programma meerdere keren dezelfde berekening moet worden gemaakt met steeds andere gegevens, zoals hier de berekening van de totale oppervlakte van de doorsnede en die van de binnendoorsnede.
Eerst wordt de functie gedefinieerd: DEF FNI op regel 10 (DEF is een statement dat hier een ander doel heeft dan het definiëren van variabelen). In regel 60 wordt de totale oppervlakte van de doorsnede berekend door voor X in FNI(X) de buitendiameter BD in te vullen.
Daarna wordt in dezelfde regel de oppervlakte van de binnendoorsnede berekend. In programmaregel 80 vindt u nog een toepassing van het PRINT USING statement.

7. Werken met het cassettedeck

7.1. Inleiding

Het wordt nu tijd om te beschrijven hoe programma's vanuit de computer naar de cassetteband kunnen worden geschreven en andersom, hoe programma's vanaf de cassette in de computer kunnen worden gelezen.
Het is uiteraard makkelijk om programma's op de cassetteband te bewaren, zodat ze later weer kunnen worden gebruikt. Het vervelende intikken (met de kans op fouten) is dan niet meer nodig. Het kan ook voorkomen dat er in een uit te proberen programma fouten zijn die alles in de war sturen en het uit- en weer inschakelen van de computer of het bedienen van de reset toets (indien aanwezig) de enige mogelijkheid is om weer normaal te kunnen werken. Het nare is dan dat het programma weer opnieuw moet worden ingetoetst. Het beste is daarom elk programma, zodra het in de computer is opgeslagen, ook al bevat het waarschijnlijk een aantal fouten, direct op de cassetteband op te nemen, zodat het steeds opnieuw in de computer kan worden gevoerd.
Het beste is een voor de computer geschikte datarecorder aan te schaffen. Het aansluiten daarvan kan geen moeilijkheden opleveren. In principe kunt u echter elke willekeurige cassetterecorder gebruiken.
Het handigste is dan een recorder met een remote control aansluiting. In dat geval wordt het in- en uitschakelen van de motor door de computer verzorgd. Voor het aansluiten van een recorder van een ''europees'' merk zult u echter uw handelaar moeten vragen om een stekeraanpassing.
U kunt bij uw cassetterecorder speciale cassettes gebruiken die u bij uw leverancier kunt kopen, maar ook gewone cassettes. Neem daarvoor ijzeroxyde cassettebandjes van een goed merk. Het beste is om C60 cassettes te nemen die u kunt openschroeven. Kort de bandjes voor de helft in en verwijder de aanloopstroken, deze zijn maar lastig. De verkorte bandjes spoelen veel sneller om. Bovendien zet u er nooit na elkaar zoveel programma's op dat ze vol raken. De officiële data cassettes zijn daarom maar twee keer vijftien minuten. Heeft u cassettes met aanloopstrook, wikkel ze dan voor het opnemen van een programma zover op dat u geen last meer heeft van de aanloopstrook.

7.2. Het SAVEN van programma's

We zullen eerst het programma voor het telefoonbestand naar de tape schrijven (hoofdstuk 6.4). We toetsen eerst dit programma in de computer. We geven dit programma de naam ''telbst''. Dit is nodig als de computer uit meerdere programma's op de tape moet selecteren. De naam wordt namelijk met het programma op de tape geplaatst. De lengte van de naam mag niet groter zijn dan zes karakters. Om het programma naar de band te schrijven heeft de computer het commando

csave ''telbst'' (RETURN)

Handel voor een recorder met remote control als volgt: plaats de recorder in de stand ''opnemen'' en toets bovenstaande zin in en druk op de RETURN toets. De recorder gaat dan draaien en het programma wordt opgenomen. Is het programma in zijn geheel naar de band geschreven dan stopt de computer de recorder en verschijnt ''Ok'' op het scherm. De handelwijze voor een recorder zonder remote control is anders. Toets csave''telbst'' in (geen RETURN). Zet de recorder in de stand opnemen (die begint meteen te draaien) en druk *daarna* op de RETURN toets. Nu wordt het programma naar de band geschreven. Is de computer hiermee klaar dan verschijnt ''Ok'' op het scherm en moet u de cassette stoppen.
Nu kan er bij het ''saven'' van alles mis zijn gegaan. Ten eerste kunnen er op de band fouten in de magnetische laag zijn. Ten tweede kan de opname sterkte (indien die niet automatisch wordt ingesteld) te laag zijn. Daarom kent de computer een methode om vast te stellen of het programma goed op de band staat. Hiervoor dient het commando:

cload?''telbst'' (RETURN)

Let vooral op het vraagteken achter ''cload''. Voor de bezitters van een recorder met remote control: spoel de tape terug, plaats de recorder in de stand ''weergeven'' en toets bovenstaande regel in, gevolgd door RETURN. Is het bij uw cassetterecorder nodig dat bij het terugspoelen de remote control de motor inschakelt toets dan in:

motor on (RETURN)

Vergeet na het terugspoelen niet om deze regel in te toetsen:

motor off (RETURN)

Zonder remote control: toets cload?''telbst'' in (uiteraard eerst terugspoelen) en druk direct daarna op de RETURN toets. Plaats daarna de recorder in de stand ''weergeven''. In beide gevallen gaat de computer het programma op de band opzoeken. Als het is gevonden meldt de computer Found:telbst. Daarna gaat hij het programma op de band vergelijken met het programma in zijn geheugen. Zijn beiden gelijk dan meldt hij ''Ok''. Merkt hij een verschil dan meldt hij

Verify error

Nu zijn er weer verschillende mogelijkheden. Het kan zijn dat het programma goed op de band staat maar dat de weergavesterkte niet goed is. Herhaal de procedure dan met een grotere of een kleinere weergavesterkte. Lukt dat niet dan staat het programma niet goed op de band. Save het dan nogmaals op een ander gedeelte van de band en als u de opnamesterkte zelf moet instellen, neem dan een andere waarde. Iets sterker dan normaal opnemen (in het rode vlak van de opnamesterktemeter) kan meestal geen kwaad. Na enig proberen zal u het wel lukken een programma op de band te krijgen. Is het gelukt, spoel dan na de laatste keer verifiëren niet terug maar laat de band staan. Noteer vooral de stand van de sterkteregelaars die u hebt moeten instellen! Als u helemaal op ''safe'' wilt gaan, save dan het programma twee keer na elkaar op een bandje of op twee verschillende bandjes.
We gaan achter dit programma nog twee programma's op de band zetten, maar nu met het commando

save''cas:<naam>'' (RETURN)

als u niet het commando csave gebruikt dan moet u behalve de naam ook aan de computer meedelen naar welk randapparaat het programma moet worden verzonden. We gaan nu het programma voor lenen uit hoofdstuk 5.2 naar de band schrijven. Hiervoor toetsen we in:

save''cas:lenen'' (RETURN)

De werkwijze is verder geheel gelijk aan die met het commando CSAVE.
Het is echter niet mogelijk te verifiëren met CLOAD? of LOAD?
Misschien vindt u zo wel genoeg, maar voor het verdere verloop van het verhaal is het nodig dat ook het programma voor sparen uit hoofdstuk 5.2 naar de band wordt geschreven. Toets het daarvoor in zoals het in dat hoofdstuk is afgedrukt en voer daarna de volgende regel in (in de direct mode)

renum 2000,10,10

List nu het programma eens. U zult zien dat de regelnummers nu zijn veranderd en beginnen met 2000.
Het eerste getal achter renum (2000) geeft aan met welk nummer het ''hernummeren'' moet beginnen, het tweede nummer (10) met welke regel het hernummeren moet aanvangen en het laatste nummer hoe groot de stappen moeten zijn. Plaats nu dit programma achter het programma ''lenen'' en noem het ''sparen''. Gebruik ook hiervoor het commando save:

save''cas:sparen'' (RETURN)

7.3. Het samenvoegen van programma's van de tape

Als u een programma goed op de band hebt gekregen dan is het niet moeilijk om het weer in de computer te laden. Een programma dat met CSAVE naar de tape is geschreven kan weer in de computer worden geladen met CLOAD. Dit komt in de volgende paragraaf aan bod. Nu gaan we het programma ''lenen'' in de computer laden. We doen dit door eerst de cassette terug te spoelen tot de teller van de cassetterecorder het begin van het programma aangeeft. U kunt ook verder terug spoelen. Bij het zoeken van het programma slaat de

computer de programma's die hij niet moet laden gewoon over (skip). We gebruiken voor het laden:

load''cas:lenen'' (RETURN)

Een programma dat met SAVE naar de tape is geschreven kan *alleen* met LOAD weer worden opgehaald. Een programma dat met CSAVE naar de band is geschreven kan *niet* met LOAD worden geladen. De handelwijze bij het laden is volkomen gelijk aan die bij CLOAD? uit de vorige paragraaf. In dit geval wordt een programma dat reeds in de computer aanwezig is, vernietigd (NEW) en het te laden programma in de plaats daarvan in het geheugen geschreven. Met een LIST kunt u vaststellen dat het programma lenen inderdaad in de computer is geladen. Verander regel 150 in:

```
150 GOTO 2000
```

Als u het programma nu RUNT gaat het fout! Laadt nu het programma ''sparen'', echter met het commando:

merge''cas:sparen'' (RETURN)

Ook nu is de handelwijze volkomen gelijk aan die bij CLOAD?.
Het commando MERGE heeft echter tot gevolg dat het programma ''sparen'' in de computer wordt geladen, zonder dat het programma ''lenen'', dat nog in het geheugen zat, wordt vernietigd. Dat is dan ook de reden waarom de regelnummers van het programma ''sparen'' moesten worden veranderd. Als u nu RUN invoert (toets f5) dan zult u eerst de gegevens voor het programma ''lenen'' kunnen invoeren. Is het resultaat daarvan getoond dan wordt u gevraagd de gegevens voor het programma ''sparen'' in te voeren en ook het resultaat hiervan zal u worden getoond. Dat betekent dus dat het programma ''sparen'' nu is samengevoegd met het programma ''lenen''. Gaat u eens na waarom regel 150 daarvoor moest worden veranderd. Het commando MERGE kan alleen worden toegepast op programma's die met SAVE naar de tape zijn geschreven.

7.4. De behandeling van een ''FILE''

De bedoeling van deze paragraaf is om een manier te beschrijven waarmee de gegevens van ons telefoonbestand op de cassetteband kunnen worden bewaard. Het programma ''telbst'' staat al op de band en moet nog worden veranderd en aangevuld. Daarom laden we het eerst in de computer. Een programma dat met CSAVE naar de cassette is geschreven kan alleen met CLOAD weer worden opgehaald:

cload''telbst'' (RETURN)

De handelwijze is gelijk aan die van CLOAD?. Om de tekst ''cload'' op het scherm te krijgen hoeft u slechts toets f7 in te drukken. Deze tekst hoeft u dan alleen nog maar aan te vullen met telbst. Ook zonder dat u afsluit met de aanhalingstekens werkt het statement! Voor uw gemak wordt het programma hieronder nog een keer afgedrukt.

```
10 SCREEN 0: WIDTH (40)
20 CLEAR 2000
30 DIM A$(50),N$(50),P$(50),T$(50),W$(50)
40 PRINT "Wilt u een nieuw adres invoeren,"
50 INPUT" ja of nee";J$
60 IF J$="nee" THEN 160
70 IF J$<>"ja" THEN 40
80 INPUT "Naam";N$(X)
90 INPUT "Adres";A$(X)
100 A$(X)= N$(X)+"    "+A$(X)
110 INPUT "Postcode";P$(X)
120 INPUT "Woonplaats";W$(X)
130 P$(X)= P$(X)+"    "+W$(X)
140 INPUT "telefoonnr.";T$(X)
150 X=X+1: PRINT: GOTO 40
160 INPUT "wat is de naam";S$
170 FOR N=0 TO X
180 I=INSTR(1,A$(N),S$)
190 IF I=0 AND N<>X THEN NEXT N
200 IF I=0 THEN 250
210 CLS: PRINT A$(N): PRINT
220 PRINT P$(N): PRINT
230 PRINT USING"Tel.    &";T$(N)
240 PRINT: PRINT: PRINT: GOTO 40
250 PRINT"Naam niet in bestand."
260 GOTO 40
```

Eerst moet dit programma worden gewijzigd. Daarvoor verwijderen we de regels 40 tot en met 70. Dat gaat het beste met het volgende commando:

delete 40-70 (RETURN)

Voer dit in (direct mode) en zie met LIST wat het resultaat hiervan is. Nu dienen de volgende regels ingevoerd te worden:

```
40 PRINT "gebruik voor laden: l, saven: s"
45 PRINT "een nieuwe naam: n, en opvragen: o."
```

```
50 INPUT K$
55 IF K$="l" THEN 3000
60 IF K$="s" THEN 2000
65 IF K$="o" THEN 160
70 IF K$<>"n" THEN 40
```

Het is de bedoeling dat u niet alleen een ''nieuwe kaart'' (nieuwe naam) kunt maken, maar ook dat u de gegevens kunt opvragen, het bestand (niet het programma zelf maar de ''kaarten'') naar de tape kunt schrijven (saven) of het van de tape weer in de computer kunt laden (load). De nu ingevoerde regels selecteren deze werkwijzen en verwijzen naar de betreffende programmadelen. Voor het saven van het bestand worden de volgende regels aan het programma toegevoegd:

```
2000 OPEN"cas:kaart" FOR OUTPUT AS #1
2010 FOR N=0 TO X-1
2020 PRINT #1, A$(N);",";P$(N);",";T$(N)
2030 NEXT N
2040 CLOSE 1
2050 GOTO 40
```

Alle gegevens die naar een randapparaat moeten worden verstuurd worden in een zogenaamde ''FILE'' geplaatst, ook een programma bijvoorbeeld. In dat geval verricht de computer een aantal handelingen als gevolg van slechts een enkel statement (CSAVE, SAVE). Voor het saven van een bestand moeten echter voor de afzonderlijke handelingen elk apart een statement worden ingevoerd. Het eerste is het openen van de file, hier met OPEN''cas:kaart''. Zowel de naam van het randapparaat als de naam van het bestand (hier ''kaart'') moeten worden vermeld. Ten tweede moet voor het zenden naar het randapparaat van de gegevens de file een uitgang worden: FOR OUTPUT. Ten derde moet de file een nummer hebben. Er kunnen namelijk meerdere files tegelijk geopend zijn en dan moet de computer toch weten middels welke file de gegevens moeten worden verzonden.

De computer is er op ingesteld dat slechts één file geopend is. Daarom is hier gekozen voor nummer 1: AS #1. Voor een hoger nummer dient u eerst het volgende commando in te voeren:

maxfiles = a

Hierin kan a een getal van 0 tot en met 15 zijn. Het filenummer 0 is gereserveerd voor SAVE en LOAD opdrachten en na het inschakelen van de computer is a gelijk 1.

In het programma ''telbst'' is X steeds 1 groter dan het grootste kaartnummer (ga dat na), vandaar dat in regel 2010 de grootste waarde van de teller op X – 1 is gesteld. Nu worden in regel 2020 de gegevens kaart voor kaart naar de cassetterecorder verzonden.
Hiertoe dient PRINT #. Dit geeft aan dat de gegevens in een geopende file moeten worden verzonden. Om aan te geven welke file is het nummer nodig: PRINT #1. De stringvariabelen A$(N), P$(N) en T$(N) mogen niet ''aan elkaar geplakt'' op de tape komen. Daarom moeten ze worden gescheiden door een komma. Nadat alle kaarten zijn verzonden moet de file weer worden gesloten:
CLOSE 1.

Voor het laden van het bestand uit de computer moeten de volgende regels worden ingevoerd:

```
3000 OPEN"cas:kaart" FOR INPUT AS #1
3010 X=0
3020 INPUT #1, A$(X), P$(X), T$(X)
3030 X=X+1
3040 IF EOF(1)=0 GOTO 3020
3050 CLOSE: X=X-1
3060 GOTO 40
```

Opnieuw moet een file worden geopend. Uiteraard moet dezelfde naam worden gebruikt, anders kan de computer het bestand op de tape niet vinden (''kaart''). Nu moet de file echter een ingang zijn (FOR INPUT). Ook nu worden de gegevens kaart voor kaart gelezen. We moeten beginnen bij kaartnummer 0 vandaar X = 0. Het INPUT # statement geeft aan dat de gegevens moeten worden binnen gehaald via een geopende file. Het nummer van de file is weer 1 (INPUT #1). Steeds als de gegevens voor een kaart zijn binnengehaald moet worden bepaald of nog meer gegevens van de tape te verwachten zijn. Zijn er geen gegevens meer van de tape te verwachten dan wordt een ''END OF FILE'' teken gegeven dat in regel 3040 de waarde – 1 krijgt. Zolang dat nog niet het geval is en nog meer gegevens te verwachten zijn is EOF nog 0 en kunnen we terug naar regel 3020 voor de volgende kaart. Ook voor het EOF teken moet het filenummer worden ingevoerd (EOF(1)). Zijn de gegevens voor de laatste kaart ingevoerd dan is het kaartnummer X weer net 1 te hoog. Dat wordt met

X = X – 1 gecorrigeerd. Daarna gaan we weer terug naar regel 40 waar de volgende werkwijze kan worden geselecteerd. Eerst moet de file echter weer worden gesloten. Dat kan ook met CLOSE zonder meer. Als geen filenummer hierbij is vermeld dan worden alle files gesloten die open zijn. Dat is in ons geval maar één file.

Vindt u de nummering van de programmaregels nu wat slordig dan kunt u intoetsen:

renum (RETURN)

Door RENUM zonder toevoeging van een getal worden de regels genummerd te beginnen met 10 en oplopend met 10.

Als u enige kennis heeft van de elektrotechniek dan kunt u gebruik maken van de commando's MOTOR ON en MOTOR OFF. In de computer verbindt een relais de contacten van de remote control steker al of niet met elkaar door. U kunt daarmee de computer iets laten inschakelen of uitschakelen. Het relais in de computer is echter maar klein. U kunt dus maar kleine spanningen en slechts kleine stromen schakelen in ohmse belastingen. Wat het relais kan verdragen is mij niet bekend.

8. De ASCII-code

8.1. Het buffergeheugen

Zoals is gebleken, is het mogelijk gegevens in te voeren via het toetsenbord met het input statement. Het is ook mogelijk de computer bepaalde handelingen te laten verrichten door middel van een programma met daarin een statement dat leest welke toets is ingedrukt. Dit is het INKEY$ statement. Het INKEY$ statement heeft steeds betrekking op een stringvariabele. De geheugenplaats die voor deze stringvariabele wordt gereserveerd wordt dan gevuld met het codegetal van het karakter dat bij de ingedrukte toets hoort. Is dat het karakter A dan is het codegetal 65. Bij het karakter a hoort het codegetal 97. Probeer het volgende programma:

```
10 CLS
20 A$=INKEY$: IF A$="" GOTO 20
30 PRINT A$
40 GOTO 20
```

Hier is in regel 20 het INKEY$ statement toegepast. Dit statement leest de toets die ingedrukt is en plaatst het codegetal dat er bij hoort in de variabele A$. Zolang er geen toets is gebruikt, zal de inhoud van A$ "leeg" zijn en dit leeg wordt aangeduidt met A$ = " ", zonder spatie tussen de aanhalingstekens. Een spatie zou namelijk aangeven dat A$ gevuld is met het codegetal voor "spatie". Dit is geheel in overeenstemming met A$ = "B" of A$ = "8". De computer moet weten wat hij moet doen als geen toets is ingedrukt, vandaar: IF A$ = " " THEN 20. Zo blijft hij netjes wachten tot een toets wordt ingedrukt. Wordt bijvoorbeeld de toets voor m ingedrukt dan wordt door PRINT A$ een m op het beeldscherm geschreven waarna weer wordt gewacht op het indrukken van de volgende toets (40 GOTO 20). We stoppen het programma en voeren een vertraging in:

```
10 CLS: Y=10000
20 A$=INKEY$: IF A$="" THEN 20
30 FOR X=0 TO Y: NEXT X
40 PRINT A$;: Y=1: GOTO 20
```

Run dit programma en toets een karakter in. U moet nu een behoorlijke tijd wachten voordat u het ingetoetste karakter op het scherm te zien krijgt (Y = 10000). Het volgende karakter dat u intoetst krijgt u direct op het scherm te zien (Y = 1). Stop het programma en RUN het opnieuw. Als u nu een karakter intoetst meent u waarschijnlijk dat u moet wachten met het intoetsen van het volgende karakter totdat het eerste op het scherm is afgedrukt. Dat is niet waar. U kunt veertig karakters intoetsen zonder dat deze verloren gaan. Toets maar eens zeer snel meer dan veertig karakters in (bijvoorbeeld vier maal de toetsen 1 tot en met 0 plus nog een paar). Doe dat voordat het eerste karakter wordt getoond. U zult zien dat u de eerste veertig van de ingetoetste karakters te zien krijgt. De andere karakters zijn verloren gegaan. Dit komt omdat het toetsenbord van een "buffergeheugen" gebruik maakt waarin veertig karakters kunnen worden opgeslagen. Door het statement INKEY$ wordt dit "toetsenbordbuffergeheugen" uitgelezen, steeds één karakter per keer en het eerst ingetoetste karakter ook het eerst. Is een karakter gelezen dan is er vanzelf plaats gemaakt voor het intoetsen van een volgend karakter. Dat niet alle karakters door INKEY$ gelijktijdig worden gelezen kunt u waarnemen als u in regel 40 Y = 200 maakt. Dan ziet u alle tekens met vertraging op het scherm komen.

8.2. De codegetallen

Het codegetal dat bij een karakter hoort, is ontleend aan de ASCII code, afkoring voor American Standard Code for Information Interchance (Amerikaanse standaard code voor informatie uitwisseling).
Toets het volgende programma in en RUN het:

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

Van elk karakter dat u nu intoetst verschijnt op het scherm het ASCII codegetal. Vergelijk deze getallen met die van lijst 2.
Het zal u duidelijk zijn dat de functie ASC(A$) in regel 30 hier het codegetal geeft dat bij het karakter behoort in A$. Er is ook een directe manier om de ASCII getallen te vinden. Stop het programma en probeer in de direct mode:

print asc("A"), asc("B"), asc("C")

Hexa-decimale code	00–1F		20–3F		40–5F		60–7F		80–9F		A0–BF		C0–DF		E0–FF	
	code	teken	code	teken	code	teken	code	teken	code	teken	code	teken	code	teken	code	teken
0	0	(nul)	32	(spatie)	64	@	96	`	128	Ç	160	á	192	▬	224	α
1	1	☺	33	!	65	A	97	a	129	ü	161	í	193	▙	225	β
2	2	☻	34	"	66	B	98	b	130	é	162	ó	194	▄	226	Γ
3	3	♥	35	#	67	C	99	c	131	â	163	ú	195	▬	227	Π
4	4	♦	36	$	68	D	100	d	132	ä	164	ñ	196	▪	228	Σ
5	5	♣	37	%	69	E	101	e	133	à	165	Ñ	197	▄	229	σ
6	6	♠	38	&	70	F	102	f	134	å	166	ª	198	▌	230	μ
7	7	•	39	'	71	G	103	g	135	ç	167	º	199	▟	231	γ
8	8	◘	40	(	72	H	104	h	136	ê	168	¿	200	▙	232	Φ
9	9	○	41	)	73	I	105	i	137	ë	169	⌐	201	▌	233	Θ
A	10	◙	42	*	74	J	106	j	138	è	170	¬	202	▙	234	Ω
B	11	♂	43	+	75	K	107	k	139	ï	171	½	203	▨	235	δ
C	12	♀	44	,	76	L	108	l	140	î	172	¼	204	▨	236	∞
D	13	♪	45	—	77	M	109	m	141	ì	173	¡	205	▼	237	Ø
E	14	♫	46	.	78	N	110	n	142	Ä	174	«	206	▲	238	∈
F	15	☼	47	/	79	O	111	o	143	Å	175	»	207	▶	239	∩
0	16	┼	48	0	80	P	112	p	144	É	176	Ã	208	◀	240	≡
1	17	┴	49	1	81	Q	113	q	145	æ	177	ã	209	⧗	241	±
2	18	┬	50	2	82	R	114	r	146	Æ	178	Ĩ	210	⋈	242	≥
3	19	├	51	3	83	S	115	s	147	ô	179	ĩ	211	▮	243	≤
4	20	┤	52	4	84	T	116	t	148	ö	180	Õ	212	▮	244	⌠
5	21	┼	53	5	85	U	117	u	149	ò	181	õ	213	▪	245	⌡
6	22	─	54	6	86	V	118	v	150	û	182	Ũ	214	▬	246	÷
7	23	─	55	7	87	W	119	w	151	ù	183	ũ	215	▩	247	≈
8	24	┌	56	8	88	X	120	x	152	ÿ	184	Ĩ	216	△	248	°
9	25	┐	57	9	89	Y	121	y	153	Ö	185	ĩ	217	⁺	249	•
A	26	└	58	:	90	Z	122	z	154	Ü	186	³/₄	218	ω	250	˙
B	27	┘	59	;	91	[	123	{	155	¢	187	⌣	219	▉	251	√
C	28	╳	60	<	92	\	124	¦	156	£	188	◇	220	▙	252	η
D	29	╱	61	=	93	]	125	}	157	¥	189	‰	221	▌	253	²
E	30	╱	62	>	94	^	126	~	158	₧	190	¶	222	▐	254	■
F	31	┼	63	?	95	_	127		159	ƒ	191	§	223	▀	255	

Lijst 2

Let op de hoofdletters! Probeer nu het volgende:

print chr$(65), chr$(66), chr$(67)

Het blijkt dat CHR$(X) de tegenhanger is van ASC(X$).
Zowel met de directe methode als met het voorafgaande programma zal u het niet lukken de ASCII codegetallen te vinden die in lijst 2 met 0 tot en met 31 zijn aangegeven. Dit komt omdat deze nummers dubbel zijn gebruikt. Ze worden namelijk ook toegepast op bepaalde werkingen die veelal met de cursorbewegingen te maken hebben. Bij toepassing van het PRINT statement wordt aan deze werkingen de voorkeur gegeven. In lijst 3 worden de codegetallen gegeven voor de diverse functies:

Codegetal	functie
8	Back Space
9	TAB
10	Cursor 1 regel naar onder
11	Cursor HOME
12	CLR HOME
13	RETURN
18	Insert
24	Select
27	Escape
28	Cursor 1 plaats naar rechts
29	Cursor 1 plaats naar links
30	Cursor 1 plaats naar boven
31	Cursor 1 plaats naar onder
32	Spatie
127	Delete

Lijst 3

Deze werkingen kunt u ook met de betreffende toetsen van het toetsenbord oproepen. Nog niet beschreven is de TAB functie. Deze verdeelt het scherm in kolommen van acht karakters. De eerste kolom vangt aan met plaats 0 op de regel (geheel links), de tweede met plaats acht, de derde met plaats 16 enz. Door de TAB functie gaat de cursor naar het begin van de eerstvolgende kolom. Daarbij worden de karakters die hij op de regel moet passeren, gewist. In het algemeen geldt dat PRINT CHR$(X) dezelfde werking heeft als het indrukken van de betreffende toets.
Het is heel goed mogelijk ook de karakters met een lager ASCII codegetal dan 32 op het scherm te printen met PRINT CHR$(1) terwijl voor X in CHR$(X) het ASCII codegetal voor het betreffende karakter uit lijst 2 moet worden vermeerderd met 64. Probeer maar eens:

screen 1: print chr$(1) chr$(65)

Ditmaal krijgt u geen ''A'' op het scherm!

Het volgende programma reageert op vier toetsen van het toetsenbord. Probeer uit te vinden welke dat zijn (denk aan SHIFT, CODE en GRAPH) door het indrukken van de diverse toetsen.

```
10 CLS
20 A$=INKEY$: IF A$="" THEN 20
30 A=ASC(A$)-166
40 IF A<1 OR A>4 THEN 120
50 ON A GOSUB 70,80,90,100
60 GOTO 20
70 PRINT"toets 1";: GOTO 110
80 PRINT"toets 2";: GOTO 110
90 PRINT"toets 3";: GOTO 110
100 PRINT"toets 4";: GOTO 110
110 PRINT" is gevonden", A$: RETURN
120 PRINT "MIS", A$: GOTO 20
```

In regel 40 is één van de logische bewerkingen toegepast die met de computer mogelijk zijn: AND, OR, NOT, XOR. EQV en IMP. Alleen als de waarde van A 1, 2, 3 of 4 is wordt doorgegaan met regel 50. In alle andere gevallen wordt in regel 120 aangegeven dat de verkeerde toets is ingedrukt. Achter het ON statement kan ook het GUSUB worden toegepast (regel 50). In dit geval wordt op vier verschillende regels een subroutine aangeroepen. Deze eindigt steeds met regel 110 waar RETURN de subroutine afsluit. Het programma springt dan terug naar de eerstvolgende regel achter het ON . . . GOSUB statement, hier regel 60. De logische bewerkingen worden in het volgende hoofdstuk verklaard.
Het voorgaande programma mag dan het ON . . . GOSUB statement demonstreren, op een andere manier kan het veel korter:

```
10 CLS
20 A$=INKEY$: IF A$=""THEN 20
30 A=ASC(A$)-166
40 IF A<1 OR A>4 THEN 60
50 PRINT"Toets"A"is gevonden",A$: GOTO 20
60 PRINT "mis", A$:GOTO 20
```

Er komen in dit programma geen nieuwe opdrachten voor, zodat het gemakkelijk te volgen is.
Het volgende programma geeft een voorbeeld van het gebruik van PRINT CHR$(X). Er komen tevens een paar nieuwe functies voor bewerkingen met stringvariabelen in voor.

```
10 A$="microcomputer"
20 B$="programmamaker"
30 C$=RIGHT$(A$,8)
40 D$=LEFT$(B$,9)
50 PRINT C$ CHR$(32) D$
```

In dit programma zijn de functies RIGHT$(X) en LEFT$(X) toegepast. Deze functies scheiden een aantal karakters af van een gegeven string.
De functie RIGHT$ scheidt karakters aan de rechterkant van de string af (hier A$). Het aantal karakters dat moet worden afgescheiden is gegeven (8). In regel 30 wordt dus het woord ''computer''

afgescheiden van A$ en in C$ geplaatst. De functie LEFT$ scheidt karakters aan de linkerkant van de gegeven string (B$) af. In regel 40 worden de negen karakters afgescheiden die samen het woord "programma" vormen. Door regel 50 worden de twee woorden achter elkaar op het scherm geplaatst.
CHR$(32) zorgt er voor dat de woorden door een spatie worden gescheiden. U ziet dat in regel 50 tussen C$. CHR$(32) en D$ geen scheidingstekens zijn toegepast. Dat is in principe dus niet nodig. Past u wel een scheidingsteken toe dan moet dat een ";" zijn.

8.3. De timer

In de computer is een zogenaamde "timer" ingebouwd. Dit is een teller die elke seconde vijftig keer wordt opgehoogd. Dat hangt samen met de frequentie van het lichtnet, die in Nederland nu eenmaal 50 Hz bedraagt. Het grootste getal dat de timer kan bevatten bedraagt 65535.
Heeft de teller deze stand bereikt dan springt hij naar 0 en begint opnieuw. De teller kan op een bepaalde waarde gezet worden met TIME = X, waarin X een getal van 0 tot en met 65535 kan zijn. De timer is te lezen met PRINT TIME. RUN het volgende programma:

```
10 CLS:TIME=0
20 PRINTCHR$(11)TIME
30 GOTO 20
```

In regel 10 wordt de timer "gereset", dat wil zeggen: op 0 gezet. In regel 20 wordt eerst de cursor met CHR$(11) naar HOME verwezen. Te beginnen op deze plaats wordt de inhoud van de timer geprint. Door deze regel voortdurend te herhalen zien we op het scherm de timer "lopen", waarbij hij door CHR$(11) steeds op dezelfde plaats van het scherm wordt geprint. Zie wat er gebeurt als u CHR$(11) vervangt door CHR$(12)!
Met behulp van de timer kunnen we een digitale klok op het scherm maken. Hiervoor kan het volgende programma dienen:

```
10 INPUT"seconden";S
20 INPUT"minuten";M
30 INPUT"uren";H
40 TIME=0: X=50: CLS
50 IF X=65500! THEN X=50: TIME=0
60 IF TIME=>X THEN S=S+1: X=X+50
70 IF S=60 THEN S=0: M=M+1
80 IF M=60 THEN M=0: H=H+1
90 IF H=24 THEN H=0
100 PRINT CHR$(11);
110 PRINT USING"###";H;M;S
120 GOTO 50
```

Het is altijd wel handig als u de klok op een bepaalde tijd kunt instellen. Daarvoor zijn dan de drie INPUT statements waarmee u de variabele S met de seconden, M met de minuten en H met de uren kunt invullen. In regel 40 wordt de timer op 0 en X op 50 gesteld. Daarmee wordt regel 50 overgeslagen. In regel 60 wordt vastgesteld of de timer de stand 50 heeft bereikt. Is dat zo dan moet S worden verhoogd.
Daarmee kan S 60 worden. In dat geval moet M met 1 worden verhoogd. Ook deze kan dan 60 geworden zijn en dan moet H worden opgehoogd. Bij deze klok kan H niet groter worden dan 23. Voor het printen van de klok is gebruik gemaakt van het PRINT USING statement. Dit statement heeft betrekking op alle drie de variabelen S, M en H. U ziet het resultaat op het scherm. Gebruikt u USING niet dan verplaatsen de getallen op het scherm als ze van 1 naar twee cijfers overgaan (van 9 naar 10 bijv.) of andersom, als ze van 59 naar 0 gaan. Elke keer als S is opgehoogd wordt X met 50 vermeerderd voor de volgende seconde. Om moeilijkheden te voorkomen kan X (en dus ook de timer) niet groter worden 65500. Is deze waarde bereikt dan wordt X weer op 50 en de timer op 0 gezet.

8.4. De functietoetsen

Welk doel de functietoetsen hebben zal u zo langzamerhand wel bekend zijn: door het indrukken van een toets vindt een werking plaats die van te voren ingeprogrammeerd is. Deze werking berust er op dat de opdracht (GOTO, LIST, RUN ed.) door het indrukken van een toets op het scherm wordt geprint, eventueel gevolgd door RETURN. Is RETURN niet ingeprogrammeerd dan kunt u zelf de tekst op het scherm aanvullen en de toets RETURN indrukken. Het voorprogrammeren van de toetsen wordt door de computer gedaan in een "startprogramma" dat in werking komt na het inschakelen van de computer. U kunt zelf ook de toetsen voorprogrammeren. Veronderstel dat u de RUN functie van toets f10 niet zo belangrijk vindt en daar liever PRINT" voor in de plaats hebt.
Hiervoor gebruikt u dan de volgende regel in de direct mode:

key 10, ''print'' + chr$(34)

De tekst die u op het scherm wilt hebben moet tussen aanhalingstekens staan: ''print''. Nu wilt u print'' op het scherm hebben en dat kan alleen door ''print'' samen te stellen met CHR$(34). Volgens lijst 2 is 34 het codegetal voor de aanhalingstekens.
Mocht u het ''list'' van toets f4 willen veranderen in list (RETURN) dan kan dat met

key 4, ''list'' + chr$(13)

CHR$(13) heeft de werking van RETURN. Probeer ook eens:

a$ = str$(atn(1)*4):key 3, a$

Hier is de functie STR$(X) toegepast. Deze functie is nodig als u de inhoud van een numerieke variabele of van een constante (ATN(1)*4) wilt gebruiken voor een stringvariabele (A$). De cijfers van het getal worden dan beschouwd als een rij karakters. Als de werking van een toets is veranderd dan ziet u de nieuwe functie op de onderste schermregel vermeld. Gaat het u vervelen steeds naar die tekst te moeten kijken dan is uitschakelen eenvoudig, toets maar in:

key off

De werking van de functie toetsen is daarmee niet uitgeschakeld. Dat kan wel met

key x,'' ''

waarin X het toetsnummer is.

8.5. De interval

We hebben al eens kennis gemaakt met een tijdsvertraging. In ons eerste programma uit paragraaf 4.1. was een vertraging nodig om de werking beter te kunnen volgen. Het is een programma dat een oplopende reeks van getallen op het scherm print. De duur van de vertraging was echter maar moeilijk te bepalen. Nu we van het bestaan van de timer afweten moet deze tijdsduur heel wat beter te bepalen zijn. We gebruiken het volgende programma:

```
10 TIME=0
20 IF TIME<>50 THEN 20
30 A=A+1: PRINT A;
40 IF A<>40 THEN 10
```

Het programma print ook nu een oplopende reeks van getallen op het scherm. Het grootste getal is 40. Nu is echter tussen het printen van elk getal een vertraging van 1 seconde. Dat wordt bereikt in regel 20.
Nadat eerst de timer in regel 10 op 0 is gezet wordt in regel 20 gewacht totdat de timer op 50 staat. Daarna wordt A opgehoogd en geprint. Zolang A nog geen 40 is wordt terug gegaan naar regel 10 waar de handeling zich herhaalt. Nu kent de MSX BASIC een statement dat deze handeling ook kan verrichten, het ON INTERVAL GOSUB. Zie hiervoor het volgende programma.

```
10 ON INTERVAL=50 GOSUB 40
20 INTERVAL ON
30 IF A=40 THEN END ELSE 30
40 A=A+1: PRINT A;: RETURN 30
```

Met de uitdrukking ON INTERVAL = 50 GOSUB 40 wordt bedoeld: ''Als de inhoud van de timer 50 hoger is onderbreek dan het lopende programma en ga naar de subroutine die met regel 40 aanvangt''. Dat betekent dat elke seconde deze subroutine wordt doorlopen en ook elke seconde A met 1 wordt verhoogd en op het scherm wordt afgebeeld. Op het einde van de subroutine staat het statement RETURN 30, hetgeen wil zeggen dat het hoofdprogramma moet worden voortgezet met regel 30. Uit regel 20 van het programma volgt dat de werking van de interval moet worden ingeschakeld. Dit statement moet als volgt worden gebruikt:

a. Aan het begin van het hoofdprogramma moet de lengte van de interval worden ingesteld en moet worden ingevoerd naar welke subroutine moet worden gesprongen.
b. De interval moet worden ingeschakeld.
c. Het hoofdprogramma moet een wachtcyclus bevatten (regel 30: IF A = 40 THEN END ELSE 30) zodat op de onderbrekingen (interrupts) kan worden gewacht.
d. Deze wachtcyclus moet een mogelijkheid hebben het programma te beëindigen als het zijn werk heeft voltooid (IF A = 40 THEN END).

Het tweede programma waarin een tijdsvertraging werd toegepast is het programma waarmee de wer-

king van het toetsenbord buffergeheugen werd uitgeprobeerd. Met het statement INTERVAL verloopt dat programma als volgt:

```
10 ON INTERVAL=500 GOSUB 40
20 INTERVAL ON
30 GOTO 30
40 INTERVAL OFF
50 A$=INKEY$
60 IF A$="" THEN RETURN 30
70 PRINT A$;: GOTO 50
```

Het hoofdprogramma wordt gevormd door de regels 10 tot en met 30 die niet veel meer inhouden dan het instellen van de lengte van de interval (10 seconden), het aangeven van de eerste regel van de subroutine, het inschakelen van de interval en de wachtcyclus (30 GOTO 30). In de subroutine wordt de interval uitgeschakeld (INTERVAL OFF). Daarna wordt door middel van INKEY$ het buffergeheugen gelezen en in regel 70 het karakter geprint, tot alle ingetoetste karakters gelezen zijn. Dan wordt naar de wachtcyclus teruggegaan. De bedoeling van dit programma is dat u een aantal toetsen aanslaat in de intervaltijd.

Welke dat zijn krijgt u daarna te zien. Is het printen achter de rug dan kunt u zoveel toetsen aanslaan als u wilt, de bijbehorende tekens worden niet meer afgedrukt omdat door INTERVAL OFF in de subroutine de interval is uitgeschakeld en het hoofdprogramma niet meer wordt onderbroken. Dit programma is geen meesterwerk van programmeerkunst. Het komt uiteindelijk in een doelloze wachtcyclus terecht en kan die niet meer op een ''nette'' manier verlaten (aan voorwaarde d is niet voldaan).

9. Het veranderen van de inhoud van geheugenplaatsen

9.1. Het lezen van en het schrijven naar geheugenplaatsen

Probeer het volgende in de direct mode:

a = peek(49152): print a

U hebt nu met de functie PEEK de inhoud gelezen van geheugenplaats 49152. Zoals u in hoofdstuk 1 hebt kunnen lezen is dit een geheugenplaats in het 32 Kbyte RAM gedeelte dat vrij is voor het opslaan van programma's en dergelijke. Laat nu ook in de direct mode de computer de volgende handeling verrichten en kies voor X een getal dat kleiner is dan 256 en *anders* is dan het getal dat u net hebt gelezen:

poke 49152,x

POKE is een statement waarmee een geheugenplaats een bepaalde inhoud kan worden gegeven. In hoofdstuk 1 is vermeld dat deze inhoud nooit groter kan zijn dan 255. Probeer nu:

print peek(49152)

Nu wordt zonder eerst een variabele te gebruiken de inhoud van geheugenplaats 49152 op het scherm geschreven. Het getal dat nu wordt gelezen is gelijk aan dat wat u eerder voor X hebt gebruikt. Probeer nu de volgende regel eens:

print peek(&HC000)

U hebt nu de inhoud van dezelfde geheugenplaats gelezen en dus moet &HC000 overeenkomen met 49152. In dit hoofdstuk zal dit worden verklaard. Terloops is al opgemerkt dat er ook nog RAM geheugenelementen zijn die niet zijn opgenomen in de geheugenruimte die in afb. 1 is aangegeven. Zo is er een 16 Kbyte geheugenblok dat wordt gebruikt door de Video Display Processor (VDP) die alles verzorgt wat met het schermbeeld te maken heeft. Verder zijn geheugenelementen aanwezig die de verbinding verzorgen tussen de processor en de randapparatuur. Deze zijn dan voor het in- en het uitvoeren van de gegevens van of naar het toetsenbord, het beeldscherm, het cassettedeck, de floppy disk drive en de printer.
Deze geheugenelementen worden de *in-* en *uitgangspoorten* genoemd. In principe is elk geheugenelement een *register,* omdat het een getal kan bevatten en bewaren. In het algemeen worden echter slechts de geheugenelementen die een bepaalde functie hebben registers genoemd.
Zo bevat de VDP ook een aantal registers.
Al de geheugenelementen die zich niet in de "normale" geheugenruimte van de processor bevinden (in- en uitgangspoorten, video RAM, VDP registers) kunnen slechts worden ingeschreven en uitgelezen met speciale commando's, statements en functies. Probeer dit maar eens:

vdp(7) = 68

Nu is alles van uw scherm verdwenen. Dat komt alleen maar omdat de kleur van de tekens nu gelijk is aan die van de achtergrond. Met toets f6 kunt u het scherm weer "normaal" krijgen.
Om een aantal mogelijkheden die in deze computer zijn ingebouwd te kunnen benutten zullen we enig inzicht moeten hebben in een aantal getallenstelsels.

9.2. Positieve binaire getallen

De computer is een elektronisch apparaat waarin getallen moeten worden weergegeven met spanningen. Nu zijn er diverse methoden om spanningen evenredig aan een getal te laten zijn. De eenvoudigste is echter die, waarbij slechts twee toestanden worden gedefinieerd, geen spanning is 0 en wel spanning is 1. Dit noemt men digitaal systeem. Elk getallenstelsel is een digitaal getallenstelsel. In dit geval moeten we gebruikmaken van het binaire (tweetallige) stelsel waarbij we slechts twee "cijfers" kennen, de 0 en de 1. In wezen is dat veel een-

voudiger dan het tientallig stelsel waar veel meer combinaties mogelijk zijn, zoals blijkt uit het grote aantal tafels van vermenigvuldiging. Daar hebben we bij het binaire stelsel geen last van! Misschien hebt u al kennis gemaakt met de binaire rekenwijze.
Voor hen die echter nog niet op de hoogte zijn volgt hier een beknopte verklaring van datgene waarin op zijn minst enig inzicht nodig is.
In het binaire stelsel kunt u slechts twee getallen met de beschikbare symbolen aangeven, de 0 en de 1. Voor grotere getallen moet u de symbolen gaan combineren: $2 \hat{=} 10$, $3 \hat{=} 11$, $4 \hat{=} 100$, $5 \hat{=} 101$ enz.

tientallig	tweetallig	macht van 2
0	0	–
1	1	2^0
2	10	2^1
3	11	2^1+2^0
4	100	2^2
5	101	2^2+2^0
6	110	2^2+2^1
7	111	$2^2+2^1+2^0$
8	1000	2^3
9	1001	2^3+2^0
10	1010	2^3+2^1
11	1011	$2^3+2^1+2^0$
12	1100	2^3+2^2
13	1101	$2^3+2^2+2^0$
14	1110	$2^3+2^2+2^1$
15	1111	$2^3+2^2+2^1+2^0$
16	10000	2^4

Lijst 4

Zoals elk cijfer in het tientallig stelsel een *plaatswaarde* heeft, zo heeft ook elk cijfer in het tweetallig stelsel een plaatswaarde.
Daarbij is alleen het cijfer 1 belangrijk. Het cijfer 0 kan moeilijk een waarde hebben. Bij het tweetallig stelsel wordt overigens niet van cijfers gesproken maar van ''bits''. In het volgende getal van acht enen is van elk bit de plaatswaarde gegeven met een macht van twee ($2^0=1$, $2^1=2$ $2^2=4$ enz. zie ook lijst 4).

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
1 1 1 1 1 1 1 1

We kunnen van elk willekeurig binair getal de decimale waarde vinden door de plaatswaarde van elk bit dat 1 is te bepalen en de gevonden waarden bij elkaar op te tellen, dus bijvoorbeeld:

2^6 2^4 2^3 2^1 2^0
1 0 1 1 0 1 1 $\hat{=}$
$2^0+2^1+2^3+2^4+2^6=$
$1+2+8+16+64=91$

De geheugenplaatsen in de computer zijn in principe registers die binaire getallen kunnen bevatten van acht bits. Deze bits kunnen slechts 1 of 0 zijn. Het binaire getal 1 vinden we in een geheugenplaats daarom terug als 00000001 en het getal 1011011 als 01011011. Het binaire getal wordt dus steeds aangevuld met nullen totdat het acht bits bevat. Het getal met de grootste waarde dat een register kan bevatten is het getal waarvan alle bit 1 zijn:

$11111111 \hat{=} 2^0+2^1+2^2+2^3+2^4+2^5+2^6+2^7=255$

Dit is in overeenstemming met wat reeds is vermeld: het grootste getal dat een geheugenplaats kan bevatten heeft een decimale waarde van 255. Een binair getal van acht bits wordt een byte genoemd.

Een 64 Kbyte geheugen heeft 65536 geheugenplaatsen van (bij de MSX computer) acht bits. De bits worden in overeenstemming met de plaatswaarde genummerd, bit 0 heeft plaatswaarde 2^0, bit 1 heeft de plaatswaarde 2^1 enzovoorts.

Bit 7 6 5 4 3 2 1 0
0 1 1 1 0 1 0 1

Van bovenstaand getal hebben de bits 0, 2, 4, 5 en 6 de waarde 1.

9.3. Negatieve binaire getallen

Als u een bandrecorder of een cassettedeck met een bandteller hebt dan kunt u deze teller, ook als de band niet aan het begin staat, met een resetknop op nul stellen. Als u daarna de opwikkelspoel één omwenteling vooruit laat maken dan staat de teller op 1. Laat u de opwikkelspoel vanuit 0 één omwenteling terug maken dan zou de teller -1 moeten aangeven. Hij geeft echter (bij een teller met drie cijfers) 999 aan. Het getal 999 op de teller komt dus kennelijk overeen met -1.

Eenzelfde methode heeft men toegepast bij uw computer. Op een bepaalde manier is het mogelijk om positieve binaire getallen in te voeren met een lengte van maximaal vijftien bits, *geen* zestien bits dus. In decimale vorm heeft dit getal een waarde van 32767. Voor deze getallen worden twee geheugenplaatsen gebruikt waarin dus plaats is voor zestien bits. Dat betekent dat de positieve getallen in elk geval te herkennen zijn omdat het hoogstwaardige bit, het zestiende bit, altijd 0 is. Ook dit getal moet namelijk worden aangevuld met nullen tot alle elementen van de geheugens zijn gevuld. Het binaire getal −1 wordt gevormd door van de geheugenelementen als ze het getal 0 bevatten, 1 af te trekken, zoals dat ook bij de teller van de recorder was gebeurd. Rekenkundig verloopt dat als volgt:

```
  1 00000000 00000000
    00000000 00000001 -
    --------------------
    11111111 11111111
```

Om de aftrekking mogelijk te maken moet een 1 voor het getal worden geplaatst. Het zestiende bit van het getal is een 1 en daarom zal het u duidelijk zijn hoe het teken van de binaire getallen in de computer te herkennen is, aan het zestiende bit. De negatieve binaire getallen worden dus gevormd door de absolute waarde van het getal af te trekken van 1 00000000 00000000. Het grootste negatieve binaire getal dat de computer kan verwerken is daarom 10000000 00000000. Dit komt overeen met het decimale getal −32768. Zo hebben we tevens de grenzen gevonden van de getallen die een integer variabele kan bevatten!

9.4. Hexadecimale en octale getallen

Het hexadecimale getallenstelsel is een zestientallig stelsel. Het vullen van geheugenplaatsen met binaire getallen is nu niet bepaald een handig systeem, de getallen zijn wat lang. Nu gebruikt men daarbij het zestientallig stelsel omdat dit nu juist zo mooi aansluit bij het tweetallig. Elk binair getal van acht bits kan geschreven worden met twee "cijfers" van het hexadecimale stelsel. Voor het zestientallige stelsel hebben we ook zestien symbolen voor de cijfers nodig (ook wel "digits" genoemd). Aan de symbolen van het tientallige stelsel (0 tot en met 9) komen we dus te kort en daarom zijn een aantal symbolen "geleend" van het alfabet, de A, B, C, D, E en de F.
In lijst 5 zijn de waarden van de cijfers uit het zestientallig stelstel gegeven in combinatie met de eerste zestien binaire getallen.

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

Lijst 5

Omzetting van getallen uit het binaire stelsel in het hexadecimale stelsel en andersom is daarom zo eenvoudig omdat met elke vier bits uit het binaire stelsel precies zestien getallen zijn weer te geven.
Voor de omzetting van binair naar hexadecimaal worden de bits van de binaire getallen in groepjes van vier verdeeld, de zogenaamde "tetraden" (engels: nibbles). Verder wordt elke tetrade vervangen door het hexadecimale cijfer dat met de waarde van de tetrade overeenkomt.
Omdat bij microcomputers steeds met binaire getallen van acht bits (inhoud van geheugenplaatsen) of zestien bits (adressen van geheugenplaatsen) wordt gewerkt is het hexadecimale getallenstelsel juist hier zo populair geworden. De omzetting van het binaire getal 01101101 naar zijn hexadecimale waarde verloopt als volgt: eerst wordt het getal in twee tetraden verdeeld; 0110 1101; daarna worden de tetraden vervangen door de gelijkwaardige digits; 6D. Bij de MSX BASIC wordt een binair getal aangegeven door een B voor het getal te plaatsen en een hexadecimaal getal door een H voor het getal te plaatsen:

B01101101 ≙ H6D

In tegenovergestelde richting is de omzetting al even eenvoudig, de digits worden vervangen door de tetraden van gelijke waarde die we zonder meer uit lijst 5 kunnen aflezen:

H3C5E ≙ B0011 1100 0101 1110

Zoals een binair getal van acht bits een byte wordt genoemd, wordt ook een hexadecimaal getal van twee digits een byte genoemd. Elke byte kent daarom een hoge digit en een lage digit. Hierbij is de plaatswaarde van de hoge digit decimaal 16. Voor het bepalen van de decimale waarde van een byte bepalen we de waarde van elke digit afzonderlijk (lijst 5) en vermenigvuldigen die met de plaatswaarde. H5E ≙ 5*16+14*1=94

Een adres van een geheugenplaats is twee bytes en omvat dus vier digits. Het is te verdelen in een hoge byte en een lage byte waarbij de plaatswaarde van de hoge byte 256 is. Willen we het adres HAC45 in een decimaal getal uitdrukken dan bepalen we de waarde van de lage byte: 4*16+5*1=69. De waarde van de hoge byte: 10*16+12*1=172. De decimale waarde van het adres is nu: 172*256+69*1=44101. Op welke manier u deze berekeningen door de computer kunt laten doen (waarvoor heeft u dat ding anders) komen we verderop in dit hoofdstuk terug.

Een veel minder toegepast getalstelsel is het achttallige stelsel of octale stelsel. Hierbij zijn slechts acht cijfers, van 0 tot en met 7.
De toepasbaarheid berust op het gegeven dat een groep van drie bits uit een binair getal precies een waarde van 0 tot en met 7 kan hebben.

Lijst 6 geeft het onderling verband weer tussen het octale, het binaire en het decimale stelsel.

octaal	binair	decimaal
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7

Lijst 6

Het omzetten van een binair getal in een octaal getal gaat al even eenvoudig als het omzetten naar een hexadecimaal getal. Stel dat het binaire getal B11010110 moet worden omgezet. Eerst het getal in groepjes van drie verdelen: B 011 010 110. Merk op dat het getal is aangevuld door er een nul voor te plaatsen! Nu de groepjes vervangen door hun octale waarden:

B 011 010 110 ≙ O 326

Voor het getal achter het ≙ teken is een O geplaatst (geen nul) om aan te geven dat het een octaal getal is.

9.5. Logische bewerkingen met binaire getallen

Met binaire getallen kunnen logische bewerkingen worden uitgevoerd. Deze logische bewerkingen zijn al genoemd in hoofdstuk 8. Van deze bewerkingen zullen in deze paragraaf de definities worden gegeven.
De logische bewerkingen hebben een wiskundig stelsel als achtergrond dat de "logica" wordt genoemd. Deze logica is gebaseerd op uitspraken die slechts "waar" of "onwaar" kunnen zijn. Daarmee zijn we weer terug bij het binair stelsel waarbij er ook slechts twee mogelijkheden kunnen zijn. In het algemeen wordt "waar" met een 1 gewaardeerd en "onwaar" met een 0. We zullen ons hier beperken tot bewerkingen met twee variabelen die elk slechts 1 of 0 kunnen zijn (of indien u dat zo wilt, waar of onwaar). Deze variabelen zullen steeds met A en B worden benoemd. Het resultaat van een bewerking kan dan X zijn. We zullen ons alleen bezig houden met de definities van deze bewerkingen.

De bewerkingen èn (AND).
Een logische bewerking met twee binaire getallen heeft steeds plaats tussen de twee gelijkwaardige bits van de getallen, bit 0 van getal A met bit 0 van getal B, bit 1 van getal A met bit 1 van getal B enz. Er zijn dan vier mogelijkheden die de volgende tabel opleveren:

A	B	A én B
0	0	0
0	1	0
1	0	0
1	1	1

Het resultaat van een èn-bewerking met twee bits is slechts dan 1 als beide bits 1 zijn.

Voorbeeld:

	7	6	5	4	3	2	1	0
getal A	1	0	1	1	0	1	1	0
getal B	1	0	0	0	1	1	0	1
resultaat A èn B	1	0	0	0	0	1	0	0

De overeenkomstige bits van getal A en getal B worden bit voor bit vergeleken. We laten de bewerking dus plaatsvinden tussen de gelijkwaardige bits van getal A en getal B. Alleen als beide bits 1 zijn is het bit in het resultaat ook 1. De andere bits zijn 0 (een andere mogelijkheid is er niet).

De bewerking of (OR).
Ook nu vindt de bewerking steeds plaats tussen twee gelijkwaardige bits. Uit de definitie voor de of-bewerking volgt de volgende tabel:

A	B	A of B
0	0	0
0	1	1
1	0	1
1	1	1

Het resultaat van een of-bewerking met twee bits is slechts dan 0 als beide bits 0 zijn.

Voorbeeld:

getal A	1	0	1	1	0	1	1	0
getal B	1	0	0	0	1	1	0	1
resultaat A of B	1	0	1	1	1	1	1	1

De overeenkomstige bits van getal A en getal B worden paar voor paar vergeleken. Alleen als beide bits 0 zijn is het overeenkomstige bit in het resultaat ook 0. De andere bits zijn 1.

De bewerking niet (NOT).
De bewerking niet heeft slechts betrekking op één variabele. De tabel is:

A	niet A
0	1
1	0

Het resultaat van een niet-bewerking met één bit is 0 als het bit 1 is en is 1 als het bit 0 is.

Deze bewerking wordt ook wel de inverse (omgekeerde) genoemd en wordt uitgevoerd op alle bits van een binair getal. Moet daarvan de inverse waarde worden genomen dan moet van elk bit de inverse worden bepaald.

Getal A	0	1	1	0	1	1	0	1
NOT A	1	0	0	1	0	0	1	0

In principe heeft u aan de en-, de of- en de niet-bewerking genoeg om alle problemen op dit terrein te kunnen oplossen. Er zijn echter ook een aantal combinaties van deze bewerkingen gedefinieerd. Omdat voor deze bewerkingen in de MSX computer ook statements beschikbaar zijn zal voor de volledigheid ook deze definities worden gegeven.

De exclusief of-bewerking (XOR).
Deze bewerking heeft weer betrekking op twee variabelen en kan met de volgende tabel worden beschreven:

A	B	A exof B
0	0	0
0	1	1
1	0	1
1	1	0

Het resultaat van een exof-bewerking is dan slechts 0 als de overeenkomstige bits *gelijk* zijn.

Voorbeeld:

getal A	1	0	1	1	0	1	1	0
getal B	1	0	0	0	1	1	0	1
resultaat A exof B	0	0	1	1	1	0	1	1

De overeenkomstige bits van getal A en getal B worden vergeleken. Als beide bits 0 zijn of als beide bits 1 zijn zijn de overeenkomstige bits in het resultaat 0. De andere bits zijn 1.

De bewerking implicatie (IMP).
Deze bewerking heeft ook weer betrekking op de gelijkwaardige bits van twee variabelen en wordt als volgt gedefinieerd:

A	B	A impl B
0	0	1
0	1	0
1	0	1
1	1	1

Het resultaat van een impl-bewerking is slechts dan 0 als A gelijk is aan 0 en A en B verschillend zijn.

Voorbeeld:

getal A		1	0	1	1	0	1	1	0
getal B		1	0	0	0	1	1	0	1
resultaat A impl B		1	1	1	1	0	1	1	0

De bewerking equivalentie (EQV).
De nu volgende tabel geeft hiervan de definitie. Merk op dat de inverse van deze bewerking gelijk is aan de exof-bewerking.

A	B	A equi B
0	0	1
0	1	0
1	0	0
1	1	1

Het resultaat van een equi-bewerking is slechts dan 1 als beide variabelen gelijk zijn.

Voorbeeld:

getal A		1	0	1	1	0	1	1	0
getal B		1	0	0	0	1	1	0	1
resultaat A equi B		1	1	0	0	0	1	0	0

Wilt u de binaire waarde weten van een decimaal getal dan kunt u hiervoor het volgende programma gebruiken:

```
10 INPUT"getal";G
20 IF G>255 THEN 10
30 FOR X=0 TO 7
40 I=(G AND 2^X)/2^X*10^X
50 R=R+I: NEXT X
60 PRINT R
```

In regel 10 wordt het decimale getal ingevoerd. Regel 20 voorkomt dat een groter getal dan 255 kan worden ingevoerd. Elk getal van 0 tot en met 255 is in de computer als een binair getal opgeslagen waarvan met (G AND 2^X)/2^X bit voor bit kan worden bepaald wat de waarde van de bits is, 1 of 0. Dit is dus een voorbeeld van het gebruik van een èn-bewerking tussen twee getallen, namelijk het getal G en het getal dat gevormd wordt met 2^X. Door de overeenkomstige bits te vermenigvuldigen met 10^X en op te tellen wordt een *decimaal* getal verkregen met alleen maar enen en nullen dat er als een binair getal uitziet.

Stel het getal is 213 (11010101) en bij de vijfde doorgang van de lus gevormd in de regel 30 tot en met 50 is X = 4.

Getal 213	Binair	1	1	0	1	0	1	0	1
2↑X = 2↑4 = 16	Binair	0	0	0	1	0	0	0	0
	AND								
G AND 2↑X = 16	Binair	0	0	0	1	0	0	0	0

(G and 2↑X)/2↑X = 16/16 = 1
(G AND 2↑X)/2↑X*10↑X = 1*10000 = 10000, dit is een decimaal getal!

Regel 50 zorgt dat de afzonderlijke decimale getallen I, die steeds een hogere macht van 10 zijn, bij elkaar worden opgeteld en zo het getal vormen dat er als een binair getal uitziet.
Nu is er een heel wat eenvoudiger manier om de binaire waarde van een decimaal getal te vinden. Probeer maar eens:

```
print bin$(3481)
```

De functie BIN$(X) geeft u de binaire waarde van het decimale getal X. Voor X kunt u positieve getallen nemen tot maximum 65355 en negatieve getallen met −32768 als grens. Probeer eens:

```
print bin$(-1), bin$(-32768)
```

en bedenk wat daarover is vermeld in de paragraaf over negatieve binaire getallen!

Nu moet u niet denken dat u zich voor niets met het bovenstaande programma hebt bezig gehouden. De bedoeling van alle programma's in dit boek is om u kennis te laten nemen met het toepassen van de diverse opdrachten, als het kan met betrekking tot het behandelde onderwerp.
Ga de werking van het volgende programma eens na. Het doel is een binair getal om te zetten in een decimaal getal.

```
10 INPUT"getal";G$
20 IF LEN(G$)<>8 THEN 10
30 FOR X=0 TO 7: P=8-X
40 I=VAL(MID$(G$,P,1))*2^X
50 R=R+I: NEXT X
60 PRINT R
```

In regel 10 wordt het binaire getal ingevoerd in een stringvariabele.
In regel 20 is de voor ons nieuwe functie LEN(G$). De lengte van een string (het aantal karakters in die string) kan met deze functie als een getal worden uitgedrukt. Bevat G$ acht karakters (in dit geval enen en nullen) dan is het resultaat van LEN(G$) gelijk aan 8. Regel 20 zorgt er voor dat alleen binaire getallen van acht bits kunnen worden ingevoerd. Kleinere getallen moeten daarom met nullen worden aangevuld. Het binaire getal is nu ingevoerd als een reeks van karakters in een string. Karakter voor karakter moet nu worden bepaald wat de waarde hiervan is. Om de karakters uit de string te lichten gebruiken we de functie MID$(G$,P,1). Deze functie werkt ongeveer gelijk aan LEFT$ of RIGHT$. Uit de stringvariabele wordt een aantal karakters afgescheiden. Het aantal wordt gegeven met het getal tussen de haakjes (hier dus 1). De plaats wordt ook aangegeven met een getal (hier P). Dit getal geeft de plaats van het eerste karakter dat uit de string moet worden gelicht, te rekenen vanaf het meest linkse karakter van de string. Stel het binaire getal is 11010101. Bij de derde doorgang van de lus in de regels 30 tot en met 50 is X = 2 en P = 8 – 2 = 6.
MID$(G$,P,1) wordt daardoor MID$(G$,6,1). Hiermee wordt het zesde bit van links af uit het getal gelicht, hier een 1 (slechts één bit want het tweede cijfer in MID$ is 1).

Dit karakter moet nu nog worden omgezet tot een cijfer in een numerieke variabele waarmee berekeningen kunnen worden gemaakt.
Hiertoe dient de functie VAL(X$). Als X$ een reeks van karakters is dat een getal moet voorstellen dan wordt door VAL(X$) de numerieke waarde van dat getal bepaald. In ons voorbeeld is I = VAL(''1'') zodat de inhoud van I 1 is. Nu wordt de decimale plaatswaarde van het bit bepaald met I*2^X = 1*2^X = 1*2^2 = 4. Zo wordt de plaatswaarde van elk bit bepaald en de gevonden waarden worden in regel 50 bij elkaar opgeteld tot het juiste decimale getal is gevonden.
De functie VAL is de tegenvoeter van STR$(X).

Probeer in de direct mode:

```
x$ = ''1'': print val(x$); val(''0'')
print val(''345'');val(''3,4,5'')
```

Probeer ook

```
x$ = ''MSX BASIC programmataal''
print mid$(x$,11,9)
```

Natuurlijk is er ook een makkelijke manier om van een binair getal een decimaal getal te maken. Probeer het volgende eens:

```
a = &b11011101: print a
```

Met het voorvoegsel &B kunt u elke numerieke variabele vullen met een binair getal, dat wil zeggen: u kunt het getal binair invoeren. Een PRINT opdracht geeft steeds de decimale waarde van het getal in de variabele. Hexadecimale en octale waarden van getallen kunnen op overeenkomstige wijze worden behandeld:

```
print hex$(32561)
a = &h7f31: print a

print oct$(24539)
print &o57733

print hex$(&b11101011)
```

9.6. Het werken met een masker

Voor sommige handelingen door de computer zal het nodig zijn dat een bepaald bit van een geheugenplaats een bepaalde waarde moet worden gegeven, terwijl de zeven andere bits de waarde die ze hadden, moeten blijven houden. Laten we aannemen dat het derde bit van een getal (dus bit 2) de waarde 0 moet krijgen en de andere bits niet mogen veranderen. We passen op dit getal dan een èn-bewerking toe met een ander getal - een zogenaamd ''masker'' - waarvan het derde bit 0 is en de andere bits 1 zijn:

Eerste voorbeeld:

getal A	0	1	1	0	1	1	0	1
Masker M	1	1	1	1	1	0	1	1
resultaat A AND M	0	1	1	0	1	0	0	1

Decimaal:

A = 109
M = 251
A AND M = 105

De èn-bewerking toepassen betekent dat een bit in het resultaat slechts dan 1 is als de overeenkomstige bits in beide getallen ook 1 zijn. Uit het voorbeeld volgt dat bit 2 in het resultaat 0 is terwijl de andere bits dezelfde waarde hebben als die in getal A. Bit 2 van het resultaat is hoe dan ook 0, ook als bit 2 in getal A al 0 was. In het algemeen moet een èn-bewerking worden toegepast met een masker waarvan alle bits 1 zijn behalve het bit waarvan het overeenkomstige bit in het resultaat 0 moet zijn.

Nu het geval waarbij een bepaald bit 1 moet worden. We gaan uit van het resultaat van het vorige voorbeeld en noemen dat getal B. Daarin moet bit 2 weer 1 worden. Het resultaat moet dan gelijk zijn aan getal A.

Tweede voorbeeld:

getal B	0	1	1	0	1	0	0	1
Masker M	0	0	0	0	0	1	0	0
resultaat B OR M	0	1	1	0	1	1	0	1

Decimaal:

B = 105
M = 4
B OR M = 109 Zie getal A

Nu moet er een of-bewerking bewerking worden toegepast op getal B en een masker waarvan alle bits 0 zijn, behalve bit 2. Vergelijken we de overeenkomstige bits en noteren we in het resultaat slechts dan een 0 als beide bits 0 zijn dan hebben we een of-bewerking uitgevoerd. Het nu verkregen resultaat is gelijk aan getal A.

In overeenstemming met deze voorbeelden voeren we de volgende bewerkingen in de direct mode uit:

poke 49152,109

De inhoud van geheugenplaats 49152 heeft nu de waarde van getal A gekregen.

poke 49152, peek(49152) and 251

Door PEEK(49152) wordt de geheugenplaats gelezen en het getal 109 gevonden. Dan wordt de AND bewerking hiermee en met het getal 251 uitgevoerd en het resultaat (eerste voorbeeld) weer in geheugenplaats 49152 geschreven met POKE 49152, . . .

print peek(49152)

Het resultaat moet nu 105 zijn.

poke 49152, peek(49152) or 4

Nu wordt de bewerking van het tweede voorbeeld uitgevoerd.

print peek(49152)

Het resultaat moet weer 109 zijn. Probeer nu nogmaals:

poke 49152, peek(49152) or 4
print peek(49152)

Naar aanleiding van het tweede voorbeeld zou u kunnen concluderen dat de or-bewerking hetzelfde is als optellen:

105 OR 4 = 109

Deze conclusie is *niet* juist. Uit het laatste wat we hebben geprobeerd volgt:

109 OR 4 = 109

10. Het toevalsgetal

10.1. De RND(X) functie

Een van de vele mogelijkheden die de computer heeft is het genereren van een getal waarop we zelf geen invloed kunnen uitoefenen. Het is een getal *tussen* 0 en 1, 0 en 1 behoren daar dus niet bij en heeft 14 cijfers achter de decimaalpunt. Probeer maar eens:

a = rnd(1): print a: print rnd(1)

U krijgt nu twee geheel verschillende getallen op uw scherm afgedrukt. Probeer ook

print rnd(0)

U krijgt nu hetzelfde getal als het tweede. Een andere mogelijkheid is

print rnd(−1), rnd(−1)

U krijgt nu twee keer hetzelfde getal op het scherm. Voor een goede werking van de RND(X) functie moet voor X steeds een positief getal worden gekozen. De grootte van X maakt verder niets uit. RUN het volgende programma:

```
10 FOR X=0 TO 8
20 PRINT RND(1): NEXT
```

U krijgt nu negen totaal verschillende getallen op het scherm te zien. Het lijkt er op alsof deze getallen volkomen "toevallig" door de computer zijn gegenereerd. RND is dan ook een afkorting van random, "in het wilde weg". Zo erg willekeurig zijn deze getallen echter nu ook weer niet. RUN het programma nogmaals en u ziet weer dezelfde getallen op het scherm. De computer berekend deze getallen, elk getal is weer afhankelijk van het voorgaande. Hij doet het echter zo dat er totaal geen lijn in valt te ontdekken. U kunt hem urenlang deze getallen laten genereren zonder dat u er twee tegenkomt die gelijk aan elkaar zijn. Willen we echter echte toevalsgetallen hebben dan moeten we de computer niet steeds door RUN met hetzelfde getal laten beginnen maar dan moeten we willekeurig in de getallencyclus binnenkomen. Dit kan met het volgende programma.

```
10 A$=INKEY$
20 A=RND(1)
30 IF A$<>CHR$(24) THEN 10
40 FOR X=0 TO 8
50 PRINT RND(1): NEXT
```

In regel 10 van dit programma ziet u het INKE$ statement. Of er nu een toets is ingedrukt of niet direct daarop wordt variabele A gevuld met een randomgetal. Is geen toets ingedrukt of is de verkeerde toets ingedrukt dan wordt weer teruggegaan naar regel 10 zodat daarna weer een ander randomgetal wordt gegenereerd. Is echter op de SELECT toets gedrukt (zie lijst 3) dan komen we bij de regels 40 en 50 terecht waardoor 9 randomgetallen op het scherm zullen verschijnen. Op welk moment we nu in de getallencyclus binnen komen hangt geheel van de tijd af die is verstreken tussen het starten van het programma en het indrukken van de toets SELECT. Deze tijd zal altijd willekeurig zijn zodat nu werkelijk van toevalsgetallen sprake is.

Nu hebben we aan dit soort toevalsgetallen niet zo veel als ze niet in de vorm zijn zoals we ze nodig hebben. Vaak zijn dat gehele getallen tussen twee grenzen, zoals we voor het imiteren van een dobbelsteen hele getallen van 1 tot en met 6 nodig hebben. We hebben in hoofdstuk 5 een programma gegeven voor het bepalen van de volgorde van de namen van en aantal kinderen. Dit gaat nu gebeuren met behulp van de computer als dobbelsteen. Ten eerste moeten we getallen groter dan 1 zien te krijgen. Dit is mogelijk door te vermenigvuldigen. Verander regel 50 van het laatste programma in:

```
50 PRINT 6*RND(1): NEXT
```

U runt het programma. Er komen in elk geval ook getallen in voor die groter zijn dan 1. Nu gaan we er hele getallen van maken.

```
50 PRINT INT(6*RND(1)): NEXT
```

Nu krijgen we een reeks hele getallen. Hierbij komt de 6 echter niet voor, de 0 wel. Dit is als volgt te verklaren.
Stel de computer genereert het getal .02319145389518. Vermenigvuldigd met 6 wordt dat .13914872337108. Dit is nog steeds een getal kleiner dan 1 en de integer hiervan is dus 0. Het grootste getal dat kan worden gegenereerd is .99999999999999. Vermenigvuldigd met zes geeft dat: 5.99999999999994. De integer hiervan is 5. Regel 50 moet daarom worden:

```
50 PRINT INT(6*RND(1))+1: NEXT
```

Het programma uit hoofdstuk 5 wordt nu als volgt:

```
10 CLS
20 PRINT"Druk op toets SELECT"
30 A$=INKEY$
40 A=RND(1)
50 IF A$<>CHR$(24) THEN 30
60 CLS: FOR X=1 TO 6
70 A=INT(6*RND(1))+1
80 ON A GOTO 90,100,110,120,130,140
90 PRINT "Jan";: GOTO 150
100 PRINT "Piet";: GOTO 150
110 PRINT "Sonja";: GOTO 150
120 PRINT "Hans";: GOTO 150
130 PRINT "Marit";: GOTO 150
140 PRINT"Ria";
150 PRINT " is ";X: PRINT: NEXT X
```

Nu komt er beslist ruzie! De computer ''gooit'' net als bij een gewone dobbelsteen soms wel meerdere keren het zelfde nummer, waardoor het ene kind vaker en het andere in het geheel niet aan de beurt komt.
Zo deugt het programma dus ook niet. Voeg de volgende regels eens toe:

```
70 A(X)=INT(6*RND(1))+1
72 FOR P=1 TO X
74 IF A(P)=A(X) AND P<>X THEN 70
76 NEXT P
80 ON A(X) GOTO 90,100,110,120,130,140
```

Om te beginnen is de variabele A in regel 70 veranderd in de variabele A(X), zodat de eens gekozen nummers worden onthouden. Door de regels 72 tot en met 76 wordt steeds gecontroleerd of het laatst gekozen nummer niet al eerder aan de beurt is geweest. Stel dat al twee nummers zijn gegenereerd (A(1) en A(2)) en nu een derde wordt gegenereerd. Door de lus in de regels 72 tot en met 76 wordt eerst A(1) met A(3) vergeleken (P = 1 en X = 3), daarna wordt A(2) met A(3) vergeleken (P = 2 en X = 3) en daarna wordt A(3) met A(3) vergeleken (P = 3 en X = 3). Dit laatste is uiteraard een gelijkheid. Deze wordt door de AND functie uitgesloten. Verder gaat de teller niet want hij gaat maar tot P de waarde van X heeft (3). Zo u ziet kunnen de logische functies ook op uitspraken worden losgelaten. De eerste uitspraak is ''A(P) = A(X)'' en dit kan waar en onwaar zijn (bijvoorbeeld A(P) = 5 en A(X) = 2, de uitspraak is *niet* waar). De tweede uitspraak is P < > X en ook dit kan waar of onwaar zijn. Alleen als beide uitspraken waar zijn, A(P) is inderdaad gelijk aan A(X) en daarbij is ook P niet gelijk aan X, wordt teruggegaan naar regel 70 voor het genereren van een nieuw randomgetal, waarbij X hetzelfde blijft.
In dit programma is een *array* toegepast. Dit is een reeks van variabelen (in dit geval numerieke variabelen) met een subscript, een getal tussen haakjes achter de variabele naam, dus (A(1), A(2) enz. Er behoeft echter niet gedimensioneerd te worden, de waarde van X wordt niet groter dan 6 (A(6)).
De RND functie is voor meer dan alleen maar een dobbelsteen te gebruiken. Er is daarom een algemene formule nodig voor het genereren van een getal tussen twee grenzen. Het laagste getal dat gegenereerd kan worden heeft de waarde 1 en het hoogste de waarde H (bij de dobbelsteen L = 1 en H = 6). Als G het ''geworpen'' getal is dan luidt de formule:

$$G = INT((1 + H - L) * RND(1)) + L$$

Vul voor de dobbelsteen een 6 voor H in en een 1 voor L en u krijgt:

$$G = INT((1 + 6 - 1) * RND(1)) + 1$$
$$G = INT(6 * RND(1)) + 1$$

wat in overeenstemming is met het voorgaande.

10.2. Twee toepassingen

We hebben nu al veel instructies en mogelijkheden van de computer leren kennen. Het wordt daarom eens tijd om een wat groter programma te proberen dat op zichzelf staat. Veel van de programma's die tot nu toe in dit boek zijn gegeven kunnen als

een deel van een groter programma worden gebruikt. U kunt er ook mee oefenen, ze veranderen of misschien verbeteren. Dat geldt trouwens voor elk programma in dit boek.

Het nu volgende programma laat u een getal raden. Het te raden getal bestaat uit vier cijfers en wordt door de computer gegenereerd. U wordt door de computer verzocht zelf een getal van vier cijfers in te toetsen. De computer geeft daarna met "plaats" aan hoeveel cijfers daarvan gelijk zijn aan die van het te raden getal *en* op de goede plaats staan. Met "goed" wordt aangegeven hoeveel van de andere cijfers ook in het getal voorkomen maar *niet* op de goede plaats staan.
Door steeds een ander getal van vier cijfers in te voeren moet u proberen het getal te raden. Lukt dat niet of bent u het zat, toets dan "help" in en de computer laat u zien welk getal u had moeten raden. Denk er om dat u woorden die de computer moet kunnen herkennen met de *juiste* karakters (hoofd- en/of kleine letters) moet invoeren.
In elk getal kunnen meerdere gelijke cijfers voorkomen. Er kan ook een 0 als eerste cijfer staan!
Door het grote aantal functies die betrekking hebben op stringvariabelen zijn met deze variabelen manipulaties mogelijk. Vandaar dat we de gekozen getallen als stringvariabelen invoeren.

```
10 REM rnd getal wordt toevalsgetal
20 CLS
30 PRINT "druk op de toets SELECT"
40 G$=INKEY$: A=RND(1)
50 IF G$<>CHR$(24) THEN 40
60 REM begin van het spel
70 WIDTH(40): CLS: T=0: G$=""
80 REM stel het te raden getal samen
90 FOR X=1 TO 4: C$=STR$(INT(10*RND(1)))
100 G$=G$+C$: NEXT X
110 PRINT "Raadt een getal van 4 cijfers"
120 REM voer het getal in
130 INPUT"welk getal";R$: PRINT
140 P=0: G=0: Y=0
150 IF R$="help" THEN 390
160 IF LEN(R$)<>4 THEN 440
170 REM tel het aantal pogingen T
180 T=T+1: FOR X=1 TO 4
190 REM neem elk cijfer apart
200 C$(X)=MID$(G$,2*X,1)
210 RC$(X)=MID$(R$,X,1)
220 REM bepaal of cijfers gelijk zijn
230 IF RC$(X)=C$(X) THEN 450
240 REM getal gevonden bij p=4
250 NEXT X: IF P=4 THEN 360
260 REM zoek naar goede cijfers
270 Y=Y+1: FOR X=1 TO 4
280 IF RC$(Y)=C$(X) THEN 470
290 NEXT X
300 REM vergelijk het volgende cijfer
310 IF Y<4 THEN 270
320 REM geef resultaat van de poging
330 PRINT "plaats=";P; TAB(33)"goed";G
340 REM voor de volgende poging
350 GOTO 120
360 REM geef het resultaat
370 PRINT "getal geraden in";T;
380 PRINT "pogingen": PRINT: GOTO 400
390 PRINT "het getal is";G$
400 INPUT"nog eens? ja of nee";J$
410 IF J$="ja" THEN 60
420 IF J$<>"nee" THEN 400
430 END
440 PRINT "4 cijfers slome": GOTO 120
450 REM tel het aantal juiste cijfers
460 P=P+1: C$(X)="": RC$(X)="A": GOTO 240
470 REM tel het aantal goede cijfers
480 G=G+1: C$(X)="": GOTO 300
```

Het enige nieuwe statement dat in dit programma voorkomt ziet u al in de eerste programmaregel, het REM statement. Dit statement heeft geen enkele werking in het programma en dient slechts om de gelegenheid te geven aanvullende aantekeningen te maken over het programma zodat het later eenvoudiger is de werking ervan te doorgronden. De regels 20 tot en met 50 brengen niets nieuws voor u. Dat hier opnieuw de toets SELECT is gekozen hoeft uiteraard niet te betekenen dat een andere toets niet hetzelfde werk zou kunnen doen.
Omdat van het te raden getal elk cijfer moet kunnen worden vergeleken met die van het ingevoerde getal moet elk cijfer apart in een stringvariabele worden geplaatst. De afzonderlijke cijfers worden door INT(10*RND(1)) gegenereerd. De functie STR$(X) maakt het mogelijk een getal uit een gewone variabele om te zetten tot een aantal karakters die in een stringvariabele kunnen worden geplaatst. Hier wordt elk cijfer in de stringvariabele C$ geplaatst en door regelmatig "optellen" komen alle cijfers in volgorde samen in de variabele G$. Ook het geraden getal wordt in een stringvariabele ingevoerd (R$). In de regels 180 tot en met 250 wordt nagegaan of twee overeenkomstige cijfers van het te raden getal en het geraden getal gelijk zijn.
Hiertoe worden de afzonderlijke cijfers elk eerst in een stringvariabele geplaatst (resp. RC$(X)). Zijn er gelijke cijfers, die ook nog dezelfde plaats hebben, dan wordt teller P opgehoogd. Als P = 4 dan is het getal geraden. De stringvariabelen die gelijk zijn worden "onschadelijk" gemaakt zodat ze niet meer meetellen (C$(X) = " ": RC$(X) = "A"). In de regels 260 tot en met 310 worden de overgeble-

ven variabelen vergeleken en bij gelijke cijfers de teller G opgehoogd. Eerst wordt één variabele van RC$ vergeleken met de vier variabelen van C$. Daarna wordt de volgende variabele van RC$ vergeleken enz. De teller voor X vormt een lus (4× voor vier variabelen) binnen die voor Y. Ook nu wordt er voor gezorgd dat een eenmaal getelde variabele niet opnieuw kan meetellen. De rest van het programma is eenvoudig te volgen. Probeer eens na te gaan hoe de computer weet hoeveel pogingen u nodig hebt gehad voor het vinden van het juiste antwoord.

Het volgende programma laat u een woord raden. Daartoe moet een ander persoon een aantal woorden invoeren (maximaal 10). De computer kiest één van die woorden en vertelt u uit hoeveel letters dat woord bestaat. U voert daarna een letter in de computer en de computer meldt hoe vaak deze letter in het woord voorkomt. U moet alle letters van het woord eerst raden. Daarna moet u ook nog het woord samenstellen. Dat kan ook nog wel eens fout gaan, denk maar aan klok en kolk. Komt u er niet meer uit type dan help.

```
10 'rnd getal wordt toevalsgetal
20 CLS
30 PRINT "druk op toets SELECT"
40 G$=INKEY$: A=RND(1)
50 IF G$<>CHR$(24) THEN 40
60 'begin van het spel
70 WIDTH(40): CLS
80 INPUT"hoeveel woorden";N
90 FOR X=1 TO N
100 PRINT "woord";X;:INPUT W$(X)
110 ON ERROR GOTO 1000: NEXT X: CLS
300 PRINT "woord raden"
310 PRINT "type bij nood help": PRINT
320 'bepaal het volgnummer (1 tot n)
330 T=T+1
340 'kies het woord
350 P(T)=INT(N*RND(1))+1: FOR Y=1 TO T
360 'bepaal of het woord al is gekozen
370 IF P(T)=P(Y) AND T<>Y THEN 350
380 NEXT Y: X=P(T): 'X=woordnummer
390 L=LEN(W$(X)):G$=""
400 PRINT "het woord heeft";L;"letters"
410 INPUT"welke letter";L$:K=0
420 IF L$="help" THEN 580
430 'bepaal of de letter in het woord is
440 FOR R=1 TO L
450 IF L$<>MID$(W$(X),R,1) THEN 480
460 'letter in het woord gevonden
470 K=K+1: G$=G$+L$: 'voeg letters samen
480 NEXT: PRINT K"keer "L$: PRINT
490 IF LEN(G$)=L THEN 520
500 ' voor de volgende letter
510 GOTO 410
520 PRINT "u hebt alle letters"
530 PRINT "uw letters zijn "G$
540 INPUT"welk woord";WW$
550 IF WW$="help" THEN 580
560 IF WW$=W$(X) THEN 590: 'goed geraden
570 PRINT "verkeerd woord": GOTO 540
580 PRINT "het woord is "W$(X): GOTO 610
590 PRINT "goed geraden"
600 'bepaal of alle woorden zijn geweest
610 IF T=N THEN 650
620 INPUT"nog eens, ja of nee";J$
630 IF J$="ja" THEN CLS: GOTO 330
640 IF J$<>"nee" THEN 620 ELSE 720
650 PRINT "u hebt alle woorden"
720 END
1000 PRINT "niet meer dan tien woorden"
1010 PRINT "het laatste woord vervalt"
1020 N=10: FOR X=0 TO 2000: NEXT X
1030 CLS: RESUME 300
```

In regel 10 vindt u een REM statement. Het woord REM is hier vervangen door het teken '. Dit is een verkorte wijze voor het invoeren van het REM statement. De regels 40 en 50 zorgen er voor dat de RND functie een echt toevalsgetal genereert. In de regels 80 tot en met 110 worden de woorden ingevoerd. U kunt niet meer dan 10 woorden invoeren omdat ze in de array met de stringvariabelen (W$(X)) worden geplaatst en deze variabelen zijn niet gedimensioneerd. Nu kunnen we een onderbreking van het programma door een foutmelding voorkomen (als geprobeerd wordt meer dan 10 woorden in te voeren) met het statement dat we zien in regel 110. Als N groter is dan 10 dan zal na het invoeren van het tiende woord een foutmelding optreden. Het statement ON ERROR GOTO voorkomt echter dat door de foutmelding het programma wordt onderbroken. In plaats daarvan wordt naar regel 1000 gesprongen. In deze en in de volgende regel wordt de fout gemeld. De vertraging in regel 1020 geeft ons nog even de tijd om dat te kunnen lezen. Daarna laat RESUME 120 het programma verder gaan met regel 300. Na het doorlopen van een foutmeldingsroutine (hier de regels 1000-1030) kan alleen een RESUME dienst doen als GOTO. Voer ON ERROR GOTO 1000 *niet* eerder in regel 110 in dan wanneer u zeker weet dat het programma verder geen fouten bevat. Zitten na regel 110 nog typefouten in het programma dan reageert ON ERROR daar ook op!
In regel 330 wordt een volgnummer aan elk woord gegeven. Daarna wordt in regel 350 een getal P(T) gegenereerd tussen 1 en N. Op de gebruikelijke wijze wordt nagegaan of dat getal al eens eerder aan de beurt is geweest. In regel 380 wordt aan X de waarde van P(T) gegeven zodat het woord W$(X) nu door het toeval is bepaald. Nu kunnen we een

letter invoeren. In een lus (regels 440 tot en met 480) wordt de gekozen letter vergeleken met de letters van het te raden woord. Is er een gelijke letter gevonden dan wordt de teller K opgehoogd en de ingevoerde letter in G$ geplaatst. Als de ingevoerde letter meerdere keren in het te raden woord voorkomt dan wordt K evenzoveel keren opgehoogd en de letter ook evenzoveel keren in G$ opgeslagen. Steeds als een letter is afgehandeld moet K nul worden voor de volgende letter (regel 410) en steeds als een woord is voltooid moet G$ ''leeg'' worden (regel 390). Of alle letters van het woord zijn geraden wordt nagegaan in regel 490, zo niet dan wordt om een nieuwe letter gevraagd (GOTO 410). In het andere geval wordt u in de gelegenheid gesteld het woord te raden. Het programma is verder eenvoudig te volgen.

Een andere manier om de te raden woorden in te voeren is die waarbij de woorden op de tape worden geplaatst. Hiertoe worden ze in een FILE naar de recorder verzonden. Hiervoor kan het volgende programma dienen.

```
10 'openen van de file
20 OPEN"cas:raden" FOR OUTPUT AS #1
100 'aantal woorden in de file
110 INPUT"hoeveel woorden"; N$
120 IF VAL(N$)<1 OR VAL(N$)>10 THEN 110
130 'invoeren van het aantal vd woorden
140 PRINT #1, N$: FOR Y=1 TO VAL(N$)
150 'invoeren van de woorden
160 PRINT"woord";Y;:INPUT B$
170 PRINT #1, B$: NEXT Y: CLOSE
```

Het OPEN statement in regel 20 is al bekend. Voor de file is de naam ''raden'' gekozen. Voor het ''woord raden'' programma is het nodig te weten hoeveel woorden (N) op de tape staan. Daarom wordt dit gegeven in N$ naar de band geschreven (regels 110 tot en met 140). Het aantal woorden mag weer niet groter zijn dan 10. In regel 120 ziet u hoe een foutmelding is voorkomen. In regel 160 wordt verzocht het woord in te toetsen (woord 1?, woord 2? enz.) tot het laatste woord (FOR Y = 1 TO VAL(N$)). In regel 170 worden deze woorden naar de band geschreven. De eerste string die de file bevat is dus het gegeven over het aantal woorden, de volgende strings zijn de woorden zelf. RUN het programma en laat een ander een aantal woorden invoeren.
Nu moeten we het programma ''woord raden'' nog zo veranderen dat in dit programma de file van de band wordt gelezen. De regelnummers van de programma's zijn overigens zo gekozen dat de latere uitbreidingen en veranderingen er zonder meer in passen.
Eerst gaan we enige regels uit het programma ''woord raden'' verwijderen. Dat zijn de regels 80 tot en met 110 en 1000 tot en met 1030. Doe dat zo:

delete 80 – 110
delete 1000 – 1030

Het commando delete is er speciaal voor om regels uit een programma te verwijderen. Dat is makkelijk als het een wat groter aantal regels betreft. Hiervoor moet u natuurlijk eerst het programma in de computer laden. Nu voeren we de volgende regels in het programma:

```
180 'haal de file van de tape
190 OPEN"cas:raden" FOR INPUT AS #1
200 'inlezen van het aantal vd woorden
210 INPUT #1, C$: N=VAL(C$)
220 'inlezen van de woorden
230 FOR D=1 TO N
240 INPUT #1, W$(D): NEXT D: CLOSE
```

Dit programmadeel is eenvoudig. De eerste string met het getal van het aantal woorden wordt in regel 210 gelezen. In de volgorde van binnenkomst worden daarna in de variabelen W$(1), W$(2) enz. de woorden geplaatst die na elkaar van de band worden gelezen (regel 240). In dit programma komt uiteraard geen PRINT statement voor die de woorden op het scherm laat zien.

Nu is het raden van tien woorden in een spelletje meestal wel genoeg. Als het echter steeds dezelfde woorden zijn die van de band worden gelezen dan weet u ze gauw genoeg uit het hoofd. Beter is het als er een aantal FILES op de band staan waaruit de computer er steeds willekeurig één kiest. Dit ''kiezen'' moet dan gebeuren met de file naam die we daarom eenvoudig ''1'', ''2'', ''3'' enz. noemen, afhankelijk van het aantal FILES. Met het volgende programma kunnen we ten hoogste tien FILES naar de band schrijven.

```
10 'invoeren van het aantal vd files
20 OPEN"cas:raden" FOR OUTPUT AS #1
30 INPUT"aantal files";A$
40 A=VAL(A$): IF A<1 OR A>10 THEN 30
50 PRINT #1, A$
60 CLOSE
70 FOR X=1 TO A
```

```
80 'open de file met naam X
90 OPEN"cas:"+STR$(X) FOR OUTPUT AS #1
100 'aantal woorden in de file
110 INPUT"hoeveel woorden"; N$
120 IF VAL(N$)<1 OR VAL(N$)>10 THEN 110
130 'invoeren van het aantal vd woorden
140 PRINT #1, N$: FOR Y=1 TO VAL(N$)
150 'invoeren van de woorden
160 PRINT"woord";Y;:INPUT B$
170 PRINT #1, B$: NEXT Y: CLOSE
180 'voor de volgende file
190 NEXT X
```

Dit programma bevat een aantal regels uit het vorige programma voor het schrijven van woorden naar de tape.
Eerst moet de computer weten uit hoeveel FILES hij kan kiezen. Dit getal moet u eerst invoeren (regel 30) waarna het in regel 50 naar de tape wordt geschreven. De regels 70 tot en met 170 worden gebruikt om bij herhaling steeds één FILE naar de tape te schrijven. Steeds wordt een FILE geopend met de naam die wordt gevormd uit de inhoud van de numerieke variabele X. Let op hoe in regel 90 deze waarde wordt omgezet in een string voor de naam van de FILE. Elke FILE wordt steeds weer gesloten met CLOSE in regel 170. Tussen deze regels worden voor elke FILE afzonderlijk de woorden ingevoerd met de regels die we al van het vorige programma kennen.
Ook het programma ''woord raden'' moet nu weer worden uitgebreid.

```
80 'inlezen van het aantal vd files
90 OPEN"cas:raden" FOR INPUT AS #1
100 INPUT #1, A$: CLOSE: A=VAL(A$)
110 F=F+1: 'F is het volgnr. vd file
120 'kies de file
130 B(F)=INT(A*RND(1))+1: T=0
140 'bepaal of de file al is gekozen
150 FOR C=1 TO F
160 IF B(F)=B(C) AND F<>C THEN 130
170 NEXT C
180 'haal de file van de tape
190 OPEN"cas:"+STR$(B(F))FOR INPUT AS #1
200 'inlezen van het aantal vd woorden
210 INPUT #1, C$: N=VAL(C$)
220 'inlezen van de woorden
230 FOR D=1 TO N
240 INPUT #1, W$(D): NEXT D: CLOSE
250 CLS: PRINT "spoel tape terug"
260 FOR D=0 TO 5: CL=10
270 COLOR CL: FOR C=0 TO 500: NEXT C
280 IF CL=10 THEN CL=15: GOTO 270
290 NEXT D: CLS
660 IF F=A THEN 710
670 PRINT "wilt u nog een serie"
680 INPUT"ja of nee";J$
690 IF J$="ja" THEN 110
700 IF J$<>"nee" THEN 670 ELSE 720
710 PRINT "u hebt alle series"
720 END
```

Hiervan hebt u de regels 180 tot en met 240 al eerder ingevoerd.
Alleen dient u regel 190 te veranderen. De naam van het OPEN statement moet namelijk steeds gelijk zijn aan het nummer van de gekozen FILE (B(F)). Het kiezen van de FILE gebeurt in regel 130 waarna wordt nagegaan of het nummer van de FILE al eerder is gekozen. Eerst moet echter het getal van de tape worden gehaald waaruit volgt uit hoeveel FILES moet worden gekozen (regels 80 en 90).
Aan het eind van het programma dienen ook een paar regels te worden toegevoegd. Behalve dat u moet weten wanneer u alle woorden uit een file hebt geraden moet u ook weten of alle files aan de beurt zijn geweest. Dat wordt nagegaan in regel 660 waarin F het aantal FILES is dat u hebt afgewerkt en A het aantal FILES dat op de band staat (zie regel 100). Hebt u nog niet alle FILES (series) gehad dan kunt u met ja of nee laten weten of u al of niet nog een serie woorden wilt raden. De tekst in regel 710 zal goed beschouwd alleen door de doorzetters op het scherm worden gezien. Omdat de FILES in een willekeurige volgorde van de tape worden gelezen en niet in de volgorde zoals ze op de tape staan is het nodig dat steeds nadat een file is gelezen de band wordt teruggespoeld. Hiervoor wordt u gewaarschuwd in de regels 250 tot en met 280. Hierbij wisselt de tekst op het scherm een aantal keren van kleur door a = 10 (geel) en a = 15 (wit).
Het is al eens eerder vermeld: als u na de nodige veranderingen wilt weten hoe een programma er uit ziet, LIST het dan eens. Als u dat met het volledige programma ''woord raden'' doet dan zult u zien dat het veel te lang is en daarom slechts in gedeelten kan worden geschreven.
Om toch een goed overzicht te hebben volgt hier het volledige programma. Zo ziet u dat een printer toch wel een nuttig apparaat kan zijn.

```
10 'rnd getal wordt toevalsgetal
20 CLS
30 PRINT "druk op toets SELECT"
40 G$=INKEY$: A=RND(1)
50 IF G$<>CHR$(24) THEN 40
60 'begin van het spel
70 WIDTH(40): CLS
80 'inlezen van het aantal vd files
90 OPEN"cas:raden" FOR INPUT AS #1
100 INPUT #1, A$: CLOSE: A=VAL(A$)
110 F=F+1: 'F is het volgnr. vd file
120 'kies de file
130 B(F)=INT(A*RND(1))+1: T=0
```

```
140 'bepaal of de file als is gekozen
150 FOR C=1 TO F
160 IF B(F)=B(C) AND F<>C THEN 130
170 NEXT C
180 'haal de file van de tape
190 OPEN"cas:"+STR$(B(F))FOR INPUT AS #1
200 'inlezen van het aantal vd woorden
210 INPUT #1, C$: N=VAL(C$)
220 'inlezen van de woorden
230 FOR D=1 TO N
240 INPUT #1, W$(D): NEXT D: CLOSE
250 CLS: PRINT "spoel tape terug"
260 FOR D=0 TO 5: CL=10
270 COLOR CL: FOR C=0 TO 500: NEXT C
280 IF CL=10 THEN CL=15: GOTO 270
290 NEXT D: CLS
300 PRINT "woord raden"
310 PRINT "type bij nood help": PRINT
320 'bepaal het volgnummer (1 tot n)
330 T=T+1
340 'kies het woord
350 P(T)=INT(N*RND(1))+1: FOR Y=1 TO T
360 'bepaal of het woord al is gekozen
370 IF P(T)=P(Y) AND T<>Y THEN 350
380 NEXT Y: X=P(T): 'X=woordnummer
390 L=LEN(W$(X)):G$=""
400 PRINT "het woord heeft";L;"letters"
410 INPUT"welke letter";L$:K=0
420 IF L$="help" THEN 580
430 'bepaal of de letter in het woord is
440 FOR R=1 TO L
450 IF L$<>MID$(W$(X),R,1) THEN 480
460 'letter in het woord gevonden
470 K=K+1: G$=G$+L$: 'voeg letters samen
480 NEXT: PRINT K"keer "L$: PRINT
490 IF LEN(G$)=L THEN 520
500 ' voor de volgende letter
510 GOTO 410
520 PRINT "u hebt alle letters"
530 PRINT "uw letters zijn "G$
540 INPUT"welk woord";WW$
550 IF WW$="help" THEN 580
560 IF WW$=W$(X) THEN 590: 'goed geraden
570 PRINT "verkeerd woord": GOTO 540
580 PRINT "het woord is "W$(X): GOTO 610
590 PRINT "goed geraden"
600 'bepaal of alle woorden zijn geweest
610 IF T=N THEN 650
620 INPUT"nog eens, ja of nee";J$
630 IF J$="ja" THEN CLS: GOTO 330
640 IF J$<>"nee" THEN 620 ELSE 720
650 PRINT "u hebt alle woorden"
660 IF F=A THEN 710
670 PRINT "wilt u nog een serie"
680 INPUT"ja of nee";J$
690 IF J$="ja" THEN 110
700 IF J$<>"nee" THEN 670 ELSE 720
710 PRINT "u hebt alle series"
720 END
```

10.3. Het voorkomen van foutmeldingen

We hebben al een paar keer kunnen zien hoe kan worden voorkomen dat door een foutmelding een programma wordt onderbroken. Uiteraard kan dat alleen maar als van te voren duidelijk is dat een fout kan optreden, zoals het invoeren van een te groot aantal gegevens. In dit hoofdstuk betrof dat het invoeren van een te groot aantal woorden. Een enkele programmaregel is dan voldoende om te voorkomen dat een dergelijke fout gemaakt kan worden, zoiets als IF N>10 THEN . . .
In principe beschermen we het programma hiermee voor fouten die voorkomen kunnen worden. Anders wordt het als fouten niet voorkomen kunnen worden. Dit komt bijvoorbeeld voor als van een variabele in een programma zowel de vierkantswortel als de inverse waarde (1/X) moet worden berekend, terwijl blijkt dat deze variabele een negatieve waarde kan hebben, nul kan worden of een positieve waarde kan bezitten. Nu kan van een negatief getal geen wortel worden getrokken terwijl delen door nul niet is toegestaan. In het volgende programma is te zien hoe dat kan worden opgelost.

```
10 CLS: C$="nvt": PRINT "X=";
20 PRINT TAB(8)"SQR(X)=";
30 PRINT TAB(18)"1/X=": PRINT
40 FOR X=-9 TO 9
50 ON ERROR GOTO 110
60 A$=STR$(CSNG(SQR(X)))
70 B$=STR$(CSNG(1/X))
80 PRINT X; TAB(8)A$; TAB(18)B$
90 NEXT X
100 GOTO 100
110 IF ERR=5 AND ERL=60 THEN A$=C$
120 IF ERR=11 AND ERL=70 THEN B$=C$
130 RESUME NEXT
```

In dit programma laten we X veranderen van −9 tot en met +9 (regel 40). In regel 50 zien we het statement ON ERROR GOTO. In regel 60 wordt de vierkantswortel berekend. In dit geval willen we de verkregen waarde in een stringvariabele hebben om hierin bij een fout ''nvt'' te kunnen plaatsen. Hiertoe dient de functie STR$(X). Met de functie CSNG(X) wordt de waarde met de lage nauwkeurigheid berekend. In regel 30 wordt op gelijke wijze de berekening 1/X uitgevoerd. De berekende waarden worden met een TAB(N) functie in een tabel geplaatst waarvoor in de regels 10, 20 en 30 de kop is gemaakt.
Nu kunnen er twee soorten foutmeldingen optreden, bij een negatieve waarde van X ''illegal function call'' met nummer 5 in regel 60 en als X=0 ''division by zero'' met nummer 11 in regel 70. Na het optreden van een fout vindt u het nummer van de fout in de variabele ERR. Probeer maar:

print 1/0

en nu

print err

Treed de fout op in een programmaregel dan heeft de variabele ERL het regelnummer als inhoud. Daardoor kan in de regels 110 en 120 worden bepaald welke variabele met "nvt" moet worden gevuld. Met RESUME NEXT wordt vanuit de foutmeldingsroutine teruggegaan naar de instructie die volgt op die waarin de fout was opgetreden. Treedt de fout op in regel 60 dan wordt in dit geval dus het programma vervolgd met regel 70. Dat na RESUME ook een regelnummer kan volgen hebt u al gezien. Alleen RESUME kan ook. Dan wordt teruggegaan naar de instructie waarin de fout optrad.

10.4. De iteratiemethode

U kent het verschijnsel: als u iemand een bal moet toewerpen en de persoon staat stil dan lukt het nog wel om de bal op de goede plaats te krijgen. Loopt de persoon dan wordt het lastig. Dan moet de bal worden gegooid naar een plaats waar nog niemand is. Deze plaats is afhankelijk van de snelheid waarmee de persoon loopt en van de tijd die de bal nodig heeft om de afstand af te leggen die hij moet gaan. Dit laatste is een onoplosbaar probleem. De afstand die de bal moet afleggen is nog niet bekend! De computer kan deze berekening aan, mits het juiste programma wordt ingevoerd. In dat programma wordt de afstand gewoon "gegokt". Met deze gegokte afstand wordt berekend waar de persoon heen loopt. De computer berekent nu hoeveel hij heeft misgegokt, corrigeert de gegokte afstand, berekent opnieuw enz. Na een aantal berekeningen is de berekende afstand dan zo dat kan worden gezegd dat de worp "raak" is. Deze rekenmethode is de "iteratiemethode" en kan bij schijnbaar onoplosbare problemen wel eens uitkomst geven. We zullen deze methode met een eenvoudig voorbeeld toelichten. Veronderstel dat we de vierkantswortel uit 10 willen berekenen (volgens de computer met SQR(10) = 3.1622776601684).
We weten dat als X de vierkantswortel uit 10 is dat X^2 = 10 en 10/X = X. Nu gokken we gewoon voor X een getal en voeren de laatstgenoemde deling uit. Stel we gokken voor X het getal 2.
10/2 = 5 (1e berekening)
Zolang het getal achter het =teken, het quotiënt, nog niet in de verste verte lijkt op de deler (2) is de wortel uit 10 nog niet gevonden. We nemen nu het gemiddelde van 2 en 5.
(2 + 5)/2 = 3.5
De uitkomst hiervan gebruiken we als deler:
10/3.5 = 2.86 (2e berekening).
Ook nu is het juiste getal nog lang niet gevonden, maar de getallen komen wel dichter bij elkaar. We geven de moed daarom nog niet op en gaan weer middelen.
(3.5 + 2.86)/2 = 3.18
10/3.18 = 3.14 (3e berekening).
Dit begint er al een beetje op te lijken en we gaan door totdat de afwijking tussen de getallen zo klein is dat de wortel met voldoende nauwkeurigheid is gevonden. Dit laatste is belangrijk. We kunnen wel aan de gang blijven met berekenen, maar eens moet er een eind aan komen. We moeten daarom een graad van nauwkeurigheid vaststellen en als die is bereikt de berekening stoppen. Met het volgende programma laten we de computer al dat werk doen.

```
10 CLS
20 INPUT"geef een getal";G:PRINT
30 W1=G*RND(1)
40 W2=G/W1
50 IF ABS(W2-W1)<W1*1E-13THEN70
60 W1=(W1+W2)/2:GOTO 40
70 PRINT "de wortel uit";G;"is";W1
```

Zoals u ziet wordt in regel 30 inderdaad een getal voor de wortel gegokt (W1). De regels 30 en 40 komen overeen met ons rekenvoorbeeld.
Regel 50 bepaalt of de nauwkeurigheid van de berekening groot genoeg is. De nauwkeurigheid die hier wordt gevraagd is op zijn minst even groot als die waarmee de computer de vierkantswortel berekent. Het verschil W2 – W1 kan wel eens negatief uitvallen. Een dergelijke waarde is steeds kleiner dan welke pos. getal dan ook, vandaar dat de functie ABS(W2 – W1) is toegepast. Deze functie geeft de waarde van een getal zonder teken, de absolute waarde. De werking van de functie wordt met de volgende regel gedemonstreerd:

print abs(+25.15); abs(–25.15); abs(0)

Run het programma en voer het getal 10 in. Probeer daarna in de direct mode:

print sqr(10)

Probeer ook eens een paar andere getallen.

Het volgende programma geeft u de hoek in radialen als u de sinus van de hoek invoert (alleen in het eerste kwadrant).

```
10 CLS
20 INPUT"HOE GROOT IS SIN(X)";Y
30 PRINT
40 H1=ATN(1)*2*RND(1)
50 H2=SIN(H1): X=Y-H2
60 IF ABS(X)<1E-13 THEN 80
70 H1=H1+X: GOTO 50
80 PRINT "de hoek is";H1;"RAD"
```

In regel 40 wordt de grootte van de hoek gegokt. Omdat de hoek niet groter kan zijn van pi/2 rad. (90 graden) wordt deze waarde als factor genomen. Met ATN(1)*2 wordt pi/2 berekend. Met de gegokte hoek H1 wordt H2 = SIN(H1) berekend. Stel H1 is te groot, dan is ook de berekende waarde voor H2 te groot. Nu is X in regel 50 negatief. De nieuwe waarde voor H1 is daardoor kleiner (regel 70). De berekening herhaalt zich totdat de gewenste nauwkeurigheid is bereikt.
RUN het programma en voer voor SIN(X) de waarde 0.5 in. Probeer daarna in de direct mode:

y = .5:x = sqr(1 – y^2): print atn(y/x)

11. De geluidsgenerator

11.1. Inleiding

U weet dat alle gegevens in de computer zijn opgeslagen in een geheugenplaats in de vorm van binaire getallen van acht bits. Verder werkt zo'n computer elektronisch, dat wil zeggen dat elk bit is te herkennen aan het al of niet aanwezig zijn van een spanning in zo'n geheugenplaats. Geen spanning is 0, wel spanning is 1. Door in een bepaald ritme een geheugenplaats in te schrijven met 1 en 0 wordt de plaats voor het laagstwaardige bit in hetzelfde ritme spanningvoerend en spanningloos, alsof er snel na elkaar een schakelaar wordt geopend en gesloten. In afb. 8 zien we hoe de spanning in zo'n geheugenplaats grafisch wordt weergegeven. Naar de vorm wordt het een blokspanning genoemd.



afb. 8

Nu kan op een bepaalde manier de spanning van zo'n geheugenplaats aan de ingang van een versterker worden gelegd waardoor als het ritme van de wisseling maar snel genoeg is, een toon te horen is. De toonhoogte is daarbij afhankelijk van het aantal wisselingen per seconde (aantal trillingen per seconde), de frequentie. Door de frequentie te regelen hebben we de toonhoogte geheel in de hand.
Ook de MSX computer kan op deze manier geluid maken. Wat dat geluid betreft is het ons echter makkelijk gemaakt omdat in de computer een speciaal onderdeel is aangebracht, de Programmable Sound Generator (PSG), waarmee diverse geluiden veroorzaakt kunnen worden. Dit geluid is dan hoorbaar via de luidspreker van het gebruikte televisietoestel of via een versterker die op de audioplug van de computer is aangesloten. De PSG kan drie stemmen gelijktijdig en onafhankelijk van elkaar laten klinken.
Nu is het voor ons niet mogelijk om alle trillingen te kunnen horen. De bovengrens ligt, afhankelijk van o.a. de leeftijd van de luisteraar, bij 12000 tot 20000 trillingen per seconde (uitgedrukt in Herz, Hz).
In het hoorbare gebied kunnen we elke trilling van voldoende sterkte waarnemen, van welke frequentie dan ook. Dat betekent dat wanneer we een trilling "vloeiend" van frequentie veranderen van laag naar hoog, we een geluid horen waarvan de toonhoogte ook vloeiend hoger wordt. Nu kunnen de meeste muziekinstrumenten slechts trillingen van een bepaalde toonhoogte veroorzaken en men heeft de frequenties van deze tonen gestandaardiseerd. Daarbij is het gehele gebied verdeeld in blokken, de "octaven". Deze verdeling is zo dat een bepaalde toon in een octaaf precies twee maal zo hoog van frequentie is dan de overeenkomstige toon van de voorgaande octaaf. Elke octaaf is weer verdeeld in twaalf stappen, de noten. Deze worden met de volgende letters aangegeven:

C, C+, D, D+, E, F, F+, G, G+, A, A+, B.

Van elke noot in deze reeks is de toonhoogte een halve "stap" hoger dan die van zijn voorganger. De noten C+, D+ enz. worden daarom wel de "halve" tonen genoemd. De notatie "+" is hier gebruikt omdat op deze manier deze halve tonen in de computer kunnen worden gevoerd. De tonen C, D, E, F, G, A en B kunt u overeen laten komen met resp. do, re, mi, fa, sol, la en si van onze bekende toonladder. Om te kunnen weten op welke manier we de computer geluid kunnen laten maken is het nodig dat we de frequentie van de diverse noten kennen. Samen met nog andere gegevens vinden we die in lijst 7.

Behalve de toonhoogte kan in bepaalde gevallen ook het volume van het geluid worden geregeld. Deze twee bepalen echter niet alleen het kenmerk van een bepaald geluid. Dit wordt mede bepaald door de klankkleur en door de zogenaamde "omhullende". Met de omhullende wordt bedoeld of een toon zeer snel "aanzwelt", zoals bij het tokke-

len op snaarinstrumenten, of slechts langzaam aangroeit, zoals bij het blazen op een fluit. Ook valt daaronder het meer of minder snel afsterven van een toon na het beëindigen daarvan. Wat de klankkleur betreft, deze wordt ingesteld door de computer en daar kunnen we dus niets aan veranderen. Met de omhullende hebben we nog al wat mogelijkheden om te variëren. Al met al valt er nog wel het een en ander in te stellen.
Hiervoor zijn twee instructies, PLAY en SOUND. We houden ons om te beginnen met het statement PLAY bezig. Hierbij moeten de volgende gegevens worden ingevoerd:

a. De toonhoogte kan op twee manieren worden ingesteld. Ten eerste kan het octaaf en de noot binnen het octaaf (A, B enz) worden gekozen. Ten tweede kan de toonhoogte met een nummer worden ingevoerd (n in lijst 7).
b. Ingesteld moet worden hoe lang we een toon willen laten klinken.
c. De lengte van de noten ten opzichte van elkaar (halve noten, kwart noten enz.) moet worden ingesteld.
d. Hoelang de langste rust duurt is gelijk met b ingesteld. De lengte van de rusten ten opzichte van elkaar moet overeenkomstig met c worden ingevoerd.
e. De golfvorm moet worden gekozen en als er sprake is van aanzwellen en afsterven dan moet de tijdsduur daarvan worden ingesteld.
f. Uiteindelijk moet ook het volume worden ingevoerd.

Op al deze punten zal afzonderlijk worden ingegaan.
U hebt overigens al het geluid van uw computer kunnen waarnemen. Als er iets mis gaat komt er een korte toon uit de luidspreker van uw tv. Die kunt u ook als volgt oproepen:

Beep

Noot/ octaaf	n	freq. Hz.	HF	LF	ruis get.
C1	—	33.6	13	101	—
C+1	1	34.6	12	157	—
D1	2	36.7	11	235	—
D+1	3	38.8	11	66	—
E1	4	41.2	10	154	—
F1	5	43.6	10	4	—
F+1	6	46.3	9	113	—
G1	7	49.1	8	232	—
G+1	8	52	8	107	—
A1	9	55	7	242	—
A+1	10	58.3	7	127	—
B1	11	61.6	7	25	—
C2	12	65.2	6	179	—
C+2	13	69.3	6	78	—
D2	14	73.3	5	245	—
D+2	15	77.6	5	161	—
E2	16	82.4	5	77	—
F2	17	87.2	5	2	—
F+2	18	92.6	4	185	—
G2	19	98.1	4	116	—
G+2	20	104	4	52	—
A2	21	110	3	249	—
A+2	22	117	3	192	—
B2	23	123	3	140	—
C3	24	131	3	89	—
C+3	25	139	3	39	—
D3	26	147	2	251	—
D+3	27	155	2	209	—
E3	28	165	2	167	—
F3	29	175	2	129	—
F+3	30	185	2	92	—
G3	31	196	2	58	—
G+3	32	208	2	26	—
A3	33	220	1	253	—
A+3	34	233	1	224	—
B3	35	246	1	198	—
C4	36	261	1	173	—
C+4	37	277	1	148	—
D4	38	293	1	125	—
D+4	39	311	1	104	—
E4	40	330	1	83	—
F4	41	349	1	65	—
F+4	42	370	1	46	—
G4	43	393	1	29	—
G+4	44	416	1	13	—
A4	45	440	0	254	—
A+4	46	466	0	240	—
B4	47	493	0	227	—
C5	48	522	0	214	—
C+5	49	554	0	202	—
D5	50	587	0	191	—
D+5	51	621	0	180	—
E5	52	660	0	170	—
F5	53	698	0	160	—

Noot/ octaaf	n	freq. Hz.	HF	LF	ruis get.
F+5	54	740	0	151	—
G5	55	785	0	143	—
G+5	56	831	0	135	—
A5	57	880	0	127	—
A+5	58	933	0	120	—
B5	59	985	0	114	—
C6	60	1044	0	107	—
C+6	61	1109	0	101	—
D6	62	1173	0	95	—
D+6	63	1242	0	90	—
E6	64	1319	0	85	—
F6	65	1396	0	80	—
F+6	66	1481	0	76	—
G6	67	1570	0	71	—
G+6	68	1663	0	67	—
A6	69	1760	0	64	—
A+6	70	1865	0	60	—
B6	71	1970	0	57	—
C7	72	2088	0	54	—
C+7	73	2217	0	51	—
D7	74	2347	0	48	—
D+7	75	2484	0	45	—
E7	76	2638	0	42	—
F7	77	2792	0	40	—
F+7	78	3962	0	38	—
G7	79	3140	0	36	—
G+7	80	3326	0	34	—
A7	81	3520	0	32	—
A+7	82	3730	0	30	30
B7	83	3941	0	28	28
C8	84	4175	0	27	27
C+8	85	4434	0	25	25
D8	86	4693	0	24	24
D+8	87	4969	0	23	23
E8	88	5276	0	21	21
F8	89	5583	0	20	20
F+8	90	5923	0	19	19
G8	91	6279	0	18	18
G+8	92	6652	0	17	17
A8	93	7040	0	16	16
A+8	94	7461	0	15	15
B8	95	7882	0	14	14
	96	8351	0	13	13

Lijst 7

11.2. Het invoeren van de gegevens met het PLAY statement

De gegevens voor het laten spelen van een melodie met het statement PLAY dienen in een string achter dit statement te worden ingevuld, dus PLAY" " of PLAY A$. Voer het volgende eens in:

```
play"cde"
```

Als de volumeregelaar van uw tv nu maar hoog genoeg staat dan hoort u de tonen C, D en E uit het vierde octaaf. Mogelijk zult u zich afvragen hoe dat kan als in de vorige paragraaf is gesteld dat er toch wel het een en ander moet worden ingevoerd. Dat komt omdat deze gegevens al vooraf op een bepaalde waarde zijn ingesteld door het "START" programma van de computer na het inschakelen van de netspanning. In de beginstituatie is dus het octaaf 4 ingeschakeld. Deze ingestelde waarden kunt u veranderen in de string achter PLAY. De nieuwe waarden blijven daarna geldig, tot u de toetsen CTRL en STOP hebt gebruikt. Dan zijn de vooraf ingestelde waarden weer terug. Dat betekent ook dat u de gegevens die betrekking hebben op de tonen die u wilt laten spelen, slechts hoeft in te voeren als ze moeten veranderen.

Voer de volgende regel eens in:

```
a$=o5c": play"o4cdefab"+a$
```

U krijgt nu de volledige toonladder te horen. U ziet hier meteen de twee manieren voor het plaatsen van de gegevens na PLAY in een string.
Het volgende had hetzelfde effect gegeven:

```
play"o4cdefabo5c"
```

Probeer nu opnieuw

```
play"cde"
```

U zult horen dat de tonen nu in de vijfde octaaf worden gespeeld. Dit is dan ook de laatst ingestelde octaaf. In het voorbeeld komen nog geen halve tonen voor. Indien nodig kunnen ze worden ingevoerd door "+", "#" of "-". achter de noot te plaatsen. Met A+ wordt dezelfde noot bedoeld als met B-.

De tweede manier voor het invoeren van de tonen is die met het getal N:

```
10 FOR P=36 TO 40 STEP 2
20 PLAY"n=p;": NEXT P
30 FOR N=41 TO 47 STEP 2
40 PLAY"n=n;": NEXT N
50 PLAY"n48"
```

Opnieuw maakt u kennis met het op twee manieren invoeren van de tonen, ten eerste met "n=p;" (let op het teken ";") voor N36, N38 en N40 en ten tweede met "n48", dus *zonder* het "="teken. U kunt elke variabelenaam gebruiken, ook N ("n=n;" voor N41, N43, N45 en N47). Ook nu wordt de complete toonladder ten gehore gebracht.

Behalve het octaaf is in de beginsituatie ook de lengte van de tonen ingesteld. De tijdsduur is zodanig dat 30 volle noten aaneen precies in een minuut passen. De noten die u gehoord hebt zijn echter geen volle noten, maar kwart noten (ook vooraf ingesteld). Het "tempo" is daarom 30*4 = 120 tellen per minuut en dat wordt ingesteld met T120. Om dit tempo te veranderen dient u in de string, achter PLAY, Tn op te nemen, waarin n het aantal tellen van kwart noten is en een waarde kan hebben van 32 tot en met 255. Is n bekend dan kan de tijdsduur van een volle noot berekend worden met $P_t = 240/n$, waarin P_t de tijd in seconden is. De duur van een kwart noot is dan $P_t/4$. Dit is belangrijk bij het werken met de omhullende.

De lengte van de noten is in te stellen met Ln. Hierin is n steeds een macht van 2 (1, 2, 4, 8 enz.). Na L1 worden volle noten voortgebracht, na L2 halve noten, na L4 kwart noten (de begininstelling) enz. Voor elke noot afzonderlijk kan de lengte worden aangegeven (in afwijking van de met Ln ingevoerde waarde die voor de volgende noten van kracht blijft) door op gelijke wijze een cijfer achter de noot te plaatsen: a2 (halve noot), a16 (zestiende noot) enz.
In de muziek komen noten voor die met een factor 1.5 moeten worden verlengd. Deze worden aangegeven met een punt achter de noot. Op gelijke wijze wordt dat bij de MSX computer uitgevoerd. De noot "a+." duurt dus 1.5 keer zo lang als "a+".

Probeer eens:

a$ = "t90 L8 o5 e c o4 g o5 c. d16 c"

Deze regel bevat alles wat tot nu toe over het invoeren van muzieknoten is gesteld. In principe kunnen alle spaties worden weggelaten. Ze zijn hier echter aangebracht om de afzonderlijke opdrachten te kunnen onderscheiden. Het tempo van de muziek is 90 tellen. Een volle noot duurt 240/90 = 2.67 seconden. Er worden achtste noten gespeeld (L8) en die duren dus 1/3 sec. Eerst worden de noten e en c van het vijfde octaaf gespeeld (o5) en daarna g van het vierde octaaf. Dan keren we weer terug naar het vijfde octaaf voor de noot c die verlengd is en daardoor de lengte heeft van drie zestiende noten. Hierna volgt een zestiende noot (c16) en als laatste klinkt de c.
Beluister het resultaat met

play a$

We zullen deze stringvariabele nog vaker nodig hebben.

De lengte van de rusten wordt op dezelfde manier ingesteld als de lengte van de noten. Ze worden ingevoerd met Rn waarin n weer een macht van 2 is. Bij gelijkblijvend tempo zal R16 een rust van gelijke lengte geven als de noten A16, B+16 enz.

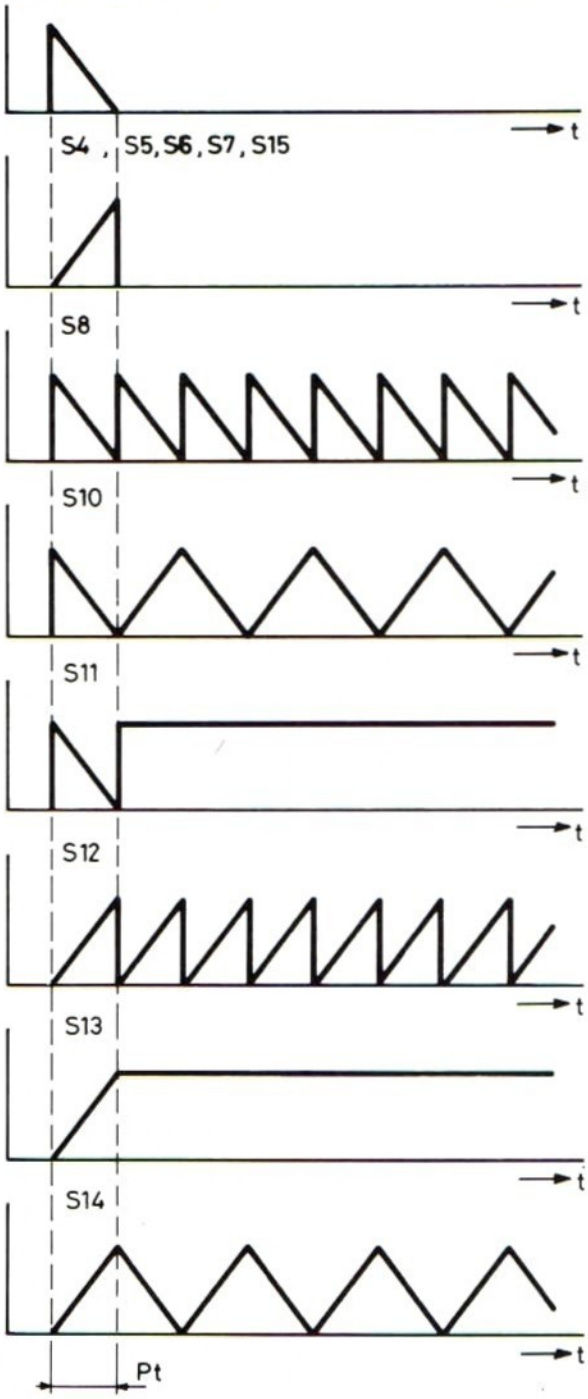
b$ = "t90 L8 o3 r8 g g r8 g g": play b$

Ook de stringvariabele b$ zullen we later nog gebruiken. Uiteraard zal alleen de tweede rust u goed opvallen. In deze string heeft L8 alleen betrekking op de noten. De lengte van de rusten moet voor elke rust apart worden ingesteld. Stelt u de noten in met het getal n dan kan een rust met n0 worden ingevoerd.

Bij het laatste voorbeeld: PLAY B$ hebt u de twee noten G voor en na de tweede rust niet gescheiden van elkaar kunnen horen. Dit zullen we met de omhullende moeten bewerkstelligen. De diverse mogelijkheden ziet u in afb. 9. Een bepaalde vorm kan gekozen worden door de opdracht Sn in de string achter PLAY te plaatsen waarbij n het nummer is dat bij de betreffende vorm is vermeld. Het volgende programma laat u het effect van diverse patonen horen.

```
10 CLS: PRINT"druk op een toets"
20 FOR N=0 TO 8 STEP 4
30 A$=INKEY$: IF A$="" THEN 30
40 GOSUB 80: NEXT N
```

```
50 FOR N=10 TO 14
60 A$=INKEY$: IF A$="" THEN 60
70 GOSUB 80: NEXT N: END
80 BEEP: PRINT"S";N
90 PLAY"t3212o4m3496s"+STR$(N)+"a"
100 RETURN
```



afb. 9

Steeds als u op een willekeurige toets gedrukt hebt hoort u het effect van een omhullende. Welke omhullende dat is wordt op het scherm aangegeven, voorafgegaan door een BEEP. De regels 20, 30 en 40 zijn voor S0, S4 en S8 (regel 20: FOR n = 0 TO 8 STEP 4). In regel 30 wordt bij elke doorgang gewacht tot een toets is ingedrukt. Daarna wordt in regel 40 naar de subroutine gesprongen voor de BEEP, het printen van Sn en het laten horen van de toon. Het invoeren van N geschiedt hier met STR$(N). Na de derde doorgang herhaalt zich het geheel voor S10 tot en met S14 in de regels 50 tot en met 70. Ga na wat er gebeurt als END in regel 70 wordt weggelaten. U ziet dat in de string na PLAY (let op: t32 L2 enz.) in regel 90 ook het gegeven m3496 is ingevoerd. Dat heeft te maken met de periodetijd die in afb. 9 met P_t is aangegeven. Deze periodetijd moet geheel afhankelijk van de tijdsduur van de noten worden gekozen.
Stel u hebt gekozen voor de omhullende S14 en u wilt gedurende het spelen van een noot twee volledige variaties horen. De tijdsduur van de toon moet dan vier maal zo groot zijn als de periodetijd in afb. 9.

Ten eerste moet deze periodetijd worden ingevoerd met de opdracht Mn, waarin n een getal van 1 tot en met 65535 kan zijn. Hoe groter n des te groter de periodetijd. De waarde van n is te berekenen met $n = 6991 * P_t$. Als halve noten worden gespeeld en het tempo is T60 dan is de tijdsduur van een hele noot 240/60 = 4 sec. Een halve noot duurt dan 2 sec. Deze moet vier keer zo lang zijn als de periodetijd in afb. 9 (in dit voorbeeld) zodat P_t .5 sec moet zijn. Nu is n voor Mn te berekenen: n = 6991*.5, hetgeen moet worden afgerond op 3496 omdat slechts gehele getallen kunnen worden ingevoerd. Probeer nu eens

play''t60L2o4 s14 m 3496 ar4b''

Ook nu zijn de nieuwe gegevens voor S en M voor de duidelijkheid weer met spaties ingevoerd. Het effect van de omhullende is dus niet alleen van de keuze van S afhankelijk maar ook van M in combinatie met de duur van de tonen (T en L).
We gaan weer terug naar onze variabelen A$ en B$. Die veranderen we als volgt:

a$ = ''s1m4661t90L8o5eco4go5c.d16c''
b$ = ''s1m4661t90L8o3r8ggr8gg''

Zoals u ziet zijn nu de gegevens S en M voor de omhullende ingevoerd.
We maken er nog een derde variabele bij:

```
c$ = "s1m4661t90L8o3ceecee"
```

We gaan deze drie na elkaar beluisteren:

```
play a$: play b$: play c$
```

U hoort nu de diverse noten duidelijk van elkaar gescheiden, terwijl het effect van de omhullende nu een geluid veroorzaakt dat een beetje op dat van een piano lijkt. Probeer nu

```
play a$,b$,c$
```

In dit geval hoort u de tonen van de drie variabelen niet na elkaar, maar tegelijk. Er wordt nu en driestemmige melodie gespeeld met de drie kanalen van de PSG. De volgende regels voeren we in om de invloed van de omhullende te demonstreren.

```
a$ = "s13m466t90L8o5eco4go5c.d16c"
b$ = "s13m466t90L8o348ggr8gg"
c$ = "s13m466t90L8o3ceecee"
play a$,b$,c$
```

Ten opzichte van de vorige variabelen zijn alleen S en M veranderd. De beginsituatie na het starten van de computer is S13 en M255.

Als laatste rest ons nog het invoeren van het volume. Hiervoor dient in de string achter PLAY de opdracht Vn geplaatst te worden waarin n een getal van 0 tot en met 15 kan zijn. Het getal 15 geeft de grootste geluidssterkte. De beginsituatie is V8. V dient niet in één string met S en M te worden ingevoerd. Staat V in de string achter S en M dan worden S en M genegeerd. Staan S en M achter V in de string dan wordt V genegeerd. Druk eens op de CTRL en de STOP toetsen en probeer het volgende in de direct mode:

```
d$ = "o5c": play"v15cdefgab xd$;"
```

Ten eerste hoort u nu als het goed is de invloed van V15. Ten tweede ziet u nog een manier om een stringvariabele in de string achter PLAY in te voeren, in dit geval d$ die moet worden voorafgegaan door "X" en worden afgesloten door ";". Op elke plaats in de string kan op deze manier een variabele worden ingevoegd (alleen bij PLAY).

11.3. Invoeren van de gegevens met het SOUND statement

De PSG bezit registers voor de gegevens die hij nodig heeft voor het opwekken van de tonen. Deze registers zijn niet in de "normale" geheugenruimte opgenomen en kunnen worden ingeschreven met het SOUND statement, dat overeenkomt met het POKE statement maar alleen betrekking heeft op de PSG registers. Het SOUND bevel moet dus gevolgd worden door het adres van het register. In dit geval het registernummer en een getal tussen 0 en met 255 dat in het register geschreven moet worden. De gegevens die ingevoerd moeten worden zijn:

a. Een getal dat bepalend is voor de frequentie van de toon. Het grootste getal dat hiervoor kan worden ingevoerd is 4095 zodat hiervoor per stem twee registers nodig zijn. Voor de drie stemmen zijn dus zes registers nodig voor het invoeren van de frequenties.
b. Een getal moet worden ingevoerd waarmee "aangewezen" wordt welke stem het geluid moet voortbrengen.
c. Het volume van elke stem moet apart in te stellen zijn. Hiervoor zijn drie registers nodig.
d. De omhullende moet ingesteld kunnen worden.
e. De periodetijd voor de omhullende moet ingesteld kunnen worden.

In totaal zijn veertien registers nodig voor het invoeren van deze gegevens. De indeling van deze registers geeft lijst 8.

register-nummer	functie
0	Lage byte frequentiegetal stem 1 (LF1)
1	Hoge byte frequentiegetal stem 1 (HF1)
2	Lage byte frequentiegetal stem 1 (LF2)
3	Hoge byte frequentiegetal stem 2 (HF2)
4	Lage byte frequentiegetal stem 3 (LF3)
5	Hoge byte frequentiegetal stem 3 (HF3)
6	Getal voor ruisfrequentie
7	Voor het kiezen van de stem
8	Volume stem 1
9	Volume stem 2
10	Volume stem 3
11	Lage byte frequentiegetal omhullende
12	Hoge byte frequentiegetal omhullende
13	Voor het patroon van de omhullende

Lijst 8

De PSG is ook in staat om ruis op te wekken. Er is daarom ook een register nodig voor het invoeren van het getal voor de ruisfrequentie.

Bij het opwekken van de tonen gaat de PSG in principe niet uit van de frequentie maar van de periodetijd. Dat is de tijd die één periode van de trilling duurt en die is dus groter nagelang de frequentie lager is. Daarom is dan ook het getal dat u moet invoeren voor een toon het grootst voor de laagste frequentie. De grootte van het getal dat u moet invoeren is te berekenen met

Gf = 111860.7813/F

Hierin is F de frequentie van de toon die u wilt horen. Het grootste getal dat u kunt invoeren is 4095. De laagste frequentie die ten gehoren kan worden gebracht is daarom 111860.7813/4095 = 27.3 Hz. Het getal Gf moet u splitsen in een hoge en een lage byte (HF en LF):

HF = INT(Gf/256): LF = INT(Gf-256*HF)

De waarden HF en LF voor de tonen van de acht octaven van de chromatische toonladder die door de PSG kunnen worden opgewekt hoeft u niet meer te berekenen. Die zijn gegeven in lijst 7.

Het getal in register 7 geeft aan welke stem door de PSG moet worden ingeschakeld. Het geluid van een stem gaat via een gelijk genummerd kanaal (stem 1, kanaal 1 enz). Van dit register worden zes bits gebruikt, drie voor de "geluidskanalen 1, 2 en 3 en drie voor de "ruiskanalen" 1, 2 en 3. Als alle bits 1 zijn dan zijn de kanalen uitgeschakeld.

	ruis	geluid	
7 6	5 4 3	2 1 0	bitnummer
	3 2 1	3 2 1	kanaalnummer
0 0	1 1 1	1 1 1	register 7

Voor het geluid zijn de bits 0 tot en met 2, voor de ruis de bits 3 tot en met 5. Zijn alle bits (alle kanalen uitgeschakeld) dan is de inhoud van het register 63 (zie paragraaf 9.2.). U kunt een kanaal inschakelen door het betreffende bit 0 te maken. Het getal in het register laat zich dan eenvoudig berekenen door de plaatswaarde van het betreffende bit af te trekken van het getal 63. Voor het geluidskanaal 1 is de plaatswaarde van het bit 1 zodat het getal in het register 63 – 1 = 62 moet zijn. Voor het kanaal 2 is de plaatswaarde 2 (63 – 2 = 61) en voor kanaal 3 is de plaatswaarde 4 (63 – 4 = 59). Wilt u de kanalen 1 en 3 inschakelen dan moet u het getal 63 verminderen met 1 voor kanaal 1 en 4 voor kanaal 3. De inhoud van register 7 moet dan zijn: 63 – (1 + 4) = 58. U kunt eventueel alle drie de geluidskanalen inschakelen.

Het volume van het geluid kunt u per kanaal regelen door in het betreffende register een getal van 0 (geluid uit) tot en met 15 (geluid op maximum) te schrijven.
Probeer eens het volgende programma:

```
10 'invoeren freq. stem 1 220 Hz. LF1
20 SOUND 0,253
30 'invoeren HF1
40 SOUND 1,1
50 'kanaal 1 aan
60 SOUND 7,62
70 'volume kanaal 1 max.
80 SOUND 8,15
```

Het programma veroorzaakt een toon van 220 Hz uit uw luidspreker. De gegevens voor de frequentie zijn met SOUND 0,253 en SOUND 1,1 in de register 0 (LF1) en 1 (HF1) geschreven (lijst 7). Kanaal 1 is geopend met SOUND 7,62 in regel 60 (63 – 1 = 62). Het volume is met SOUND 8,15 voor kanaal 1 op maximum gezet. Het geluid zal slechts te stoppen zijn met

SOUND 8,0

of door het indrukken van de CTRL en de STOP toetsen. Voer nu de volgende regels in (*geen* NEW):

```
70 'volume van max. naar nul
80 FOR V=15 TO 0 STEP -1
90 SOUND 8,V
100 'vertraag
110 FOR X=0 TO 50: NEXT X
120 'herhaal de vol. variatie
130 NEXT V: GOTO 80
```

Het volume zal nu vertraagd (regel 110) worden verminderd van max. naar nul door de regels 80 en 130.
We hebben nu zelf de omhullende gemaakt. De periodetijd is afhankelijk van de vertraging in regel 110. Verander deze maar eens:

```
110 FOR X=0 TO 10: NEXT X
```

We kunnen ook de omhullende zelf veranderen:

```
70 'volume van nul naar max.
80 FOR V=0 TO 15
```

Op deze manier kunnen we de omhullende maken zoals we dat zelf willen. We kunnen zoals u weet ook de computer dat werk laten doen:

```
70 'invoeren periodetijd lage byte
80 SOUND 11,49
90 'invoeren periodetijd hoge byte
100 SOUND 12,8
110 'invoeren partroon omhullende
120 SOUND 13,8
130 'inschakelen v.d. omhullende
140 SOUND 8,16
```

Het getal voor de periodetijd van de omhullende is berekend met de formule $n = 6991 * P_t$ zoals we die ook bij het statement PLAY hebben gebruikt. Voor de periodetijd is 0.3 sec. aangehouden:
$n = 6991 * .3 = 2097.3$. Van dit getal moet de hoge byte en de lage byte worden bepaald.
Hbyte = INT(n/256); Hbyte = INT(2097.3/256) = 8.
In regel 100 is deze in register 12 geschreven.
Lbyte = INT(n − 256*Hbyte).
Lbyte = (2097.3 − 256*8) = 49. Met deze waarde wordt register 11 ingeschreven (regel 80).

Voor de omhullende is nummer 8 gekozen (afb. 9). In regel 120 is dit getal ingevoerd. In het register voor het volume van kanaal 1 moet nu het getal 16 worden geschreven om de variatie van de geluidssterkte (de werking van de omhullende) in te schakelen (regel 140). Zoals u ziet kan evenmin als met het statement PLAY hier het volume worden ingesteld.
Als alles goed is gegaan ziet het programma in uw geheugen er nu als volgt uit:

```
10 'invoeren freq. stem 1 220 Hz. LF1
20 SOUND 0,253
30 'invoeren HF1
40 SOUND 1,1
50 'kanaal 1 aan
60 SOUND 7,62
70 'invoeren periodetijd lage byte
80 SOUND 11,49
90 'invoeren periodetijd hoge byte
100 SOUND 12,8
110 'invoeren partroon omhullende
120 SOUND 13,8
130 'inschakelen v.d. omhullende
140 SOUND 8,16
```

We gaan nu proberen met een rijtje toetsen een toonladder te laten klinken. De enige karakters waarvan de toetsen netjes op een rijtje staan zijn de toetsen 1 tot en met 9. Voor een toonladder hebben we daarvan de toetsen 1 tot en met 8 nodig.

```
60 'voorkeuze van het octaaf
70 O=4
80 'frequenties voor de tonen
90 F(1)=32.625: F(2)=36.6875: F(3)=41.25
100 F(4)=43.625: F(5)=49.0625
110 F(6)=55: F(7)=61.5625: F(8)=65.25
120 'geluid via kanaal 1
130 SOUND 7,62: SOUND 8,16
140 'periodetijd omhullende
150 SOUND 11,49:. SOUND 12,8
160 'wacht op een toets
170 A$=INKEY$: IF A$="" THEN 170
180 'bepaal welke toets
190 A=VAL(A$)
200 'terug bij verkeerde toets
210 IF A<1 OR A>8 THEN 170
220 'bereken freq. getal v.d. toon
230 F=F(A)*2^(O-1): G=111860.7813#/F
240 HF=INT(G/256): LF=INT(G-HF*256)
250 'invoering periodetijd v.d. toon
260 SOUND 0,LF: SOUND 1,HF
270 'zet geluid aan
280 SOUND 13,0
290 'voor de volgende toets
300 GOTO 170
```

In dit programma wordt uitgegaan van de frequenties van de tonen uit het laagste octaaf. De frequenties worden in de regels 90 tot en met 110 ingevoerd. De tonen die we te horen krijgen zijn die van het vierde octaaf waarvoor in regel 70 o = 4 wordt ingevoerd. De berekening van de juiste frequentie voor een toon vindt plaats in regel 230 met $F = F(A) * 2^{(o-1)}$. Voor elke hogere octaaf moet de frequentie van de tonen met 2 worden vermenigvuldigd. Voor octaaf 2 met 2^1, voor octaaf 3 met $2 * 2^1 = 2^2$ enz. In regel 190 wordt de numerieke waarde bepaald van de ingedrukte toets. Deze vinden we in de variabele A zodat met F(A) de frequentie voor de ingedrukte toets wordt geselecteerd. Behalve de juiste frequentie wordt in regel 230 ook het getal berekend dat voor die frequentie moet worden ingevoerd. Na het berekenen van de LF waarde en de HF waarde hiervan vindt invoering in de registers 0 en 1 plaatst voor kanaal 1 (regel 260). De verdere gegevens voor de tonen zijn ingesteld in de regels 130 en 150. Met SOUND 13,0 wordt niet alleen de golfvorm gekozen maar wordt ook het geluid ingeschakeld. Uitschakelen hoeft niet, door de gekozen golfvorm klinkt de toon slechts één keer. Uiteindelijk wordt teruggegaan

naar regel 170 waar op de volgende toets wordt gewacht.
Als we het programma met de volgende regels uitbreiden maken we het geschikt voor het spelen van de tonen van vijf octaven. Hierbij worden de functietoetsen 1 tot en met 5 gebruikt en het nummer van de toets bepaalt het nummer van het gekozen octaaf.

```
10 'invoering van octaaf nr.
20 ON KEY GOSUB 320,330,340,350,360
30 'inschakelen van de toetsen
40 KEY(1) ON: KEY(2) ON: KEY(3) ON
50 KEY(4) ON: KEY(5) ON
310 'bepaal het octaaf nummer
320 O=1: RETURN 170
330 O=2: RETURN 170
340 O=3: RETURN 170
350 O=4: RETURN 170
360 O=5: RETURN 170
```

Een overeenkomstig statement als in regel 20 hebben we al eens ontmoet in ON INTERVAL GOSUB . . . In dit geval slaat ON KEY op het indrukken van een functietoets en wel een functie toets die met KEY ON daarvoor is ingeschakeld. Hier zijn dat de functietoetsen 1 tot en met 5 waarmee een gelijkgenummerd octaaf kan worden ingeschakeld. Achter ON KEY GOSUB kunnen maximaal 5 regelnummers van subroutines worden ingevuld. Er kunnen daarom niet meer dan vijf octaven worden gekozen.
Steeds zal het programma, als het met één van de regels van het hoofdprogramma bezig is, worden onderbroken als op een functietoets wordt gedrukt en naar de subroutine gaan die door het nummer van de toets wordt aangegeven (zoals bij ON A GOSUB . . .). Na het afwerken van de subroutine wordt steeds weer naar regel 170 teruggesprongen voor het wachten op het indrukken van een toets. Kenmerkend voor de statements ON KEY GOSUB en ON INTERVAL GOSUB is dat de opdracht altijd wordt uitgevoerd als een programmaregel (geeft niet welke regel) van het hoofdprogramma wordt afgewerkt. Steeds als de intervaltijd is verstreken (bij ON INTERVAL) of als op de betreffende functietoets wordt gedrukt (bij ON KEY) wordt het programma onderbroken. De opdracht werkt bij wijze van interrupt.

11.4. Ruis

Een belangrijke functie van de PSG bij het veroorzaken van geluidseffecten is het opwekken van ruis. Een kenmerk van ruis is dat het bestaat uit een zeer groot aantal zeer willekeurige trillingen waarin geen bepaalde toon is te onderkennen. De ruis kan worden ingeschakeld met de bits 3, 4 en 5 van register 7. Voor het inschakelen van kanaal 1 moet bit 3 nul worden, voor kanaal 2 moet bit 4 nul worden en voor kanaal 3 moet bit 5 nul worden. Het beste effect wordt verkregen als alle drie de kanalen worden gebruikt. Probeer het volgende programma.

```
10 SOUND 6,27
20 SOUND 7,55
30 SOUND 8,16
60 SOUND 11,49
70 SOUND 12,8
80 SOUND 13,8
```

In register 6 moet het ruisgetal worden ingevoerd. Hiermee wordt een afsnijfrequentie ingesteld voor het bepalen van de ''klankkleur'' van de ruis. Het getal kan worden berekend met $n = 111860.7813/f$ waarin f de afsnijfrequentie is. Hoe lager de afsnijfrequentie des te ''voller'' de ruis klinkt (de lagere tonen worden afgesneden). N kan een getal zijn van 0 tot en met 31. U hoeft het niet uit te rekenen. U kunt het vinden in lijst 7. In dit programma wordt alleen kanaal 1 ingeschakeld; $63 - 8 = 55$ (regel 20). De rest van het programma spreekt voor zichzelf. Als u wilt weten hoe het klinkt als alle kanalen worden ingeschakeld, voeg dan de volgende regels toe.

```
20 SOUND 7,7
40 SOUND 9,16
50 SOUND 10,16
```

In dit geval moet in register 7 het getal 7 worden geschreven: $63 - (32 + 16 + 8) = 7$. Het invoeren van veel programmaregels met het statement SOUND kunt u vergemakkelijken door een functietoets te gebruiken:

key 1, ''sound''

U kunt de ruis voor het bereiken van geluidseffecten ook samen laten klinken met een gewone toon. Probeer hiervoor het volgende programma:

```
10 'voor veertien registers
20 FOR R=0 TO 13
30 READ D
40 'het invoeren van de gegevens
```

```
50 SOUND R,D: NEXT R
60 'LF en HF voor 3 kanalen
70 DATA 179,8,179,8,179,8
80 'ruisgetal
90 DATA 27
100 'alle kanalen inschakelen
110 DATA 0
120 'omhullende inschakelen
130 DATA 16,16,16
140 'voor de periodetijd omhullende
150 DATA 49,8
160 'omhullende nummer 8
170 DATA 8
```

Er is hier weer eens een andere manier gebruikt voor het invoeren van de gegevens. Het READ statement leest de DATA en met SOUND, R,D wordt deze data in de juiste registers geschreven. Bij de diverse DATA is duidelijk aangegeven welke functie ze heeft. Met SOUND 7,0 worden alle bits van register 7 nul zodat alle kanalen geopend zijn. De frequentie van de toon waarmee de ruis samenklinkt is 65.2 Hz. Als u regel 70 als volgt verandert wordt de frequentie van de toon 880 Hz:

```
70 DATA 127,0,127,0,127,0
```

12. Grafische functies, de veertigkolommode

12.1. De indeling van het scherm

Het volgende programma plaatst één van de karakters uit de "karakterset" van de computer in het midden van het scherm.

```
10 SCREEN 0: WIDTH(40)
20 'cursor 11 plaatsen naar onder
30 FOR Y=0 TO 10
40 PRINT CHR$(31);: NEXT Y
50 'cursor 19 plaatsen naar rechts
60 FOR X=0 TO 18
70 PRINT CHR$(28);: NEXT X
80 PRINT CHR$(1)CHR$(66)
```

In lijst 2 kunt u zien dat het karakter het ASCII codegetal 2 heeft. Omdat dit kleiner is dan 32 moet dit met PRINT CHR$(1)CHR$(64+2) op het scherm worden geplaatst. Het volgende programma heeft hetzelfde effect:

```
10 CLS
20 'cursor naar kolom 19 en regel 11
30 LOCATE 19,11
40 PRINT CHR$(1)CHR$(66)
```

U ziet hoe sommige opdrachten het ons makkelijk maken! LOCATE X,Y is een opdracht die de cursor naar kolomnummer X laat gaan (meest linkse kolom heeft nummer 0) en naar regelnummer Y (de bovenste regel heeft nummer 0). Probeer in de direct mode:

print vpeek(459)

Het antwoord zal zijn: 2. Probeer nu ook:

cls: vpoke 459,2

CLS wist het gehele scherm. Het karakter dat u nu weer op het midden van het scherm ziet staan moet dus door de VPOKE opdracht daar zijn geplaatst.

In het apparaat dat u heeft aangeschaft bevinden zich in principe twee computers. Met de eerste daarvan bent u steeds al bezig geweest. Met de tweede ook, maar dat heeft u nog niet zo gemerkt. De tweede computer is opgebouwd uit de Video Display Processor (VDP) en een 16 Kbyte RAM geheugen dat we de "VIDEORAM" zullen noemen. Het geheel bevindt zich dan ook buiten de normale 64 Kbyte geheugenruimte van de "hoofdcomputer" die u tot nu toe al het werk hebt laten verrichten. De Video Display Processor verricht al het werk dat nodig is om het beeld op uw scherm te krijgen en gebruikt daarbij gegevens die in de VIDEORAM worden opgeslagen. Wilt u zelf de VIDEORAM inschrijven of uitlezen dan hebt u daarvoor speciale POKE- en PEEK opdrachten nodig: VPOKE en VPEEK.
In de VIDEORAM is een geheugenblok gereserveerd waarin voor elke plaats op het scherm voor een karakter een geheugenplaats is aangewezen. In elke geheugenplaats kan een getal van 0 tot en met 255 worden geplaatst en dat getal moet gelijk zijn aan de ASCII code van het karakter dat op het scherm zichtbaar moet worden. Er zijn op het scherm vierentwintig regels (drieentwintig plus de onderste regel met de KEY-functies) en op elke regel kunnen maximaal 40 karakters worden geschreven. Steeds als de VDP een volledig schermbeeld moet vormen worden stuk voor stuk alle geheugenplaatsen voor het scherm "afgetast" en het karakter dat in een bepaalde geheugenplaats met de ASCII code is aangegeven wordt op de bijbehorende plaats van het scherm geschreven. Dat houdt in dat alle geheugenplaatsen voor het scherm met een codegetal moeten zijn gevuld. Probeer eens:

print vpeek(460)

De geheugenplaats 460 volgt op degene die we net hebben ingeschreven met VPOKE 459,2. Het resultaat zal 32 zijn, de ASCII code voor spatie.
Laat het volgende in de directmode uitvoeren:

cls: print: vpoke 0,249

Voer dit soort opdrachten nooit op de laatste regel van het scherm uit! U ziet nu een bal getekend op de meest linkse plaats van de bovenste regel (be-

denk dat het kader niet van het veld is te onderscheiden). Probeer ook:

width(30): vpoke 0,249

Hoe groot uw tekstbreedte ook is, steeds zal de bal verschijnen op de meest linkse plaats van de bovenste regel. Deze plaats hoort kennelijk bij de geheugenplaats 0. RUN nu het volgende programma (denk om NEW).

```
10 FOR X=1 TO 39
20 VPOKE X-1,32: VPOKE X,249
30 NEXT X
```

De bal doorloopt de bovenste regel van het veld en blijft op het eind van deze regel staan. In dit programma wordt eerst geheugenplaats 0 gewist (POKE X – 1,32). Daardoor wordt een spatie geschreven op de plaats waar de bal eerst stond. Daarna wordt de bal op de plaats daarnaast geprint. Door het verhogen van X lijkt de bal de gehele bovenste regel te doorlopen. Verander regel 10 in:

```
10 FOR X=40 TO 79
```

De bal doorloopt nu de tweede regel. Verander regel 10 nogmaals.

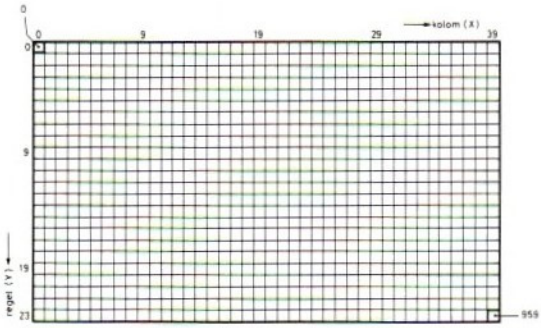
```
10 FOR X=920 TO 959
```

Nu wordt de onderste regel van het veld (die normaal niet te beschrijven is) doorlopen.

De indeling van het scherm zal nu duidelijk zijn. Nummeren we de plaatsen op het veld in overeenstemming met de bijbehorende geheugenplaatsen, dan zijn de plaatsen oplopend genummerd, regel voor regel, van boven naar beneden en beginnend aan de linker zijde. De eerste geheugenplaats is 0.

Omdat er 24 regels zijn van 40 kolommen zijn er 24*40 = 960 geheugenplaatsen voor het gehele scherm nodig, van 0 tot en met 959 (afb. 10). Geven we de kolommen met X aan (kolom 0; X = 0, kolom 1; X = 1 enz.) dan kunnen we de geheugenplaats voor een bepaalde plaats P op het scherm vinden met

P = BAD + X + 40*Y



afb. 10

We moeten namelijk voor elke regel (Y) 40 geheugenplaatsen rekenen. In deze formule is BAD het adres van de geheugenplaats die correspondeert met de meest linkse plaats op de bovenste regel van het scherm (beginadres). In dit geval is BAD 0.
Een plaats op het scherm waar een karakter moet komen wordt met X en Y aangegeven. Moet een karakter komen op regelnummer 13 en op plaatsnummer 25 op die regel (kolomnummer 25) dan is Y = 13 en X = 25.
Door deze twee waarden in bovenstaande formule in te vullen, vinden we de geheugenplaats waar de ASCII code van het karakter moet worden ingeschreven met VPOKE (0 + 25 + 40*13 = 545). De waarden X en Y van een punt op het scherm noemen we de coördinaten van dat punt. Het geheugenblok waarin zich de geheugenplaatsen voor het scherm bevinden wordt hier het ''schermgeheugen'' genoemd.

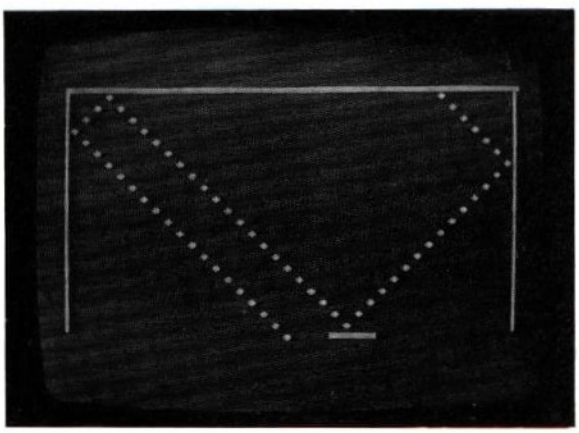
12.2. Bewegende beelden

U hebt in de vorige paragraaf een bal over het scherm kunnen zien bewegen. Dat ging eenvoudig. Steeds de bal in een volgende plaats printen en de vorige wissen. Probeer eens wat voor uitwerking het volgende programma heeft.

```
20 WIDTH(40)
70 'maak rnd tot toevalsgetal
80 CLS: PRINT "druk op een toets"
90 A$=INKEY$: A=RND(1)
100 IF A$="" THEN 90
220 XD=1
230 'bal willekeurig vanaf 1e lijn
240 X=INT(35*RND(1))+2: YD=1: Y=1
250 LOCATE X,Y: PRINT CHR$(249)
260 S=50
360 'vertraag de bal
370 FOR V=0 TO S: NEXT
380 'wis de bal
390 LOCATE X,Y: PRINT CHR$(32)
400 'voor de volgende plaats v.d. bal
```

```
410 X=X+XD: Y=Y+YD
420 'bepaal of de bal de zijmuur raakt
430 IF X=1 OR X=38 THEN XD=-XD
460 'bepaal of de bal verdwijnt
470 IF Y=22 THEN 240 ELSE 250
```

Dit programma laat een bal onder een hoek van 45° bewegen over uw scherm. Soms weerkaatst hij tegen de zijkanten van het scherm maar steeds verdwijnt hij aan de onderkant, waarna onmidellijk aan de bovenkant weer een nieuwe bal verschijnt. We zullen het programma doorlopen vanaf de situatie direct na RUN. Met regel 20 zult u dacht ik niet zoveel moeite hebben. De bal moet op een willekeurige plaats bovenaan het scherm verschijnen. Daarom wordt de X-waarde met een RND-getal bepaald. De regels 70 tot en met 100 zorgen voor een echt randomgetal en in regel 240 wordt de X-waarde gegenereerd. In eerste instantie wordt deze waarde steeds met 1 vermeerderd door XD = 1 (regel 220) in regel 410. Om de bal schuin over het scherm te laten gaan wordt ook de Y-waarde regelmatig verhoogd. In regel 240 is daarvoor YD 1 gemaakt. Door Y = 1 wordt de bal in regel 250 bovenaan het scherm geprint. Om de bal niet met zo'n grote snelheid over het scherm te laten gaan volgt een vertraging waarvan de lengte bepaald is door S = 50 in regel 260. Na deze vertraging wordt de bal gewist (regel 390). Als X de waarde 38 heeft gekregen dan raakte de bal de zijkant van het scherm en moet hij van richting worden veranderd. Nu moet de X-waarde steeds kleiner worden met X = X − 1. Daarvoor wordt door XD = − XD in dit geval XD negatief. Verkrijgt X de waarde 1 dan moet X weer worden vermeerderd. Door XD = − XD wordt de negatieve waarde van XD verandert in een positieve. In regel 470 wordt bepaald of de bal aan de onderkant van het scherm is. Zo ja dan beginnen we in regel 240 weer met een nieuwe. Anders wordt in regel 250 de bal op de nieuwe plaats geprint.



Raakt de bal de zijkant van het scherm dan lijkt het alsof hij een muur raakt (X = 1 of X = 38). Als we deze muur erbij tekenen en onderaan het scherm een soort slaghout vormen waarmee we de bal kunnen terugslaan en ook nog wat geluid toevoegen dan hebben we een compleet programma voor tennis in je eentje: squash.

```
10 'squash
20 WIDTH(40)
30 'freq. getal ruis, kanaal 1 aan
40 SOUND 6,27: SOUND 7,54: SOUND 8,16
50 'instellen periodetijd omhullende
60 SOUND 11,49: SOUND 12,8: SOUND 0,55
70 'maak rnd tot toevalsgetal
80 CLS: PRINT "druk op een toets"
90 A$=INKEY$: A=RND(1)
100 IF A$="" THEN 90
110 'print bovenkant muur
120 LOCATE 1,0: PRINT STRING$(38,220)
130 'print zijmuren
140 FOR Y=1 TO 21: LOCATE 0,Y
150 PRINT CHR$(222): LOCATE 39,Y
160 PRINT CHR$(198);: NEXT Y
170 'print hoekpunten
180 LOCATE 0,0: PRINT CHR$(212)
190 LOCATE 39,0: PRINT CHR$(214)
200 'plaats slaghout in het midden
210 XP=17: GOSUB 600
220 XD=1
230 'bal willekeurig vanaf 1e lijn
240 X=INT(35*RND(1))+2: YD=1: Y=1
250 LOCATE X,Y: PRINT CHR$(249)
260 S=50
270 'bepaal of het slaghout is geraakt
280 AD= (Y+1)*40+X
290 IF VPEEK(AD)=223 THEN GOSUB 630
300 'ga na of CRSR toets in is
310 A$=INKEY$: IF A$=""THEN 370
320 'CRSR toets rechts
330 IF ASC(A$)=28 THEN GOSUB 510
340 'CRSR toets links
350 IF ASC(A$)=29 THEN GOSUB 560
360 'vertraag de bal
370 FOR V=0 TO S: NEXT
380 'wis de bal
390 LOCATE X,Y: PRINT CHR$(32)
400 'voor de volgende plaats v.d. bal
410 X=X+XD: Y=Y+YD
420 'bepaal of de bal de zijmuur raakt
430 IF X=1 OR X=38 THEN XD=-XD: GOSUB 490
440 'bepaal of de bal de bovenkant raakt
450 IF Y=0 THEN GOSUB 660
460 'bepaal of de bal verdwijnt
470 IF Y=22 THEN 240 ELSE 250
480 'subr. schakelt geluid in
490 SOUND 1,9: SOUND 13,0: RETURN
500 'subr. plaatst slaghout naar rechts
510 XP=XP+2: IF XP=35 THEN XP=33
520 'wis het slaghout
```

```
530 VPOKE 879+XP,32: VPOKE 880+XP,32
540 GOTO 600
550 'subr. plaatst slaghout naar links
560 XP=XP-2: IF XP=-1 THEN XP=1
570 'wis het slaghout
580 VPOKE 885+XP,32: VPOKE 886+XP,32
590 'subr. tekent het slaghout
600 FOR X1=0 TO 3: VPOKE 881+XP+X1,223
610 NEXT X1: S=40: RETURN
620 'subr. maakt geluid voor slaghout
630 SOUND 1,0: SOUND 13,0
640 YD=-YD: RETURN
650 'subr. maakt geluid voor bovenkant
660 SOUND 1,9: SOUND 13,0
670 YD=-YD: Y=2: RETURN
```

Het slaghout is te verplaatsen met de toetsen voor horizontale cursorbewegingen. We zullen het programma in "blokken" doornemen.
De regels van het vorige programma zijn met de overeenkomstige regelnummers hierin opgenomen en behoeven geen toelichting. Alleen regel 430 hiervan is veranderd. Hiermee moet u dus even rekening houden bij het intoetsen.
Voor het printen van de bovenkant van de muur is een nieuw statement toegepast in regel 120. Het statement STRING$(A,B) geeft een string van karakters. Het aantal karakters wordt door het getal A aangegeven. Het getal B geeft de ASCII code van de gewenste karakters. Zo wordt door regel 120, te beginnen met de plaats 1,0 op het scherm een rij van 38 karakters geprint. Welk karakter kunt u zien in lijst 2 onder het codegetal 220. U kunt ook rechtstreeks aangeven welk karakter u geprint wilt hebben, door in plaats van het getal, het karakter tussen aanhalingstekens in te voeren. Probeer maar eens:

```
a$= string$(10,"*")
locate 11,15: print a$
```

Op welke manier de rest van de muur wordt geprint is eenvoudig te volgen (regels 140 tot en met 190). Het slaghout moet op schermregel 23 worden geprint. Dit kan niet met een gewone PRINT opdracht omdat hierna door scrolling het gehele beeld één regel naar boven gaat.
Daarom wordt in de regels 600 en 610 (subroutine) het slaghout met een VPOKE opdracht getekend. XP geeft hierbij de plaats op de regel aan.
Deze is bij aanvang in het midden (regel 210). Na het printen van de bal wordt eerst nagegaan of het slaghout is geraakt. Dit moet zich dan op de regel direct *onder* die van de bal bevinden (Y + 1). In regel 290 wordt met VPEEK nagegaan of zich op dit adres het karakter van het slaghout bevindt. Zo ja dan wordt in de subroutine op regel 630 een toon veroorzaakt waarna met YD = −YD de bal weer omhoog gaat. Dan is het weer mogelijk dat de bal de bovenrand raakt. Dit wordt nagegaan in regel 450. Is dat het geval dan wordt weer een geluid veroorzaakt en wordt de bal opnieuw van richting veranderd (subroutine op regel 660).
Eenzelfde soort van geluid wordt gemaakt als de bal de zijmuren raakt (regel 430). Direct na het printen van de bal wordt met INKEY$ nagegaan of een cursortoets in ingedrukt. Is in het geheel geen toets ingedrukt dan geeft ASC(A$) een foutmelding. Vandaar dat met IF A$ = " " THEN 370 de regels 330 en 350 worden overgeslagen. Is er een toets ingedrukt dan wordt het slaghout verplaatst (zie voor de ASCII getallen lijst 3). Dit verplaatsen kost tijd. Vandaar S = 40 in regel 610. De vertraging in regel 370 wordt daarmee kleiner dan door S = 50 in regel 260.

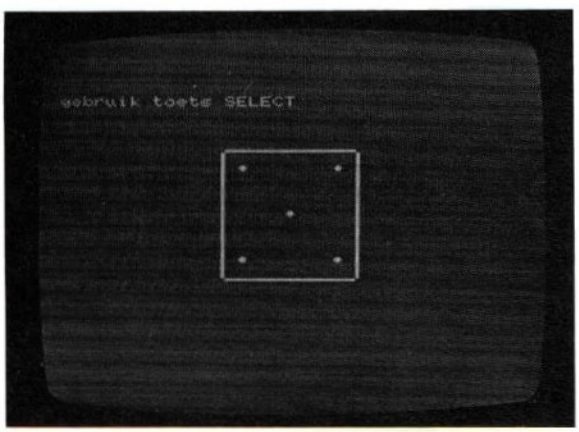
12.3. De dobbelsteen

Het volgende programmavoorbeeld laat u geen nieuwe instructies zien, maar wel de toepassing van een aantal van de statements die we tot nu toe ontmoet hebben. De bedoeling is dat de afbeelding van een dobbelsteen op het scherm komt, dus met "ogen" en die zich een beetje gedraagt zoals een dobbelsteen waarmee een worp gedaan is, dat wil zeggen totdat hij "stil ligt" zien we eerst het aantal ogen steeds veranderen. Als "oog" is het karakter met het ASCII codegetal 249 genomen, een grote stip dus. De dobbelsteen wordt ongeveer in het midden van het scherm afgebeeld. Het volgende programma kunt u hiervoor gebruiken. Om het programma makkelijker te kunnen volgen zijn de sprongadressen vaak regels met een REM statement. Als u deze regels niet wilt invoeren dan zult u de sprongadressen moeten veranderen.

```
10 'groen veld, gele karakters
20 CLS: COLOR 10,12: WIDTH(40)
30 PRINT"gebruik toets SELECT"
40 'speler doet een worp
50 A$=INKEY$: A=RND(1)
60 IF A$<>CHR$(24) THEN 50
70 'teken de randlijnen v.d. steen
80 CLS: S=249: GOSUB 230: T=0
90 'bepaal het aantal ogen
100 A=INT(6*RND(1))+1
110 ON A GOSUB 460,430,400,360,330,290
120 T=T+1
```

```
130 'volgend rnd getal na wissen
140 IF S=32 THEN S=249: GOTO 90
150 'bereken de vertraging
160 Y=150+25*T^2
170 'vertraging
180 FOR V=1 TO Y: NEXT V
190 'uitgerold, dan volgende worp
200 IF T=7 THEN PRINT CHR$(11): GOTO 30
210 'voor het wissen van de stand
220 S=32: GOTO 110
230 'omraming van de dobbelsteen
240 LOCATE 14,5: PRINT STRING$(11,192)
250 LOCATE 14,17: PRINT STRING$(11,195)
260 FOR Y=6 TO 16: LOCATE 13,Y
270 PRINT CHR$(222): LOCATE 25,Y
280 PRINT CHR$(198): NEXT Y: RETURN
290 'afbeelding voor 6
300 H=-4: V=0: GOSUB 480
310 H=4: V=0: GOSUB 480
320 GOTO 360
330 'afbeelding voor 5
340 GOSUB 360
350 GOTO 460
360 'afbeelding voor 4
370 H=4: V=4: GOSUB 480
380 H=-4: V=-4: GOSUB 480
390 GOTO 430
400 'afbeelding voor 3
410 GOSUB 430
420 GOTO 460
430 'afbeelding voor 2
440 H=4: V=-4: GOSUB 480
450 H=-4: V=4: GOTO 480
460 'afbeelding voor 1
470 H=0: V=0: GOTO 480
480 'print een stip/spatie
490 LOCATE 19+H,11+V
500 PRINT CHR$(S): RETURN
```

Het printen van een stip verzorgt de subroutine op regel 480. Als H = 0 en V = 0 dan wordt de stip voor de 1 geprint. Voor de andere stippen moeten H en/of V +4 of −4 zijn. Voor het printen van een bepaald aantal ogen is vaak gebruik gemaakt van een combinatie. Zo kan een 3 worden gevormd



met de 1 en de 2. Een vier wordt gevormd met de 2 en nog twee diagonaal geplaatste stippen. De 5 wordt gevormd uit de combinatie van de 1 en de vier en voor een zes wordt ook van de vier gebruik gemaakt. De variabele A bevat de waarde van de worp (regel 100). In regel 110 wordt dan een bepaalde subroutine gekozen.
Afhankelijk van de worp komt u bij een bepaalde subroutine terrecht.
Is A bijvoorbeeld 5 dan gaat u naar regel 330. Hier wordt eerst de subroutine op de regel 360 ingeschakeld voor het afbeelden van de 4 en daarna wordt naar regel 460 gesprongen voor de afbeelding van de 1. De RETURN in regel 500 sluit de subroutine af. Hier wordt dus een subroutine *binnen* een andere subroutine aangeroepen. Dit geeft geen enkel bezwaar en kan zelfs meerdere keren. Het tekenen van de omraming van de dobbelsteen gaat overeenkomstig de muur in het vorige programma en kan daarom geen moeilijkheden opleveren (subroutine op regel 230).

Na het starten van het programma worden we uitgenodigd op de toets SELECT te drukken. Deze handeling zet het ''rollen'' van de steen in gang. Op de gebruikelijke wijze is voor een echt toevalsgetal gezorgd. Na de worp wordt S 249 voor het printen van een stip en daarna wordt de omraming van de steen getekend (regel 80: GOSUB 230). In regel 100 wordt een randomgetal gegenereerd die een waarde kan hebben van 1 tot en met 6. Dan volgt het tekenen van de afbeelding van het aantal ogen.
Een teller T wordt opgehoogd. Regel 140 wordt overgeslagen. De tekening moet enige tijd zichtbaar blijven waarvoor een vertraging nodig is. De grootte van de vertraging wordt in regel 160 berekend met behulp van de waarde van T. In regel 180 wordt vertraagd. Regel 200 wordt overgeslagen en nu wordt S gelijk aan 32. Opnieuw wordt de afbeelding getekend, nu echter met spaties. Dat heeft tot gevolg dat de afbeelding wordt gewist. Daarna wordt in regel 140 S weer 249 en wordt *zonder* vertraging in regel 100 een nieuw randomgetal bepaald. Opnieuw wordt een afbeelding getekend. De vertraging die nu optreedt is belangrijk groter dan de vorige omdat T groter is geworden terwijl de vertraging toeneemt met T^2 (Y = 150 + 25*t^2). Het geheel herhaalt zich totdat bij T = 7 de steen is ''uitgerold'' en door regel 200 wordt teruggegaan voor de volgende worp.

12.4. De opbouw van de karakters

Alles wat met een computer te maken heeft is binair en bestaat in principe uit kleine eenheden. Deze eenheden kunnen niet meer dan 1 of 0 zijn. Zo is het ook met de opbouw van het scherm. Het "veld" van het scherm is opgebouwd uit 49152 eenheden die allemaal slechts in een bepaalde kleur voluit kunnen oplichten. Omdat deze eenheden elk in een andere kleur kunnen oplichten kunnen ze worden waargenomen als stippen. Vaak kan worden gesproken van oplichten in een voorgrondkleur of in een achtergrondkleur. Het oplichten van een stip in de voorgrondkleur is steeds het gevolg van een 1 in een geheugenplaats en het oplichten in een achtergrondkleur van een 0 in een geheugenplaats. Door de juiste combinatie van nullen en enen worden de karakters zichtbaar.

Het veld is georganiseerd in 192 stippenregels (dit ter onderscheid van de karakterregels) van 256 stippen per regel. Horizontaal kunnen maximaal 40 karakters op een regel en voor elk karakter zijn zes stippen nodig. Dat betekent dat voor de karakters 6*40=240 stippen worden gebruikt. Er blijven in de breedte dus 15 stippen ongebruikt.
Verticaal worden per karakter acht stippenregels gebruikt zodat er 192/8=24 karakterregels zijn. Hiervan is, zoals we reeds gemerkt hebben, de onderste regel (regel 23) niet met een PRINT statement te beschrijven. Dit is wel mogelijk met een VPOKE statement.
Uit het voorgaande zou volgen dat voor elk karakter een "rooster" van acht stippen hoog en zes stippen breed wordt gebruikt. Dit is niet waar. Voor elk karakter wordt een rooster van acht stippen hoog en ook *acht* stippen breed toegepast. Dit hangt uiteraard samen met het voor de computer welhaast magische getal 8 en dat slaat weer op het aantal bits wat in een geheugenplaats past. Omdat toch slechts zes stippen in de breedte ter beschikking zijn worden ze "overlappend" naast elkaar geplaatst. Afb. 11 geeft hiervan een voorstelling. Een rooster van acht bij acht stippen wordt ook wel een "matrix" genoemd en door in dit matrix een aantal stippen in een bepaald patroon te laten oplichten (de voorgrondkleur te laten aannemen) worden de karakters getekend. In afb. 11 is te zien hoe de A en de b worden gevormd. In het algemeen zijn de karakters zodanig gevormd dat de drie rechtse rijen "blank" zijn, twee om de overlapping mogelijk te maken en nog één om een ruimte te houden tussen de karakters op het scherm. Bij bepaalde karakters is dat niet mogelijk gebleken, zoals u zult zien als u het volgende in de direct mode uitvoert:

```
cls: vpoke 459,2
```

Dit figuur heeft u al eerder gezien, maar er had eigenlijk een volle maan op uw scherm moeten staan. Een gedeelte hiervan is nu afgedekt door de spatie rechts naast het karakter.

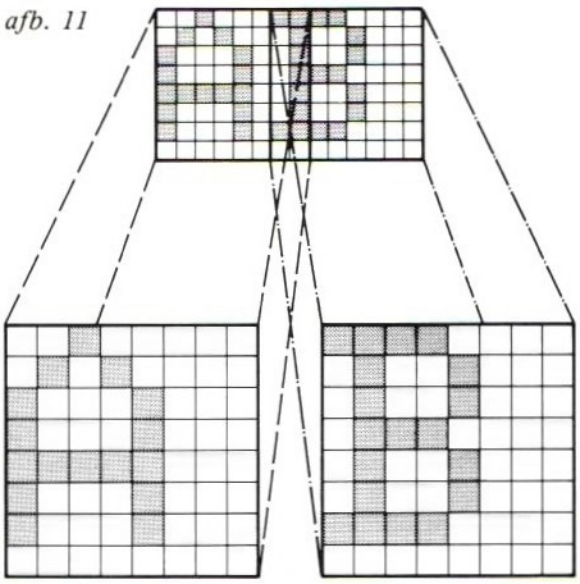
De gegevens voor elk karakter worden in geheugenplaatsen opgeslagen.
Per karakter zijn 8 geheugenplaatsen nodig. Er zijn 256 verschillende karakters zodat een blok van 8*256=2048 geheugenplaatsen nodig is voor alle karakters. Dit blok wordt de "karaktergenerator" genoemd. In de karaktergenerator vinden we steeds blokjes van acht geheugenplaatsen, voor elk karakter zo'n blokje. De eerste geheugenplaats van een blokje bevat de gegevens voor de bovenste regel van het karakter, de tweede geheugenplaats die voor de regel daaronder enz. Uit afb. 11 blijkt dat het blokje voor de hoofdletter A gevuld moet zijn met:

binair	hex	decimaal
00100000	20	32
01010000	50	80
10001000	88	136
11111000	F8	248
10001000	88	136
10001000	88	136
00000000	00	0

We noemen dit het "bitpatroon" voor het karakter A. Zie voor de omrekening van binair naar hexadecimale en decimale getallen hoofdstuk 9. De laatste regel van het karakter is "blank" om de karakters op de regels boven elkaar gescheiden te houden.
De geheugenblokjes voor de karakters in de karaktergenerator zijn in de volgorde geplaatst die overeenkomt met de ASCII code zoals die gegeven is in lijst 2. Het is daarom eenvoudig de gegevens voor een bepaald karakter te vinden. De formule hiervoor is

BAD+8*ASC

afb. 11



Waarin BAD het beginadres van de karaktergenerator is en ASC de ASCII code voor het betreffende karakter. U vindt dan het beginadres van het geheugenblokje voor het karakter. In de veertigkolommode is BAD 2048. Dit is een adres van de VIDEORAM. Direct na de keuze SCREEN 0 wordt vanaf dit adres de gegevens voor de karakters in de geheugenplaatsen van de karaktergenerator geschreven. Wilt u de gegevens voor het karakter C (ASC = 67) hebben dan vindt u het beginadres van het geheugenblokje hiervoor met

2048 + 8*67 = 2584.

De werkwijze voor het printen van een karakter op het scherm is als volgt: de computer "kijkt" naar het schermgeheugen, vindt op een bepaalde plaats (0 + X + 40*Y) een ASCII codegetal, berekent met 2048 + ASC*8 de plaats van het geheugenblokje in de karaktergenerator en print de gegevens die hij daar vindt op het scherm op de plaats die correspondeert met de gelezen geheugenplaats in het schermgeheugen. Al deze handelingen worden door de VDP verricht. Zoals u weet dienen de gegevens voor de karakters met het toetsenbord, met PRINT opdrachten of met VPOKE in het schermgeheugen te worden geplaatst. De plaatsen in dit geheugen die niet op deze wijze worden ingeschreven zijn door een voorgaande CLS of SCREEN X steeds gevuld met het getal 32 voor spatie.

De VDP bezit een aantal registers waarin hij de gegevens bewaart die nodig zijn bij het printen van de karakters. Eén van die gegevens is die voor de kleur. Het VDP registernummer hiervoor is 7. Probeer maar eens:

```
color 10,4
c=vdp (7): print (c and 240)/16
print c and 15
```

U ziet nu weer de kleurnummers terug die we met COLOR 10,4 hebben ingevoerd. De werking die plaats heeft gehad is als volgt:

Bitnummer	7	6	5	4	3	2	1	0
VDP register 7	1	0	1	0	0	1	0	0
Masker (240)	1	1	1	1	0	0	0	0
VDP (7) AND 240	1	0	1	0	0	0	0	0

(VDP (7) AND 240)/16 = 1010 (10 dec.)

Hier ziet u een toepassing van het werken met een masker. Het nummer voor de voorgrondkleur (de enen in het bitpatroon) vindt u dus in de hoogste vier bits van het VDP register 7. Het nummer voor de achtergrondkleur wordt gevormd door de vier laagste bits in dit register.

Bitnummer	7	6	5	4	3	2	1	0
VDP register 7	1	0	1	0	0	1	0	0
Masker (15)	0	0	0	0	1	1	1	1
VDP (7) AND 15	0	0	0	0	0	1	0	0

Bin. 00000100 is dec. 4

Wilt u de karakters op het scherm weer wit hebben voer dan het volgende uit:

vdp (7) = 15*16 + 4

Hierin is 15 het kleurnummer voor wit. COLOR 15 werkt overigens ook heel best. Wilt u alleen de achtergrondkleur veranderen met het statement COLOR dan kan dat, bijvoorbeeld:

```
color ,5
```

Een andere methode is:

vdp (7) = vdp (7) and 240 or 5

De AND-bewerking wordt eerst uitgevoerd, daarna de or-bewerking. De bewerking:

Bitnummer	7 6 5 4 3 2 1 0
VDP register 7	1 1 1 1 0 1 0 0
Masker (240)	1 1 1 1 0 0 0 0
VDP (7) AND 240	1 1 1 1 0 0 0 0
Binair 5	0 0 0 0 0 1 0 1
VDP (7) AND 240 OR 7	1 1 1 1 0 1 0 1

Dit laatste getal wordt door VDP (7) = in het VDP register 7 geplaatst.

Met het volgende programma kunt u het bitpatroon van elk karakter uit de karakterset op uw scherm krijgen. Het enige wat u moet doen is de ASCII code van dat karakter invoeren.

```
10 SCREEN 0
20 INPUT"ASCII-code";AS
30 'voor acht geheugenplaatsen
40 PRINT: FOR N=0 TO 7
50 'haal 1 regel v.h. patroon op
60 A=VPEEK(2048+AS*8+N)
70 'bepaal de binaire waarde
80 B$=BIN$(A)
90 'bepaal of de string 8 kar. telt
100 IF LEN(B$)=8 THEN 150
110 'bepaal het verschil in lengte
120 B=8-LEN(B$)
130 'vul de string aan met nullen
140 B$=STRING$(B,"0")+B$
150 'print 1 regel v.h. patroon
160 PRINT USING"###";A;
170 PRINT" ";B$
180 'voor de volgende regel
190 NEXT N: PRINT CHR$(11);
200 'wis de bovenste schermregel
210 PRINT SPACE$(15)CHR$(11);
220 'voor het volgende karakter
230 GOTO 20
```

In regel 60 wordt met 2048 + AS*8 het beginadres van het geheugenblokje van het gewenste karakter berekend. Door N van 0 tot en met 7 te veranderen (regel 40) wordt bij elke doorgang van de lus steeds een geheugenplaats gelezen en het resultaat daarvan op het scherm geprint.
Omdat daarbij N wordt verhoogd wordt steeds de naast hogere geheugenplaats gelezen. De binaire waarde van het gelezen getal wordt in regel 80 berekend. Omdat het bitpatroon makkelijker te herkennen is als de string steeds acht bits lang is wordt een kortere string met nullen aangevuld (regel 140). Na het printen van het bitpatroon moet bovenaan het scherm opnieuw de vraag "ASCII-code?" verschijnen maar het bitpatroon moet blijven staan. Dit is in dit programma als volgt opgelost:
Eerst gaat de cursor naar HOME met PRINT CHR$(11) op regel 190. Dan volgt in regel 210 SPACE$(X). Deze functie geeft een string van X spaties. In regel 210 wordt dus een string van 15 spaties geprint, waarmee de tekst op deze regel wordt gewist. Dan gaat opnieuw de cursor naar HOME en wordt in regel 230 teruggesprongen naar regel 20 waar het programma opnieuw begint. Het bitpatroon is nog niet van het scherm verdwenen.

12.5. Het zelf karakter maken

We hebben de ceremonie die de VDP moet volgen voor het vullen van het scherm al eens opgesomd. Eerst wordt in het schermgeheugen gekeken (naar elke plaats) welke ASCII code daar is ingeschreven, daarna wordt uit de karaktergenerator het bitpatroon van het karakter gehaald en rekening houdend met de kleurcode in het VDP-register 7, het karakter punt voor punt op het scherm geplaatst. Nu moet de VDP wel weten met welke geheugenplaatsen in de VIDEORAM het schermgeheugen en de karaktergenerator aanvangen. Deze gegevens en ook nog andere gegevens die met het scherm te maken hebben worden opgeslagen in zogenaamde BASIS-registers. Deze registers bevatten de diverse gegevens die de VDP nodig heeft voor de diverse schermmoden. Als de VDP een bepaalde schermmode is opgedragen zal hij zijn eigen (VDP-) registers steeds vullen uit de BASIS-registers met de gegevens die hij voor deze mode nodig heeft. In de 24 kolommode vindt hij in het BASIS-register 0 het aanvangsadres van het schermgeheugen (0) en in BASIS-register 2 het aanvangsadres van de patroongenerator (2048). Met de functie BASE (X) kunnen deze registers worden uitgelezen maar ook worden ingeschreven. Probeer maar:

print base (0); base (2)

Het schermgeheugen vinden we op de adressen 0 tot en met 959 van de VIDEORAM en de karaktergenerator op de adressen 2048 tot en met 4095.
De eerste vrije plaats boven de karaktergenerator is dus 4096. Nu is in de VIDEORAM nog ruimte genoeg om bijvoorbeeld de karaktergenerator op een andere plaats te zetten. We laten hem maar eens aanvangen met de geheugenplaats 4096:

base (2) = 4096

Voert u deze opdracht uit in de direct mode dan zult u beslist geen enkel karakter meer kunnen herkennen. Dat is logisch ook, want de VDP vindt in het nu aangewezen blok geen enkel bitpatroon wat hij nodig heeft om de karakters op de juiste manier te kunnen printen. We brengen het scherm daarom weer in de oude situatie met

base (2) = 2048

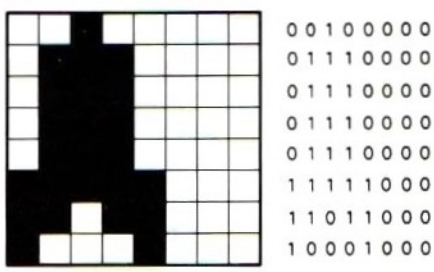
Dit zult u "blind" in moeten typen.
Met het volgende programma gaan we de gehele inhoud van de karaktergenerator overbrengen naar het volgende blok.

```
10 FOR N=2048 TO 4096
20 VPOKE N+2048, VPEEK(N)
30 NEXT
```

Dit programma heeft even zijn tijd nodig maar na afloop kunnen we weer proberen:

base (2) = 4096

Nu blijft het scherm geheel zo als het was en alle karakters zijn te herkennen. De VDP vindt dus weer de bitpatronen die hij nodig heeft.
De karaktergenerator kan overigens slechts met stappen van 2048 adressen worden verplaatst. In deze nieuwe karaktergenerator kunnen we de bitpatronen veranderen zoals we willen, zonder de oorspronkelijke bitpatronen kwijt te raken.
Voor het ontwerpen van karakters maken we gebruik van een 8×8 rooster. Nu hebben we nog al wat karakters ter beschikking in de karaktergenerator maar een raketje is daar nog niet bij. Dat gaan we dan maar eens maken. Zie hiervoor afb. 12. In het rooster gaan we de vakjes zwart maken die tezamen het figuur vormen. We houden daarbij rekening met de overlapping van de karakters. Nu gaan we op elke regel voor elk vakje dat wit is een 0 invoeren en voor elk vakje dat zwart is een 1. Er ontstaan zo binaire getallen die we met behulp van tabel 5 gaan omzetten in hexadecimale getallen. We kunnen in de MSX computer zowel decimale, hexadecimale, octale als binaire getallen invoeren. De binaire getallen zijn nogal lang en daarom kiezen we hexadecimale getallen omdat het omzetten van binair naar hexadecimaal met behulp van lijst 5 erg eenvoudig is. De binaire getallen worden in groepjes van vier bits verdeeld en elk groepje wordt vervangen door het hexadecimale getal dat in lijst 5 hiervoor is vermeld. De getallen zijn dan (van boven naar beneden):



afb. 12

20, 70, 70, 70, 70, F8, D8, 88

In deze volgorde, dus te beginnen met het getal dat gevormd is uit de bovenste regel van het rooster, worden ze in de computer gevoerd. Daarvoor gebruiken we het volgende programma:

```
10 CLS
20 FOR N=0 TO 7: READ A
30 VPOKE 4096+172*8+N,A
40 NEXT N
50 DATA &h20,&h70,&h70,&h70,&h70
60 DATA &hf8,&hd8,&h88
```

Elk karakter dat ingevoerd wordt moet een codegetal hebben. In dit geval is het codegetal 172. De getallen worden dan ingevoerd in het geheugenblokje dat begint op adres $4096 + 8*172 = 5472$. Als u nu de GRAPH toets gelijk met toets 1 indrukt dan ziet u het resultaat. PRINT CHR$(172) doet hetzelfde.
Probeer maar eens het volgende programma:

```
10 'inverse karakters
20 '97-120 ASCII voor kleine letters
30 FOR AS=97 TO 120: AD=4096+AS*8
40 'voor 8 stippenregels per karakter
50 FOR N=0 TO 7: AR=AD+N
60 'bepaal de inverse waarde
70 A=VPEEK(AR): AN=A XOR 255
80 'inverse karakters op de
90 'plaats van de hoofdletters
100 VPOKE AR-256,AN
110 NEXT N: NEXT AS
```

In plaats van hoofdletters krijgt u nu *inverse* letters op het scherm. Dit komt tot stand door AN = A XOR 255. Stel dat de binaire waarde van de inhoud van A 00110101 is. De XOR bewerking met 255 verloopt als volgt:

Bitnummer	7	6	5	4	3	2	1	0
A	0	0	1	1	0	1	0	1
255	1	1	1	1	1	1	1	1
A XOR 255	1	1	0	0	1	0	1	0

Bij de XOR bewerking worden de bits in het resultaat slechts dan 1 als de overeenkomstige bits van de getallen waartussen de bewerking plaats heeft, ongelijk zijn. Dit zijn de bits die in getal A 0 zijn. De bits die in getal A 1 zijn worden in het resultaat 0. Van de karakters die met het resultaat van deze bewerking worden gemaakt is dus de achtergrond voorgrond geworden en de voorgrond wordt achtergrond. De bitpatronen van de inverse karakters worden in regel 100 door AR-256 gebracht op de plaats van de hoofdletters (ASC("a")=97, ASC("A")=65, (97-65)*8=256). De logische bewerking NOT kan hier niet worden gebruikt. Probeer maar:

print bin$(not 2)

De inverse van het binaire getal 10 levert een zestien bits binair getal op en dat is wel wat groot voor een geheugenplaats die niet meer dan acht bits kan bevatten.

In deze paragraaf zijn we steeds uitgegaan van een zelf gedefinieerde karaktergenerator. De bitpatronen van de karakters in de originele karaktergenerator kunnen uiteraard op gelijke wijze worden veranderd.

12.6. Het verplaatsen van het schermgeheugen

Zoals we de plaats van de karaktergenerator zelf kunnen bepalen kunnen we ook het schermgeheugen op een andere plaats in het VIDEORAM situeren. Door steeds "over te schakelen" van het originele schermgeheugen naar het andere schermgeheugen kunt u leuke effecten bereiken. Het schermgeheugen is te verplaatsen in stappen van 1024 geheugenplaatsen. Tussen het originele schermgeheugen en de karaktergenerator op adres 2048 is nog plaats voor een tweede schermgeheugen dat aanvangt op adres 1024. Probeer eens:

base (0) = 1024

Ik denk niet dat u het een succes zult vinden. Het scherm is een grote warboel en met geen toets op het toetsenbord is daar verandering in te brengen. Dan maar weer terug naar de oorspronkelijke plaats:

base (0) = 0

Dit moet u weer blind intoetsen. Probeer nu:

base(0) = 1024: screen 0

Nu is alles ineens weer mogelijk. Alle verrichtingen die u in de "normale" situatie in de veertigkolommode kunt uitvoeren zijn ook nu mogelijk. Wilt u tijdens een programma omschakelen tussen twee schermgeheugens dan kunt u daarvoor de functie VDP(2) = A gebruiken. Op de plaats van A moet dan een getal worden ingevuld dat kan worden berekend met A = BAD/1024, waarin BAD het beginadres van het betreffende schermgeheugen is. Schakelen we om tussen de beginadressen 0 en 1024 dan moet het getal A gelijk zijn aan resp. 0 en 1.

```
10 'vul nieuw schermgeheugen met "A"
20 FOR X=1024 TO 2047
30 VPOKE X,65: NEXT X
40 'schakel naar nieuw schermgeheugen
50 VDP(2)=1
60 FOR X=0 TO 2000: NEXT
70 'schakel naar oorspr. schermgeheugen
80 VDP(2)=0
```

Eerst wordt in dit programma de geheugenruimte die voor het tweede schermgeheugen gebruikt zal worden gevuld met het karakter "A". Daarna wordt overgeschakeld. U krijgt de inhoud van het tweede schermgeheugen te zien zonder het SCREEN 0 te moeten gebruiken. De situatie wordt weer normaal (na een vertraging) door terug te schakelen met VDP(2) = 0.

13. Grafische functies, de tweeendertigkolommode

13.1. De indeling van het scherm

Bij de tweeendertigkolommode is voor elke karakter op het scherm een rooster beschikbaar van 8 bij 8 stippen. De tekens komen daardoor verder uit elkaar te staan terwijl de karakters die de hele breedte van het rooster nodig hebben ook geheel zichtbaar zijn. Deze mode is daardoor het meest geschikt om te worden gebruikt met de grafische tekens. Er gaan ook in deze mode 256 stippen op een regel zodat precies 256/8 = 32 karakters op een regel passen. Door WIDTH(32) wordt dus de gehelde schermbreedte gebruikt. De indeling van de VIDEORAM is bij deze mode als volgt:

```
    0 - 2047 patroongenerator.
 6144 - 6911 schermgeheugen.
 6912 - 7039 spritekenmerkgeheugen.
 8192 - 8223 kleurgeheugen.
14336 -      spritepatroongeheugen.
```

U ziet dat ten opzichte van de veertigkolommode er een aantal functies bij gekomen zijn.
Wat de indeling van het scherm betreft, ook nu correspondeert de eerste plaats van het schermgeheugen met de meest linkse plaats op de bovenste karakterregel. Verder is de nummering oplopend per regel van links naar rechts en van boven naar beneden. Om het adres van een geheugenplaats te vinden dat correspondeert met een bepaalde plaats op het scherm moeten we nu de formule

AD = BAD + X + 32*Y

toepassen. Ten eerste moet er voor elke regel 32 geheugenplaatsen worden opgeteld (32*Y) en ten tweede moet voor BAD, het adres van de eerste geheugenplaats van het schermgeheugen, 6144 worden genomen. Probeer eens

screen 1: vpoke 6511,2

Er verschijnt nu een *volle* maan op het midden van het scherm (6511 = 6144 + 15 + 11*32).
Zo u kunt zien is een bepaald gedeelte van de VIDEORAM gereserveerd als kleurgeheugen. Het VDP register 7 is nu alleen in gebruik voor het kader van het scherm. Dit kunt u vaststellen met

VDP (7) = 10

Uiteraard moet alles in dit hoofdstuk in de tweeendertigkolommode gebeuren!
Voor de kleuren van de karakters en hun achtergrond is het kleurgeheugen. Elke geheugenplaats in dit blok correspondeert met acht karakters. Het eerste geheugen met de karakters waarvan de ASCII-code 0 tot en met 7 is, het tweede met de karakters waarvan de ASCII-code 8 tot en met 15 is enz. De voorgrondkleur wordt bepaald door het getal in de hoge tetrade van de geheugenplaats, de vier hoogste bits dus. De achtergrondkleur wordt bepaald door het getal in de laagste tetrade.
Het volgende programma demonstreert de werking van het kleurgeheugen.

```
10 SCREEN 1
20 'voor karakters met ASCII-code<32
30 FOR X=0 TO 31
40 PRINT CHR$(1)CHR$(64+X)CHR$(32);
50 'voor karakters met hoge ASCII-get.
60 NEXT: FOR X=32 TO 255
70 PRINT CHR$(X)CHR$(32);: NEXT X
80 'voor steeds vier kleurgeheugens
90 FOR X=0 TO 31 STEP 4
100 'zwarte karakters, l.groene achtergr.
110 VPOKE 8192+X,16*1+3
120 'd.blauwe karakters op l.geel
130 VPOKE 8193+X,16*4+11
140 'd.rode karakters op wit
150 VPOKE 8194+X,16*6+15
160 '.d.groene karkters op l.rood
170 VPOKE 8195+X,16*12+9
180 NEXT
```

Eerst wordt de volledige karakterset op het scherm geprint, de karakters onderling gescheiden door een spatie (CHR$(32)). Daarna worden steeds vier opeenvolgende registers van het kleurgeheugen van gegevens voorzien. De voorgrondkleur wordt ingebracht door het codegetal van de kleur met 16 te vermenigvuldigen. De plaatswaarde van de hoge

tetrade in een geheugenplaats is namelijk gelijk aan 16. Het codegetal voor de achtergrondkleur kan hierbij worden opgeteld. Het programma bevat geen nieuwe statements en is eenvoudig te volgen. Als het u groen en geel voor de ogen wordt gebruik dan toets f6. Het statement COLOR geeft *alle* karakters dezelfde kleur en achtergrond.

13.2. Het verplaatsen van het kleurgeheugen

Ook in deze mode kan het schermgeheugen verplaatst worden zoals in de vorige paragraaf is beschreven. U zult dan wel moeten bepalden welk gedeelte van de VIDEORAM daarvoor vrij is. In het volgende programma wordt als tweede schermgeheugen de geheugenplaatsen 2048 tot en met 2815 gebruikt.

```
10 'schakel naar een nieuw geheugen
20 BASE(5)=2048: SCREEN 1
30 'vul het nieuwe schermgeheugen
40 FOR A=2048 TO 2815
50 VPOKE A,65: NEXT A
60 FOR X=0 TO 1000: NEXT X
70 'schakel terug naar orig. geheugen
80 BASE(5)=6144: SCREEN 1
```

Eerst wordt overgeschakeld met BASE (5) = 2048 en SCREEN 1. Dan ziet u hoe het scherm wordt gevuld met het karakter "A" waarna weer wordt teruggeschakeld. Met VDP(2) = 2 (2048/1024 = 2) of met VDP(2) = 6 (6144/1024 = 6) kunt u weer tijdens een programma tussen de geheugens omschakelen zonder SCREEN 1 te moeten gebruiken. Het VDP register nummer 2 dient steeds bij alle moden voor schermgeheugen.
Ook het verplaatsen van de karaktergenerator en het maken van eigen karakters gaat overeenkomstig met de methode van de vorige paragraaf.
In het volgende programma wordt getoond hoe dat werkt *in* een programma met behulp van het VDP register 4.

```
10 'vul het schermgeheugen met "A"
20 SCREEN 1: FOR X= 6144 TO 6911
30 VPOKE X,65: NEXT X
40 'breng karakter in 2e karktergen.
50 FOR X=0 TO 7: READ A
60 VPOKE 2048+65*8+X,A: NEXT X
70 DATA &h20,&h70,&h70,&h70
80 DATA &h70,&hF8,&hD8,&h88
90 'schakel tweede karaktergen. in
100 VDP(4)=1
110 FOR X=0 TO 2000: NEXT X
120 'schakel oorspr. karaktergen. in
130 VDP(4)=0: CLS
```

De karaktergenerator is in stappen van 2048 geheugenplaatsen te verplaatsen. Het getal dat in het VDP register 4 moet worden geschreven voor het aanwijzen van de plaats van de karaktergenerator wordt daarom gevonden met A = BAD/2048. In dit geval is BAD voor de tweede karaktergenerator 2048 zodat A = 1. Eerst wordt het gehele scherm weer gevuld met het karakter "A". Omdat slechts één soort karakter is gebruikt behoeft ook maar één bitpatroon in de tweede karaktergenerator te worden ingevoerd. Welk karakter is ingevoerd merkt u vanzelf als u het programma RUNT. Deze methode is ook weer bruikbaar voor de veertigkolommode en ook dan moet het VDP register 4 worden gebruikt. Uiteraard is ook de methode die in de vorige paragraaf is beschreven voor het verplaatsen van de karaktergenerator voor de tweeendertigkolommode geschikt.

Ook het kleurgeheugen is te verplaatsen. Dit gaat met stappen van 64 geheugenplaatsen tegelijk. Het basisregister 6 geeft de plaats van het kleurgeheugen:

print base (6)

Het volgende programma demonstreert het gebruik van het BASE register 6 en het VDP register 3 voor het verplaatsen van het schermgeheugen en het schakelen tussen twee geheugens in een programma.

```
10 'omschakelen naar nieuw kleurgeh.
20 BASE(6)=2048: SCREEN 1
30 'voorgrond geel, achtergrond zwart
40 FOR X=2048 TO 2079
50 VPOKE X,10*16+1: NEXT X
60 'voorgrond zwart, achtergrond geel
70 'invoeren in tweede kleurgeheugen
80 FOR X=8192 TO 8223
90 VPOKE X,1*16+10: NEXT X
100 LOCATE 11,12: PRINT"hallo"
110 'omschakelen tussen de geheugens
120 C=128
130 VDP(3)=C
140 'bepaal of een toets is ingedrukt
150 A$=INKEY$: IF A$<>"" THEN 210
160 'vertraag
170 FOR X=0 TO 500: NEXT X
180 'wissel van geheugen
190 IF C=128 THEN C=32 ELSE C=128
200 GOTO 130
210 'herstel oude situatie
220 BASE(6)=8192: SCREEN 1
```

Eerst wordt met BASE(6) = 2048: SCREEN 1 de geheugenruimte van 2048 tot en met 2097 als kleur-

geheugen aangewezen. Dit geheugen wordt gevuld met 10*16+1=161 voor de kleur van de voorgrond (geel) en de achtergrond (zwart).
Natuurlijk werkt VPOKE X,161 vlugger dan VPOKE X,10*16+1, omdat dan niet elke keer de waarde 161 behoeft te worden uitgerekend. Aan het invoeren van 10*16+1 is echter eenvoudiger te zien welke kleurcode in de hoge tetrade wordt geladen en welke in de lage tetrade van de geheugenplaats. Het oorspronkelijke kleurgeheugen gaan we gebruiken als tweede geheugen en laden daar de kleurcodes voor geïnverteerde tekens, voorgrondkleur zwart en achtergrondkleur geel.
Na het printen van "hallo" op het scherm volgt het omschakelen tussen de twee kleurgeheugens waarbij in elke cyclus wordt nagegaan of een toets is ingedrukt. Is dat gebeurd dan wordt de oude situatie weer hersteld met BASE(6)=8192: SCREEN 1. De inhoud voor het VDP register wordt berekend met 2048/64=32 en 8192/64=128.

In het algemeen geldt voor het veranderen van een beginadres van een geheugenblok:

base(n)=BAD: screen m.

Voor het schermgeheugen is BAD een meervoud van 1024.
Voor een karaktergenerator is BAD een meervoud van 2048.
Voor een kleurgeheugen is BAD een meervoud van 64.

Voor het in een programma omschakelen tussen geheugenblokken wordt gebruikt:

VDP(r)=G

Voor een schermgeheugen:
r=2 en G=BAD/1024.
Voor een karaktergenerator:
r=4 en G=BAD/2048.
Voor een kleurgeheugen: r=3 en G=BAD/64.
De maximumwaarde van G is 255.

13.3. Het gebruik van sprites

Een sprite is een figuur dat zich over het scherm kan verplaatsen zonder dat daardoor de afbeelding op het scherm wordt verstoord. Het is als een ondoorzichtige figuur dat zich voor het scherm langs schijnt te bewegen. Als in het figuur een opening is dan ziet u door de opening de afbeelding (voor- en achtergrond) ter plaatse van het veld.
Voor het invoeren van een sprite zijn de volgende gegevens nodig:

a. Het bitpatroon van de sprite.
   Dit bitpatroon kan het formaat 8×8 maar ook 16×16 hebben.
b. Het spritenummer.
   Dit kan een nummer van 0 tot en met 31 zijn.
c. De plaats van de sprite.
d. De kleur van de sprite.
   Hiervoor kan één van de vijftien kleuren uit lijst 1 worden gekozen, plus "transparant".

Het bitpatroon voor een sprite met het formaat van 8×8 wordt op gelijke wijze uit een tekening afgelezen als bij de eigengemaakte karakters is gedemonstreerd. De hieruit ontstane getallen (bytes) worden ook in dezelfde volgorde ingevoerd. Het wordt ingevoerd met de functie SPRITE$(n).
Nemen we als voorbeeld de raket dan wordt het patroon hiervoor ingevoerd met:

```
sprite$(n)=chr$(&h20)+chr$(&h70)+chr$(&h70)
+chr$(&h70)+chr$(&h70)+chr$(&hF8)+chr$
(&hD8)+chr$(&h88)
```

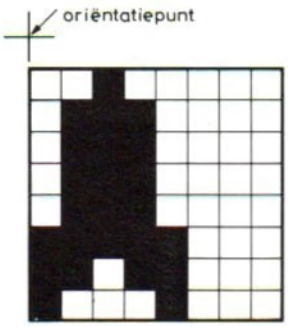
Andere methoden van invoeren worden in de programmavoorbeelden gegeven.
Het nummer n in SPRITE$(n) geeft het spritepatroonnummer aan. Dit heeft een andere betekenis dan het spritenummer. Een bepaalde sprite (met een bepaald spritenummer dus) kan van meerdere spritepatronen gebruik maken (slechts één tegelijk).
Voor het invoeren van een bitpatroon van een sprite met een 16×16 formaat wordt het rooster in twee kolommen van acht stippen breed verdeeld. Elke kolom levert zestien binaire getallen van acht bits op. Deze worden in de volgorde linker kolom - rechter kolom van boven naar beneden ingevoerd.

De andere gegevens worden met het PUT SPRITE statement ingevoerd.

put sprite n, (X,Y),c,m

Hierin is n het spritenummer en kan een getal van 0 tot en met 31 zijn.

X en Y zijn de coördinaten die de plaats van de sprite op het scherm aangeven. De X-waarde kan een getal van -32 tot en met 255 zijn. Hierbij is $X=0$ de meest linkse stip op een stippenregel van het scherm en $X=255$ de meest rechtse stip op de regel (er zijn 256 stippen op een regel). De Y-waarde kan een getal zijn van -17 tot en met 191. Er zijn 192 stippenregels, genummerd van 0 tot en met 191. Het oriëntatiepunt van de sprite bevindt zich precies één stippenregel boven de linkerbovenhoek (afb. 13). Bij $Y=-1$ is de bovenkant van de sprite gelijk met de bovenkant van het veld en bij $X=0$ is de linkerkant van de sprite gelijk met de linkerkant van het scherm.



afb. 13

Achter PUT SPRITE is c het kleurnummer uit de lijst 1. Dit kleurnummer bepaalt in welke kleur de enen in het spritepatroon zullen oplichten. Voor de nullen wordt *geen* (achtergrond-) kleur ingevoerd. De achtergrond van de sprite (de nullen) is steeds doorzichtig. De m achter PUT SPRITE is het nummer van het spritepatroon. Dit nummer hoeft dus niet gelijk te zijn aan het spritenummer.
Het volgende programma laat u zien hoe een sprite zich over het veld verplaatst.

```
10 SCREEN 1: COLOR 1,1
20 'invoeren bitpatroon sprite
30 A$=CHR$(&H20)+CHR$(&H70)+CHR$(&H70)
40 B$=CHR$(&H70)+CHR$(&H70)+CHR$(&HF8)
50 C$=CHR$(&HD8)+CHR$(&H88)
60 SPRITE$(0)=A$+B$+C$
70 'begin co-ordinaten
80 DY=1: X=123: Y=-9
90 'teken de sprite
100 PUT SPRITE 0,(X,Y),14,0
110 'vertraag
120 FOR V=0 TO 50: NEXT
130 'bereken nieuwe co-ordinaten
140 Y=Y+DY
150 'bepaal of uiterste waarde is bereikt
160 IF Y=-9 OR Y=191 THEN DY=-DY
170 'sprite naar de nieuwe plaats
180 GOTO 90
```

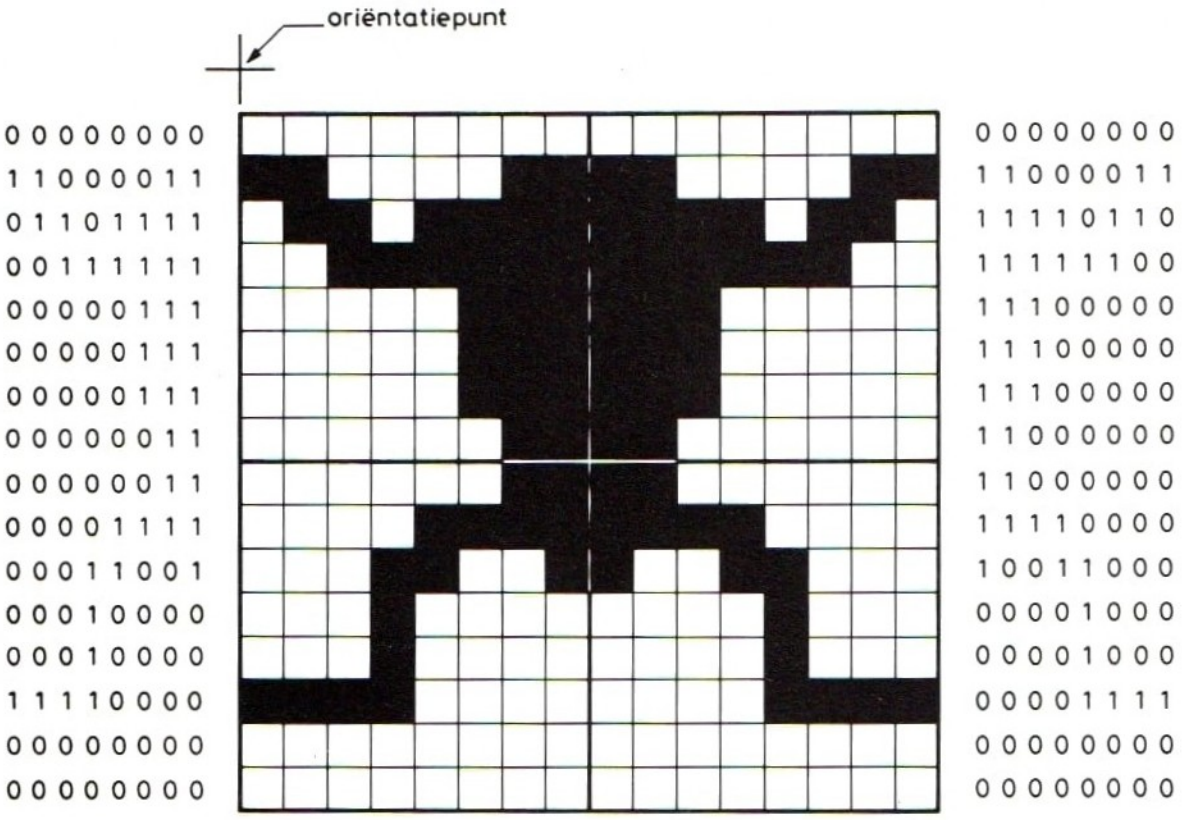
Voor het invoeren van het bitpatroon van de sprite zijn in de regels 20 tot en met 30 de variabele A$, B$ en C$ gevormd uit de gegevens voor dat patroon (de raket). Deze worden in regel 50 met patroonnummer 0 ingevoerd. In regel 70 worden de begincoördinaten van de sprite op $X=123$ (ongeveer het midden van een regel) en op $Y=-9$ gesteld. Het is een 8×8 sprite en dus staat hij bij aanvang boven het scherm. Om de sprite niet te snel te laten gaan is een vertraging ingevoerd waarna de nieuwe Y-coördinaat wordt berekend. De X-waarde verandert niet zodat de sprite zich na RUN verticaal over het veld gaat bewegen. We zien hem boven, achter het kader vandaan, op het scherm komen en later onderaan weer van het scherm verdwijnen. Door $DY=-DY$ in regel 150 gaat de sprite daarna weer omhoog. De snelheid van de sprite is te regelen in regel 120.
Door niet de Y- maar de X-waarde te veranderen kunnen we de sprite horizontaal heen en weer laten bewegen. Verander daarvoor het programma met de volgende regels:

```
80 DX=1: Y=95: X=-8
140 X=X+DX
160 IF X=-8 OR X=255 THEN DX=-DX
```

U kunt zien dat bij $X=255$ de sprite nog net niet helemaal aan de rechterkant van het scherm is verdwenen.
De afmetingen van de spite kunnen twee keer zo groot worden gemaakt. Dat betekent nog niet dat u dan een sprite krijgt van 16×16 stippen, maar wel een sprite die dezelfde grootte heeft. Hij is echter opgebouwd uit 8×8 stippen. Hiervoor moet u de volgende regel invoeren:

```
10 SCREEN 1,1: COLOR 1,1
```

Het commando SCREEN 1,1 is hiervoor verantwoordelijk. De eerste 1 bepaalt de schermmode (32 kolommen). De tweede 1 bepaalt dat de sprites van het *vergrote* 8×8 formaat zijn. Dit geldt dan wel voor *alle* sprites die op het scherm zijn. SCREEN 1,0 is het commando voor de normale 8×8 sprite grootte. Wilt u sprites op het scherm met het formaat 16×16 stippen dan moet u het commando SCREEN 1,2 invoeren.
Het volgende programma toont twee sprites op het scherm van het formaat 16×16. Aan één van de twee is dat niet te zien. Dit is de raket. Deze is op



afb. 14

normale grootte afgebeeld op het 8×8 veld aan de bovenkant van de linker kolom van het 16×16 rooster van het spritepatroon 0. De rest van dit patroon is gevuld met nullen. De afbeelding van het spritepatroon 1 is een kikker, althans had daar op moeten lijken. In afb. 14 is de sprite getekend met daarnaast de binaire getallen die uit het figuur volgen. De getallen die links naast het figuur staan worden het eerst ingevoerd, daarna die rechts naast het figuur staan. Ook hier is het oriëntatiepunt één puntenrij boven de linkerbovenhoek.

```
10 'sprite 16x16 stippen
20 SCREEN 1,2: COLOR 1,1
30 'invoeren bitpatroon sprite
40 'voor sprite$(0)
50 FOR V=0 TO 31: READ A
60 A$=A$+CHR$(A): NEXT V
70 'voor sprite$(1)
80 FOR V=0 TO 31: READ B
90 B$=B$+CHR$(B): NEXT V
100 SPRITE$(0)=A$: SPRITE$(1)=B$
110 'begin co-ordinaten
120 DY(0)=-1: X(0)=128: Y(0)=191
130 DY(1)=1: X(1)=123: Y(1)=-9
140 'teken de sprite
150 PUT SPRITE 0,(X(0),Y(0)),14,0
160 PUT SPRITE 1,(X(1),Y(1)),12,1
170 'vertraag
180 FOR V=0 TO 10: NEXT V
190 'bereken nieuwe co-ordinaten
200 Y(0)=Y(0)+DY(0): Y(1)=Y(1)+DY(1)
210 'bepaal of uiterste waarde is bereikt
220 FOR A=0 TO 1
230 IF Y(A)=-17 OR Y(A)=191 THEN 270
240 NEXT
250 'sprite naar de nieuwe plaats
260 GOTO 140
270 DY(A)=-DY(A): GOTO 240
280 'spritepatroon 0
290 DATA &h20,&h70,&h70,&h70
300 DATA &h70,&hF8,&hD8,&h88
310 DATA 0,0,0,0,0,0,0,0
320 DATA 0,0,0,0,0,0,0,0
330 DATA 0,0,0,0,0,0,0,0
340 'spritepatroon 1
350 DATA &h00,&hC3,&h6F,&h3F
360 DATA &h07,&h07,&h07,&h03
370 DATA &h03,&h0F,&h19,&h10
380 DATA &h10,&hF0,&h00,&h00
390 DATA &h00,&hC3,&hF6,&hFC
400 DATA &hE0,&hE0,&hE0,&hC0
410 DATA &hC0,&hF0,&h98,&h08
420 DATA &h08,&h0F,&h00,&h00
```

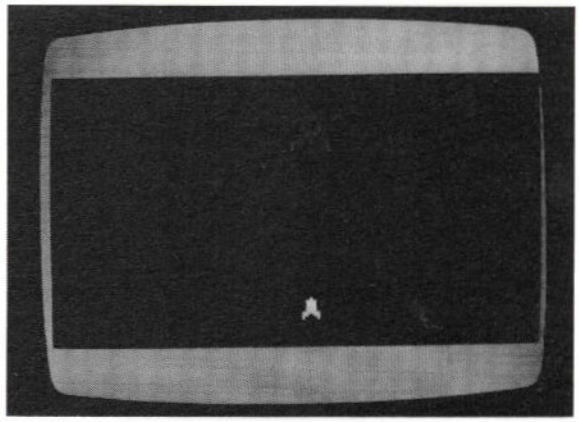
Voor het invoeren van het spritepatroon 0 voor de raket wordt in regel 60 de variabele A$ samengesteld uit de data in de regel 290 tot en met 330. Het READ statement in regel 50 leest deze data.

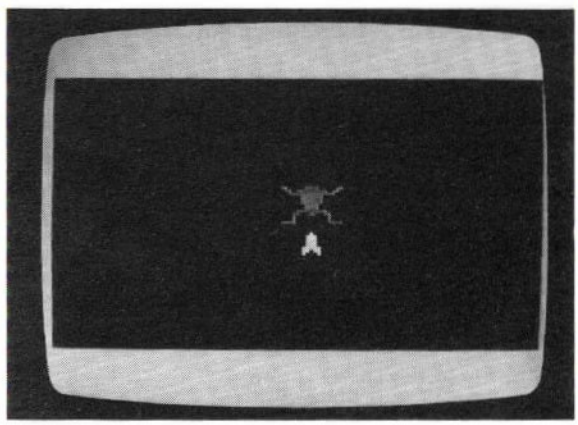
Op dezelfde manier wordt het spritepatroon 1 voor de kikker ingevoerd. Eén sprite begint onderaan het scherm (regel 120, Y(0) = 191) en de andere bovenaan het scherm (regel 130)). Regel 150 tekent de sprite met patroon 0 (laatste nummer achter PUT SPRITE onder aan het scherm (X(0),Y(0)). De Y-waarden worden op de gebruikelijk manier veranderd zodat de sprites verticaal over het scherm bewegen. Als de sprites elkaar passeren ziet u dat de raket voor de kikker langs gaat. Dit komt omdat de raket met spritenummer 0 een hogere prioriteit heeft dan de kikker met spritenummer 1. De sprite met het laagste spritenummer heeft steeds de hoogste prioriteit en zal voor een sprite langs gaan die een hoger spritenummer heeft. Als we de spritenummers verwisselen zult u zien dat de kikker voor de raket langs gaat.

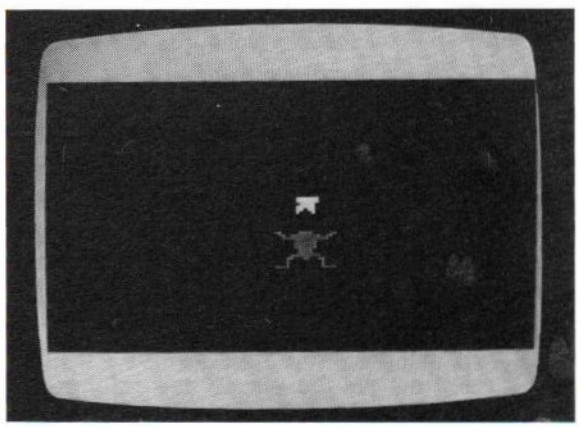
```
150 PUT SPRITE 1,(X(0),Y(0)),14,0
160 PUT SPRITE 0,(X(1),Y(1)),12,1
```

De computer kan vaststellen of de sprites elkaar raken. Hiertoe dient het statement ON SPRITE GOSUB dat in dezelfde serie thuishoort als ON INTERVAL GOSUB enz. Zodra de sprites elkaar raken (de enen in het patroon) wordt het programma onderbroken (interrupt) en naar een subroutine gesprongen. We voeren daarvoor de volgende regel in *(geen* NEW, de regels 150 en 160 worden gewijzigd):

```
22 'voor het raken van de sprites
24 ON SPRITE GOSUB 500
26 SPRITE ON
92 'voor sprite$(2)
94 FOR V=0 TO 31: READ C
96 C$=C$+CHR$(C): NEXT V: S=0
105 SPRITE$(2)=C$
150 PUT SPRITE 1,(X(0),Y(0)),14,S
160 PUT SPRITE 0,(X(1),Y(1)),12,1
430 'spritepatroon 2
440 DATA &h00,&h00,&hFC,&hFC
450 DATA &h78,&hF8,&hD8,&h88
460 DATA 0,0,0,0,0,0,0,0
470 DATA 0,0,0,0,0,0,0,0
480 DATA 0,0,0,0,0,0,0,0
500 SPRITE OFF: S=2: BEEP
510 PUT SPRITE 0,(X(0),Y(0)),14,2
520 RETURN
```







Omdat een botsing tussen een kikker en een raket wel enig effect zal hebben moet een derde spritepatroon worden ingevoerd. Dit heeft in de regels 92 tot en met 96 plaats. In de regels 24 en 26 zien we de regels die nodig zijn om te reageren op het botsen van twee sprites op het scherm. Een dergelijke botsing doet het programma naar de subrou-

tine op regel 500 springen (ON SPRITE GOSUB 500). Hier wordt met SPRITE OFF de werking uitgeschakeld. Deze opdracht moet steeds aan het begin van een dergelijke subroutine staan. Daarna wordt de variabele S=2, wordt een BEEP geluid gegeven en sprite 0 (de raket) met het spritepatroon 2 getekend. Nu kan de werking eventueel weer worden ingeschakeld met SPRITE ON. Omdat verdere botsingen geen effect meer zullen hebben is dat achterwege gelaten. Met RETURN komen we terug bij het hoofdprogramma waar het onderbroken was door de interrupt.
Terug in het hoofdprogramma wordt sprite 0 verder steeds met spritepatroon 2 getekend. Hiervoor is S ingevoerd (regel 150) die eerst 0 was (regel 96) maar in de subroutine 2 is geworden.

In dit programma is voor het tekenen van een bepaalde sprite van spritepatroon gewisseld. Dit is ook nodig als we een sprite bewegingen willen laten maken, zoals de kikker zijn poten moet bewegen om te kunnen zwemmen. In het volgende programma zwemt de kikker alleen met zijn achterpoten. Dit wordt gedaan door regelmatig na elkaar van spritepatroon te wisselen.

```
10 'sprite 16x16 stippen
20 SCREEN 1,2: COLOR 1,1
30 'invoeren bitpatroon sprite
40 'voor sprite$(0)
50 FOR V=0 TO 31: READ A
60 A$=A$+CHR$(A): NEXT V
70 SPRITE$(0)=A$
80 'voor sprite$(1)
90 RESTORE: GOSUB 310
100 RESTORE 460: GOSUB 310
110 RESTORE 400: GOSUB 310
120 RESTORE 490: GOSUB 310
130 SPRITE$(1)=G$: G$=""
140 'voor sprite$(1)
150 RESTORE: GOSUB 310
160 RESTORE 520: GOSUB 310
170 RESTORE 400: GOSUB 310
180 RESTORE 550:GOSUB 310
190 SPRITE$(2)=G$
200 'co-ordinaten bij aanvang
210 X=128: Y=191
220 'voor drie figuren
230 FOR S=0 TO 2
240 'teken de sprite
250 PUT SPRITE 0,(X,Y),12,S
260 'vertraag
270 FOR V=0 TO 10: NEXT V
280 'sprite boven aan het veld, opnieuw
290 Y=Y-1: IF Y=-17 THEN 200
300 NEXT S: GOTO 220
310 'subr. vormt string van DATA
320 FOR V=0 TO 7: READ G
330 G$=G$+CHR$(G): NEXT V: RETURN
340 'data bovenlijf links
350 DATA &h00,&hC3,&h6F,&h3F
360 DATA &h07,&h07,&h07,&h03
370 'data onderlijf links 0
380 DATA &h1F,&h17,&h11,&hB0
390 DATA &h60,&h00,&h00,&h00
400 'data bovenlijf rechts
410 DATA &h00,&hC3,&hF6,&hFC
420 DATA &hE0,&hE0,&hE0,&hC0
430 'data onderlijf rechts 0
440 DATA &hF8,&hE8,&h88,&h0D
450 DATA &h06,&h00,&h00,&h00
460 'data onderlijf links 1
470 DATA &h03,&h0F,&h19,&h10
480 DATA &h10,&hF0,&h00,&h00
490 'data onderlijf rechts 1
500 DATA &hC0,&hF0,&h98,&h08
510 DATA &h08,&h0F,&h00,&h00
520 'data onderlijf links 2
530 DATA &h03,&h0F,&h19,&h10
540 DATA &h10,&h10,&h30,&hE0
550 'data onderlijf rechts 2
560 DATA &hC0,&hF0,&h98,&h08
570 DATA &h08,&h08,&h0C,&h07
```

Voor dit programma moeten drie spritepatronen voor de kikker worden ingevoerd. Omdat het bovenlichaam bij alle drie de patronen hetzelfde is wordt hiervoor slechts één keer de DATA ingevoerd. Door de regels 50 en 60 wordt eerst op de normale wijze het spritepatroon 0 ingevoerd door de DATA regels 350 tot en met 450 te lezen. Nu wordt voor spritepatroon 1 eerst met RESTORE er voor gezorgd dat opnieuw met lezen zal worden begonnen met DATA regel 350. In de subroutine 310 wordt acht keer DATA gelezen. Dan volgt RESTORE 460. Dat heeft tot gevolg dat nu door de subroutine de DATA op de regels 470 en 480 wordt gelezen. Opnieuw volgt een RESTORE, nu voor het lezen van de DATA op de regels 410 en 420. Dit gaat zo door tot alle DATA die nodig is voor het spritepatroon 1, is gelezen. Spritepatroon 2 wordt op dezelfde manier samengesteld. Door regel 250 worden de drie patronen steeds na elkaar getekend omdat in regel 230 S achterelkaar 0, 1 en 2 wordt. De rest van het programma is slechts een herhaling van voorgaande programmadelen.

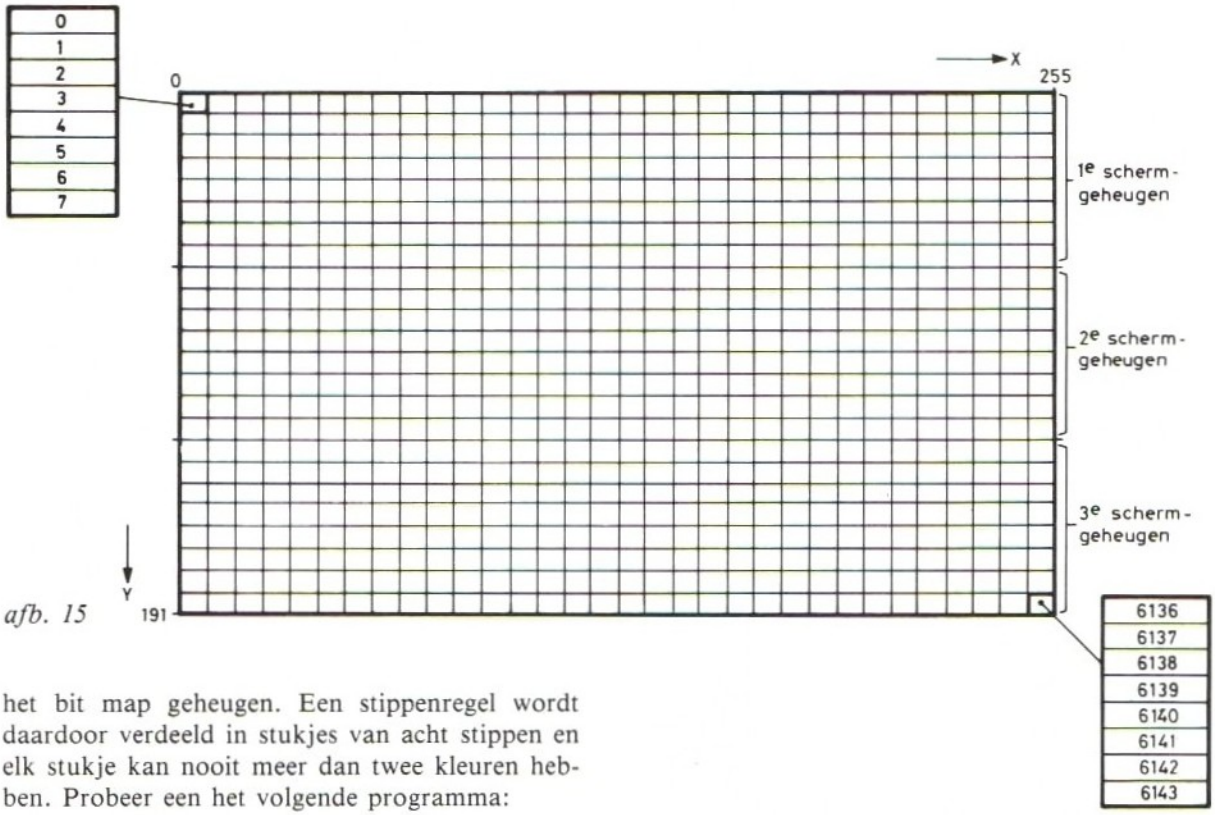
14. Grafische functies, hoge resolutie

14.1. De indeling van het scherm

Tot nu toe zijn we gewend geweest het scherm te beschrijven met karakters die zijn geplaatst in velden van 6×8 stippen of 8×8 stippen. Zelfs een sprite is in principe als karakter te beschouwen. Hoge resolutie wil echter zeggen dat elke stip op het scherm apart "aan" of "uit" kan worden gezet (de voorgrond- of achtergrondkleur kan aannemen). In principe hebben we ons daar al mee bezig gehouden bij het maken van karakters. In dat geval kunnen we ook bepalen welke stip op het scherm zal oplichten en welke niet, omdat we het karakter zelf kunnen vormen en ook zelf de plaats van het karakter op het scherm kunnen bepalen. Dit principe wordt ook toegepast bij de hoge resolutie. Deze mode kan worden ingeschakeld met SCREEN 2, maar kan alleen in de programmode worden gebruikt. Het principe van deze grafische mode is als volgt: plaats in het schermgeheugen in de eerste geheugenplaats een 0, in de tweede geheugenplaats een 1, in de derde een 2 enz. Nu correspondeert de eerste karakterplaats op uw scherm (het blokje van 8×8 stippen geheel links bovenaan) met de eerste acht bytes van de karaktergenerator, de karakterplaats daarnaast met de tweede acht bytes van de karaktergenerator enz. Als we nu maar de juiste bits in de karaktergenerator 1 maken dan kunnen we elk figuur op het scherm tekenen, van welke vorm dan ook. Dat betekent dat we nu niet het schermgeheugen moeten inschrijven (dat is gevuld met 0, 1, 2 enz.) maar de karaktergenerator. De karaktergenerator wordt daarom wel het "bit map geheugen" genoemd.
Nu zijn er $24*32=768$ karakterplaatsen van 8×8 stippen op het scherm. De geheugenplaatsen van het schermgeheugen kunnen geen grotere getallen dan 255 bevatten zodat we slechts een derde van het schermgeheugen kunnen inschrijven. Er is daarom niet één schermgeheugen maar er zijn er drie, die ook weer corresponderen met drie karaktergeneratoren. Het eerste schermgeheugen is voor de bovenste acht karakterregels ($8*32=256$), het tweede voor de middelste acht karakterregels en het derde voor de onderste acht karakterregels. De eerste patroongenerator vinden we op de geheugenplaatsen 0 tot en met 2047, de tweede van 2048 tot en met 4095 en de derde van 4096 tot en met 6143. De drie karaktergeneratoren sluiten dus elkaar aan en vormen zo een bit map geheugen op de geheugenplaatsen 0 tot en met 6143, waarvan elk blokje van acht geheugenplaatsen correspondeert met een blokje van acht bij acht stippen op het scherm (afb. 15). Nu is er een vrij uitgebreide berekening nodig om te weten welk getal (byte) in een geheugenplaats van het bit map geheugen moet worden geschreven (met VPOKE) om eem bepaalde stip op het scherm te doen oplichten. Bij de MSX computer hoeven we ons daar geen zorgen om te maken. We kunnen met een eenvoudig statement elke willekeurige stip op het scherm laten oplichten, het statement PSET.
De indeling van het VIDEORAM is als volgt:

0 - 6143	bit map geheugen.
6144 - 6911	schermgeheugen.
6912 - 7039	sprite kenmerkgeheugen.
8192 - 14335	kleurgeheugen.
14336 -	spritepatroongeheugen.

Behalve dat u in deze mode ook met sprites kunt werken heeft u ook een nogal uitgebreid kleurgeheugen ter beschikking. Dit kleurgeheugen bevat evenveel geheugenplaatsen als het bit map geheugen zodat elke geheugenplaats in het kleurgeheugen de kleur bepaalt van acht stippen (naast elkaar) op het scherm. Het codegetal voor de voorgrondkleur wordt in de hoge tetrade en die voor de achtergrondkleur in de lage tetrade geladen. De kleur van de acht stippen die corresponderen met de eerste geheugenplaats in het bit map geheugen (0) wordt bepaald door het getal in geheugenplaats 8192. De kleur van de acht stippen daar onder door het getal in geheugenplaats 8193 enz., tot het gehele 8×8 rooster is afgewerkt. De volgende acht geheugenplaatsen in het kleurgeheugen zijn voor het tweede rooster, zodat het kleurgeheugen dezelfde relatie met de stippen op het scherm heeft als



afb. 15

het bit map geheugen. Een stippenregel wordt daardoor verdeeld in stukjes van acht stippen en elk stukje kan nooit meer dan twee kleuren hebben. Probeer een het volgende programma:

```
10 SCREEN 2
20 'voor alle plaatsen v.h. kleurgeh.
30 FOR X=8192 TO 14335
40 'achtergrondkleur geel
50 VPOKE X,10:NEXT
60 GOTO 60
```

U ziet dat dit programma karakter voor karakter en regel voor regel het scherm vult met de achtergrondkleur geel, door het getal 10 te schrijven in de lage tetrade van de geheugenplaatsen.
De regel 60 is nodig om in de hoge resolutiemode te blijven.
Het zou wat lang duren om op deze manier bij elk programma de achtergrondkleur in te voeren. Daar is een veel vluggere methode voor.
De volgende basisregisters bevatten de aanvangsadressen voor de geheugenblokken bij de hoge resolutiemode:

BASE(10) schermgeheugen.
BASE(11) kleurgeheugen.
BASE(12) bit map geheugen.
BASE(13) sprite kenmerkgeheugen.
BASE(14) spritepatroongeheugen.

RUN het volgende programma:

```
10 COLOR 1,10: SCREEN 2
20 FOR X=0 TO 255
30 'geef een stip de voorgrondkleur (1)
40 PSET(X,0),1: FOR V=0 TO 25: NEXT V
50 'wis de stip
60 PRESET(X,0): NEXT X
70 GOTO 70
```

U ziet nu een zwarte stip langs de bovenrand van het scherm gaan. Deze stip wordt op het scherm geplaatst door de opdracht PSET(X,Y),c. De kleur is zwart omdat voor c het getal 1 is ingevoerd. Voor Y is 0 ingevoerd en dat is kennelijk voor de bovenste stippenregel. De plaats op die regel wordt bepaald door de waarde van X. Deze kan een waarde hebben van 0 tot en met 255 en neemt toe van links naar rechts. De gele achtergrond van het veld wordt verkregen door het invoeren van COLOR 1,10 voor het screen 2. Dat betekent dan wel dat bij terugkeer in de karaktermode het scherm geel zal zijn met zwarte tekens (door de 1 in COLOR 1,10). De opdracht PRESET wist de stip weer nadat hij even zichtbaar is geweest (vertraging).

RUN nu het volgende programma:

```
10 COLOR 1,10: SCREEN 2
20 FOR Y=0 TO 191
30 'geef een stip de voorgrondkleur (1)
40 PSET(0,Y),1: FOR V=0 TO 25: NEXT V
50 'wis de stip
60 PRESET(0,Y): NEXT Y
70 GOTO 70
```

U zal nu de stip langs de linkerrand van het veld verticaal naar beneden zien bewegen. Het nummer van de stippenregel wordt weergegeven door de Y-waarde (van 0 tot en mar 191) en neemt toe van boven naar beneden. Hieruit volgt dat de plaats van een stip op het scherm kan worden vermeld met de X- en de Y-coördinaat.
Op deze manier kan met PSET ook een lijn getekend worden. Verwijder hiervoor uit regel 60 PRESET(0,Y) eens en RUN het programma dan opnieuw.
U kunt in deze mode sprites gebruiken door het invoeren van SCREEN 2,n waarin n 0 is voor 8×8 sprites van normale grootte, 1 is voor 8×8 sprites van dubbele grootte, 2 is voor 16×16 sprites van normale grootte en 3 is voor 16×16 sprites van dubbele grootte.

14.2. Het tekenen van rechte lijnen en cirkels

Het tekenen van rechte lijnen is een zeer eenvoudige zaak met het statement LINE
(X0,Y0)–(X1,Y1),c. Dit statement is in een programma in te voeren met de begincoördinaten X0,Y0, de eindcoördinaten X1,Y1 en de kleur c van de lijn. We gaan eerst een horizontale en een verticale rechte lijn proberen. Gebruik voor u het programma invoert de toets F6.

```
10 SCREEN 2
20 LINE(1,95)-(200,95),15
30 LINE(3,20)-(3,150),1
40 GOTO 40
```

Eerst wordt de horizontale lijn getekend (regel 20). Deze lijn begint niet geheel vooraan het scherm (X0 = 1) en de kleur is wit.
De verticale lijn wordt daarna getekend. U ziet dat we met het tekenen van verschillend gekleurde lijnen voorzichtig moeten zijn. Een klein stukje van de horizontale lijn is zwart. Dat komt omdat het eerste stukje van de stippenregel waarop de lijn is getekend drie kleuren zou moeten hebben, wit voor de horizontale lijn, zwart voor de verticale lijn (deze wordt later getekend en loopt daarom "voor" de witte lijn langs), terwijl de eerste stip van de regel de achtergrondkleur blauw moet hebben. De laatst getekende lijn heeft de prioriteit zodat de eerste stip van de regel blauw en de andere stippen zwart worden. Verander regel 20 in

```
20 LINE(0,95)-(200,95),15
```

Hiermee wordt de fout voorkomen omdat nu de acht stippen op de regel slechts de kleuren wit en zwart moeten aannemen.
Met het statement LINE kunnen ook rechthoeken op het scherm worden getekend. De ingevoerde coördinaten zijn dan twee tegenoverelkaar liggende hoekpunten van de rechthoek. Dit wordt bereikt door achter het kleurnummer de letter b te plaatsen. Het is ook mogelijk om achter het kleurnummer de letters bf te plaatsen. De rechthoek die dan wordt getekend wordt geheel opgevuld in de kleur die met het kleurnummer is aangegeven.

```
10 SCREEN 2
20 LINE (48,50)-(128,100),15,B
40 LINE (168,30)-(216,190),10,BF
60 GOTO 60
```

Dit programma tekent twee rechthoeken, de een alleen de omtrek en de ander geheel opgevuld. Als u uit de gegevens in het programma niet kunt zien welke hoekpunten als coördinaten zijn opgegeven voer dan de volgende regels in.

```
30 LINE (48,50)-(128,100),6
50 LINE (168,30)-(216,190),1
```

De lijnen worden getekend met dezelfde begin- en eindcoördinaten als de hoekpunten van de bijbehorende rechthoek en in een contrasterende kleur. U ziet dat in de hoekpunten drie kleuren bij elkaar komen. Moeilijkheden daarmee zijn voorkomen door de waarde voor de X-coördinaten een meervoud van 8 te laten zijn.
Ook voor het tekenen van cirkels en ellipsen is een statement, CIRCLE.
Als gegevens hiervoor moeten ten minste de coördinaten van het middelpunt en de straal worden ingevoerd. Dan kan ook een kleurnummer volgen. Probeer het volgende programma:

```
10 SCREEN 2
20 CIRCLE (127,95),50,10
30 GOTO 30
```

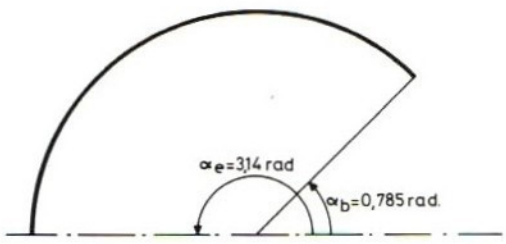
Dit programma had een cirkel in het midden van uw scherm moeten tekenen, maar dat is een ellips geworden. De reden daarvan is dat de stippen op het scherm in de breedte een grotere afmeting hebben dan in de hoogte. Er kan daarvoor een vermenigvuldigingsfactor worden ingevoerd. Deze bedraagt 1.4. Ook bij het tekenen van vierkanten moet voor de lengte van de zijden met deze factor rekening worden gehouden. Verander regel 20 in:

```
20 CIRCLE (127,95),50,10,,,1.4
```

U ziet nu een redelijke cirkel op scherm (hoe is het met de afregeling van uw tv?, die moet natuurlijk goed zijn!). De middellijn is gemeten in punten, horizontaal 100 punten en verticaal 140 punten. Het aantal komma's tussen de vermenigvuldigingsfactor en het kleurgetal is nodig omdat tussen deze getallen nog twee getallen kunnen worden ingevoerd, die hier zijn weggelaten. Met deze getallen is het mogelijk om een cirkelsegment te laten tekenen.

```
20 CIRCLE (127,95),50,10,.785,3.14,1.4
```

Nu heeft u als eerste getal achter het kleurnummer de beginhoek ingevuld. Deze is 0.785 rad (45°). In afb. 16 is de beginhoek aangegeven. Als tweede getal heeft u de eindhoek ingevuld (3.14 rad., 180°). Als laatste volgt weer het verhoudingsgetal 1.4. Als u dit getal anders neemt dan 1.4 dan wordt een ellips getekend:



afb. 16

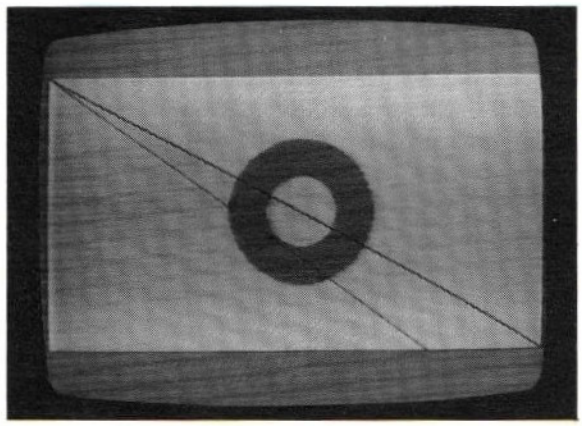
```
10 SCREEN 2
20 CIRCLE (127,95),50,10,,,3
30 PAINT (127,95),10
40 GOTO 40
```

Dit laatste is in voorgaand programma gebeurd. Er is ook nog een nieuw statement gebruikt: PAINT. Door de coördinaten op te geven van een punt in de ellips (in dit geval het middelpunt) met daar achter het kleurnummer dat het zelfde is als dat waarmee de ellips is getekend, wordt de ellips opgevuld met deze kleur. Dit werkt overigens bij elk figuur dat geheel omsloten is met lijnen van dezelfde kleur. Probeer ook eens het programma met in regel 30 de coördinaten van een punt buiten de ellips:

```
30 PAINT (190,95),10
```

Het volgende programma toont u twee concentrische cirkels (cirkels met hetzelfde middelpunt) waarbij de ruimte tussen de cirkels is ingekleurd. Hiervoor is het PAINT bevel de coördinaten van een punt gegeven dat tussen deze twee cirkels ligt (regel 70). De ring die hierdoor wordt getekend wordt doorsneden door twee lijnen. U zult het met mij eens zijn dat het wel erg lastig is een lijn zo een cirkel te laten snijden dat dat geen moeilijkheden kan geven met de kleuren!

```
10 'voor witte achtegrond
20 COLOR ,15: SCREEN 2
30 'voor twee concentrische cirkels
40 CIRCLE(128,95),50,10,,,1.4
50 CIRCLE(128,95),25,10,,,1.4
60 'inkleuren ruimte tussen de cirkels
70 PAINT (100,95),10
80 'twee lijnen door de cirkel
90 LINE(0,0)-(191,191),8
100 LINE(0,0)-(255,191),1
110 GOTO 110
```



14.3. Het DRAW statement

Het is mogelijk een figuur te laten tekenen door steeds vanuit een gegeven punt de richting en de afstand van een tweede punt aan te geven. Tussen

deze punten wordt dan een rechte lijn getrokken. Het laatste punt dient dan als startpunt voor een volgende lijn waarvoor weer met de richting en de afstand het eindpunt kan worden gegeven. Dit kan worden herhaald tot het gehele figuur is getekend. Er is ook een mogelijkheid om het volgende punt door middel van coördinaten aan te wijzen. Al de gegevens dienen in een string achter het statement DRAW te worden geplaatst. De volgende gegevens dienen te worden ingevoerd:

a. De kleur van de lijn.
b. De plaats van het beginpunt.
c. De plaats van het eerstvolgende punt.

Wat betreft de kleur van de lijn kunnen we kort zijn. Daarvoor wordt in de string een letter geplaatst (de c) gevolgd door het kleurnummer. Na deze instructie krijgt elk getekend lijnstuk deze kleur, tot een nieuwe kleurinstructie een andere kleur voorschrijft.

In principe is de plaats van het beginpunt, het eindpunt van de laatst getrokken lijn (ook de lijn die met een ander statement dan DRAW is getekend). Een nieuw beginpunt kan worden aangewezen zoals een volgend punt wordt aangewezen. Deze instructie dient dan te worden voorafgegaan door de letter b. Het is dan alsof een (onzichtbare) cursor naar de nieuwe plaats is gegaan om vandaar uit verder te tekenen.

Voor het aangeven van de plaats van een volgend punt zijn er verschillende mogelijkheden. Ten eerste kunnen de coördinaten van dat punt worden ingevoerd door mx,y in de string te plaatsen. Ook kan worden aangegeven hoeveel stippen in X-richting (naar rechts of naar links) en hoeveel in Y-richting moet worden gegaan om het volgende punt te vinden. Hiervoor dient m ± X, ± Y. De derde manier is die waarbij de afstand en de richting van het volgende punt wordt gegeven. Voor de afstand wordt eerst de letter s ingevoerd gevolgd door een getal. Het getal is een maat voor het aantal stippen dat een eenheid bevat. Later moet dan weer het aantal eenheden worden vermeld. Wilt u eenheden van 5 stippen dan moet u een getal achter s vermelden van 4*5 = 20: s20. Voor 8 stippen per eenheid moet dus s32 in de string achter DRAW worden geplaatst. Het aantal stippen per eenheid blijft gelden tot een nieuwe waarde wordt ingevoerd.

Voor de richting wordt een letter ingevoerd. Er zijn acht mogelijkheden.

rn	n eenheden naar rechts.
fn	naar rechtsonder.
dn	n eenheden naar onder.
gn	naar linksonder.
ln	n eenheden naar links.
hn	naar linksboven.
un	n eenheden naar boven.
en	naar rechtsboven.

Ook bij een gecombineerde beweging (bijvoorbeeld rechtsonder) heeft n betrekking op het aantal eenheden. In dat geval wordt n eenheden horizontaal verplaatst en evenzoveel eenheden verticaal. ''s40r5'' houdt in dat de verplaatsing per eenheid 10 stippen is en r5 geeft de plaats van het volgende punt op 5*10 stippen naar rechts. Om met ''s40e3'' het volgende punt te vinden moet u 30 stippen naar rechts en 30 stippen naar boven. In het volgende programma ziet u al deze mogelijkheden toegepast.

```
10 'voor grijze achtgrond
20 COLOR ,14: SCREEN 2
30 'naar het beginpunt
40 DRAW"bm100,40
50 'vier stippen per eenheid
60 'teken de omtrek
70 DRAW"s20c1r8f8d8g8L8h8u8e8"
80 'verticale lijnen
90 DRAW"c12br8d24bL8u24"
100 'van linksboven naar rechtsonder
110 DRAW"c6f16bg8h16"
120 'horizontale lijnen
130 DRAW"c2r24bd8124"
140 'van rechtsboven naar linksonder
150 DRAW"c8bm+8,+8m+8,-24"
160 GOTO 160
```

Achter de REM statements kunt u lezen door welke regels van het programma bepaalde lijnen zijn getekend.

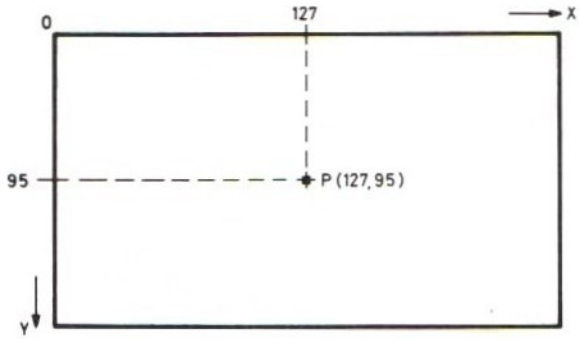
Indien gewenst kan de oriëntatierichting over hoeken van 90° worden veranderd. Daarmee wordt dan bedoeld dat door de instructie rn geen lijn naar rechts, maar naar boven over het scherm wordt getrokken als de oriëntatierichting over een hoek van 90° is gedraaid. Door middel van de instructie an in de string achter draw kan de oriëntatierichting n keer 90° worden gedraaid. Bij n = 0 treedt geen draaiing op. De grootste waarde voor n is 3. Het volgende programma demonstreert de werking.

```
10 SCREEN 2
20 FOR N=0 TO 3: A$="a"+STR$(N)
30 DRAW"xa$;c10bm127,95s20d7r5u7L5"
40 NEXT
50 GOTO 50
```

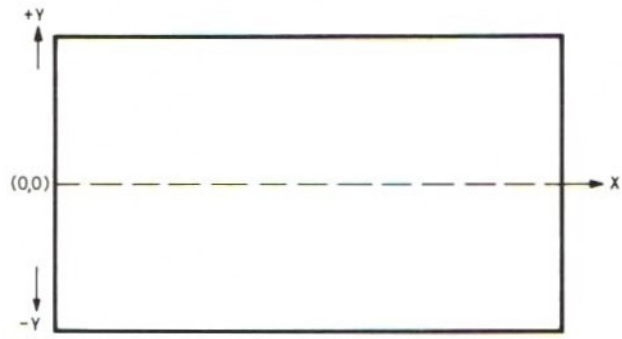
Door FOR A = 0 TO 3 wordt vier keer een rechthoek getekend, eerst met A = 0 zodat geen verandering van de oriëntatierichting optreedt en een vierkant wordt getekend (rechtsonder). Dit is een vierkant daar in de string achter draw voor r en L, 5 eenheden zijn genomen en voor u en d, 7 (7 = 5*1.4). Omdat daarna de oriëntatierichting is gedraaid kan het volgende figuur (rechtsboven) geen vierkant zijn. De verhouding van het aantal stippen klopt nu niet meer. Het derde figuur is weer een vierkant en het vierde een rechthoek. Nu is de oriëntatierichting drie keer gedraaid en zal zo blijven totdat de oude situatie weer met DRAW "a0" is hersteld. Let op de manier waarop in regel 30 de stringvariabele A$ is ingevoerd.

14.4. Grafieken

We hebben in dit hoofdstuk nu al een aantal keren een punt op het scherm moeten oriënteren en daarvoor werden de X- en de Y-coördinaten van dat punt bepaald. Dit kan omdat we in principe gebruik maken van een denkbeeldige horizontale lijn boven aan het scherm, de X-as. De getallen op deze as geven het nummer van de kolommen van de stippen aan. Het grootste getal op de X-as is daarom 255 en het kleinste 0. Verder maken we gebruik van een denkbeeldige verticale lijn aan de linkerkant van het scherm, de Y-as. De getallen op de Y-as geven het nummer van de stippenregels aan. Het grootste getal op de Y-as is 191. Een punt op het scherm wordt genoteerd door tussen haakjes eerst de X-waarde en daarna de Y-waarde te noteren (afb. 17). Dat de X-as bovenaan het scherm loopt bevalt in veel situaties niet. Vaak is het beter de X-as als een denkbeeldige lijn over het midden van het scherm te laten lopen (afb. 18). Vanaf het punt (0,0) is Y positief naar boven met een maximum van 95 en negatief naar onder met een maximum van 96. De computer trekt zich echter niets aan van ons gescharrel met de X-as en moet gewoon de coördinatie ontvangen die op de oude manier zijn berekend. Oriënteren we een punt met de X-as in het midden van het scherm dan moet voor de verticale coördinaat de waarde 95-Y worden ingevoerd.



afb. 17



afb. 18

Voor het tekenen van grafieken van functies maken we gebruik van het statement PSET. We hebben al eens gezien hoe hiermee een rechte lijn werd getekend. Behalve voor cirkels en ellipsen is het echter het enige statement waarmee ook kromme lijnen op het scherm kunnen worden geprint. We beginnen toch maar met een rechte lijn:

```
10 COLOR ,1: SCREEN 2
20 LINE(0,95)-(255,95),3
30 FOR X=0 TO 240: Y=.75*X-90
40 PSET(X,95-Y),15: NEXT X
50 GOTO 50
```

Eerst wordt over het midden van het scherm een groene lijn getekend die de X-as moet voorstellen. Daarna wordt voor elke waarde van X (van 0 tot en met 240) de bijbehorende Y-waarde berekend met Y = .75*X – 90.
Steeds wordt bij elke doorgang van de lus een punt (X,95 – Y) aangezet met PSET. Het resultaat is een rechte lijn die een hoek met de X-as maakt kleiner dan 45° en uit een aaneengesloten rij punten bestaat. bij een lijn die een grotere hoek maakt dan 45° zijn de punten echter niet meer aangesloten:

```
10 COLOR ,1: SCREEN 2
20 LINE(0,95)-(255,95),3
30 FOR X=0 TO 60: Y=3*X-90
40 PSET(X,95-Y),15: NEXT X
50 GOTO 50
```

Dit soort lijnen kunnen we beter omwerken tot x = f(y) en hem laten tekenen bij verschillende waarden van Y (FOR Y = 90 TO 90):

```
10 COLOR ,1: SCREEN 2
20 LINE(0,95)-(255,95),3
30 FOR Y=-90 TO 90: X=(Y+90)/3
40 PSET(X,95-Y),15: NEXT Y
50 GOTO 50
```

Het volgende programma tekent een sinusfunctie op het scherm (als u zich tot deze functies niet aangetrokken voelt slaat u ze gewoon over):

```
10 COLOR ,1: SCREEN 2
20 LINE(0,95)-(255,95),3
30 FOR X=0 TO 255: Y=90*SIN(X*.025)
40 PSET(X,95-Y),15: NEXT X
50 GOTO 50
```

Voor de diverse waarden van X wordt Y berekend met Y = 90*sin(X*.025). De vermenigvuldiging van X met .025 geeft bij X = 251 een waarde van 6.28 (2π) zodat tenminste één volledige periode op het scherm wordt getekend. De maximumwaarde van Y is 90. Een andere bekende functie is de parabool.

```
10 COLOR ,1: SCREEN 2
20 LINE(0,95)-(255,95),3
30 FOR X=80 TO 174
40 Y=.1*X^2-25.4*X+1520
50 PSET(X,95-Y),15: NEXT X
60 GOTO 60
```

Ook nu wordt over bepaalde gedeelten geen aaneengesloten lijn getekend. Dan maar proberen met x = f(y)

```
10 COLOR ,1: SCREEN 2
20 LINE(0,95)-(255,95),3
30 FOR Y=-92 TO 95
40 X1=(25.4+SQR(25.4^2-.4*(1520-Y)))/.2
50 X2=(25.4-SQR(25.4^2-.4*(1520-Y)))/.2
60 PSET(X1,95-Y),15
70 PSET(X2,95-Y),15: NEXT Y
80 GOTO 80
```

Dat betekent dan wel dat steeds een vierkantsvergelijking moet worden opgelost en dat kost tijd. Denk er om dat bij een dalparabool de waarde van Y niet te klein mag worden! Ook nu is het nog niet helemaal zoals het moet zijn en daarom gaan we voor het onderste gedeelte van de grafiek de twee programma's combineren:

```
10 COLOR ,1: SCREEN 2
20 LINE(0,95)-(255,95),3
30 FOR Y=-92 TO 95
40 X1=(25.4+SQR(25.4^2-.4*(1520-Y)))/.2
50 X2=(25.4-SQR(25.4^2-.4*(1520-Y)))/.2
60 PSET(X1,95-Y),15
70 PSET(X2,95-Y),15: NEXT Y
80 FOR X=120 TO 135
90 Y=.1*X^2-25.4*X+1520
100 PSET(X,95-Y),15:NEXT X
110 GOTO 110
```

14.5. Het printen van tekst

Zoals u weet hebben we in deze mode geen "echte" karaktergenerator ter beschikking en om karakters uit de normale karakterset op het scherm te brengen zullen we de bitpatronen moeten invoeren in het bit map geheugen. Nu kent de MSX computer hiervoor een erg eenvoudige methode.
Op dezelfde manier als we een reeks karakters in een file naar de cassetterecorder kunnen schrijven kunnen we dat ook naar het scherm. Uiteindelijk kunnen we ook het scherm als een randapparaat beschouwen. De naam voor dit randapparaat is GRP.
Verander het voorgaande programma (toch naar intypen dus) regel 10 en voeg de andere regels hieraan toe:

```
10 COLOR 7,1: SCREEN 2
110 OPEN"grp:" FOR OUTPUT AS #1
120 PRESET (10,40)
130 PRINT #1,"Parabool"
140 CLOSE
150 GOTO 150
```

Als u zich nog kunt herinneren hoe de werking was bij het saven van een file naar het cassettedeck dan zult u met deze regels geen enkele moeite hebben. Om de karakters op de juiste plaats te krijgen is het statement PRESET gebruikt. U kunt daarmee de tekens op elke plaats van het scherm brengen.

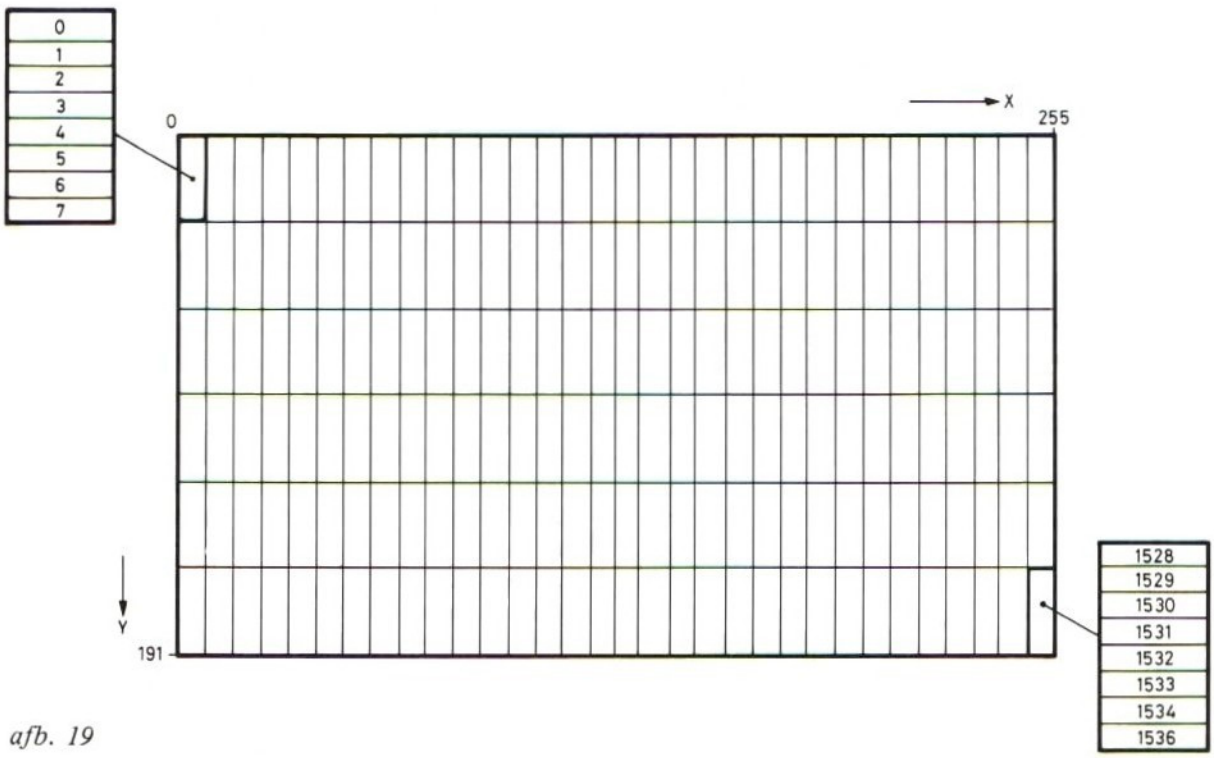
15. Grafische functies, multi color mode

15.1. Indeling van het scherm

De multi color mode heeft veel overeenstemming met de hoge resolutiemode, hoewel er van hoge resolutie bepaald niet meer kan worden gesproken. Ook in deze mode worden er stippen op het scherm geplaatst maar die zijn maar liefst in zowel de hoogte als de breedte vier keer zo groot als de stippen waaruit het veld is opgebouwd. We zullen daarom maar niet meer van stippen spreken maar van ''blokjes'' en elk blokje beslaat een rooster van vier bij vier stippen. Ook bij deze mode is er sprake van een bit map geheugen (de oorspronkelijke karaktergenerator) en een schermgeheugen. Dit laatste is weer gevuld met oplopende getallen. De indeling van de VIDEORAM is als volgt:

0 -	1535	bit map geheugen en kleurgeheugen.
2048 -	2815	schermgeheugen.
6912 -	7039	spritekenmerkgeheugen.
14336 -		spritepatroongeheugen.

De indeling van het bit map geheugen komt overeen met dat van de hoge resolutiemode. De eerste acht geheugenplaatsen corresponderen met een rooster van twee blokjes breed en acht blokjes hoog. Elke geheugenplaats bevat de gegevens voor twee naast elkaar liggende blokjes. De volgende acht geheugenplaatsen bevatten de gegevens voor het daarnaast liggende rooster van twee blokjes breed en acht blokjes hoog. In de hoogte van het scherm passen zo zes van deze roosters en op een regel 32. In totaal passen 6*32 = 192 roosters op het scherm. Voor elk rooster zijn acht geheugenplaatsen nodig zodat het bit map geheugen er 8*192 = 1536 moet bevatten. Afb. 19 geeft hiervan een voorstelling. Het bit map geheugen is tevens



afb. 19

het kleurgeheugen. De lage tetrade van een geheugenplaats bevat de kleur die het rechter blokje zal aannemen en de hoge tetrade de kleur die het linker blokje zal aannemen. Probeer hiervoor het volgende programma:

```
10 COLOR ,15: SCREEN 3
20 FOR X=0 TO 7: VPOKE X,16*(15-X)+(X+1)
30 NEXT
40 GOTO 40
```

De eerste regel in het programma zorgt met COLOR ,15 er voor dat het scherm geheel wit zal worden. Dat houdt in dat hierdoor elke geheugenplaats van het bit map geheugen gevuld zal worden met 16*15+15, dus het kleurnummer 15 in de hoge tetrade en in de lage tetrade. Daarna worden de eerste acht geheugenplaatsen van het bit map geheugen met kleurnummers ingeschreven. De eerste geheugenplaats met 16*15+1 (X=0), dus het kleurnummer voor wit in de hoge tetrade en het kleurnummer voor zwart in de lage tetrade. Dat is ook duidelijk aan het resultaat op het scherm te zien. U ziet op het scherm het eerste rooster van 2×8 blokjes in alle kleuren die maar mogelijk zijn. Op een regel kunnen niet meer dan 64 blokjes. Desondanks is de regel van 0 tot en met 255 gemerkt. Dus niet de nummers van de blokjes is aangegeven maar de nummers van de stippen. Dat houdt in dat elk blokje met vier X-coördinaten kan worden aangegeven. Het volgende programma laat een blokje langs de X-as over het scherm gaan. Een zelfde programma hebben we ook al eens bij de hoge resolutiemode geprobeerd.

```
10 SCREEN 3
20 FOR X=0 TO 255
30 PSET(X,0),10
40 FOR V=0 TO 10: NEXT V
50 PRESET(X,0)
60 NEXT X
70 GOTO 70
```

In verticale richting is de nummering van 0 tot en met 191 aangehouden, dus ook vier coördinaten per blokje.

```
10 SCREEN 3
20 FOR Y=0 TO 191
30 PSET(0,Y),10
40 FOR V=0 TO 10: NEXT V
50 PRESET(0,Y)
60 NEXT Y
70 GOTO 70
```

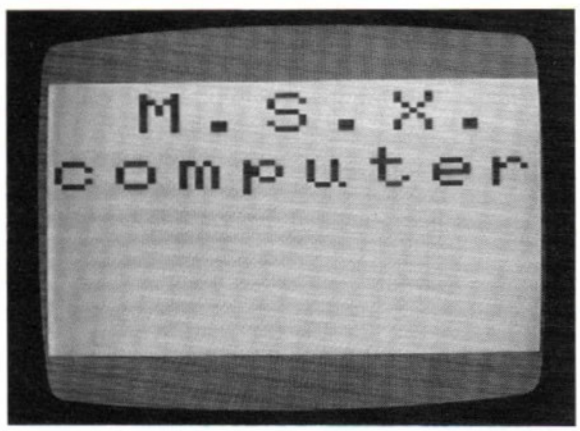
Het bovenstaande programma laat een blokje langs de Y-as over het scherm gaan.

Het resultaat van deze indeling is dat een blokje met zestien (vier bij vier) verschillende coördinaten kan worden aangegeven. In het volgende programma wordt vier keer een blokje getekend met vier verschillende coördinaten. Deze coördinaten vallen echter alle vier binnen hetzelfde blokje en daarom zien we er ook maar één op het scherm. In dit programma is het statement PSET gebruikt.

```
10 COLOR ,10: SCREEN 3
20 PSET(40,40),1:PSET(41,40),1
30 PSET(40,41),1:PSET(41,41),1
40 GOTO 40
```

Alle opdrachten die bij de hoge resolutiemode mogelijk zijn kunnen ook hier worden gebruikt. Ook de manier van het printen van tekst op het scherm is hetzelfde, hoewel het resultaat wel enigszins verschillend is.
Probeer het volgende programma maar eens:

```
10 COLOR 6,15: SCREEN 3
20 OPEN"grp:" FOR OUTPUT AS #1
30 PRESET(50,8)
40 PRINT #1,"M.S.X."
50 PRESET(6,50)
60 PRINT #1,"computer"
70 CLOSE
80 GOTO 80
```

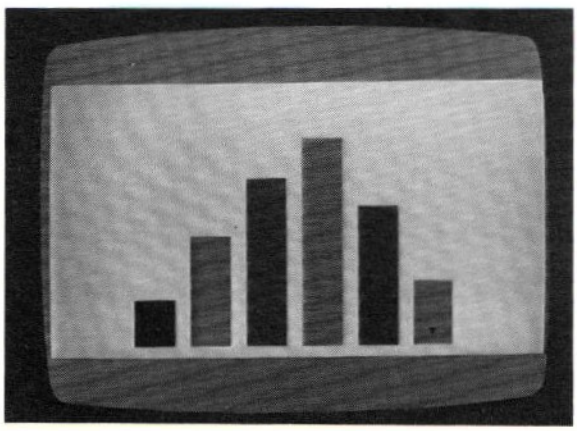


De volgende basisregisters bevatten de aanvangsadressen voor de geheugenblokken in de multi color mode.

BASE(15) schermgeheugen.
BASE(16) wordt niet gebruikt.
BASE(17) bit map geheugen.
BASE(18) spritekenmerkgeheugen.
BASE(19) spritepatroongeheugen.

15.2. Twee toepassingen

Eén van de toepassingen van het scherm is het gebruik voor een zogenaamd "staafdiagram". Het volgende programma geeft hiervan een voorbeeld. Hierin worden zes verticale kolommen getekend van verschillende kleur en van verschillende lengte. In de regels 20 tot en met 70 wordt met A het kolomnummer (van 0 tot en met 5), met H de hoogte van de kolom en met C de kleur van de kolom ingevoerd. Elke regel verwijst naar de subroutine op regel 90 waarin de kolom wordt getekend. Eerst wordt in regel 90 de plaats berekend waar de kolom in X-richting moet beginnen. Elke kolom neemt in totaal 28 stippen in beslag en de eerste begint op coördinaat 44 (A = 0). De tweede begint dan op de coördinaat 44 + 1*28, de derde op coördinaat 44 + 2*28 enz.
De breedte van de kolom wordt bepaald door X2 = X1 + 16. In regel 120 wordt eerst een aantal blokjes naast elkaar getekend op de regel met de Y-coördinaat 180. Omdat de blokjes vier stippen breed zijn kan X steeds met 4 worden verhoogd (regel 110). Daarna wordt in regel 100 de Y-coördinaat met vier stippen verlaagd en kan opnieuw een aantal blokjes worden getekend. Dit gaat door tot de Y-waarde gelijk is aan 18 – H. Dan is de hoogte van de getekende kolom gelijk aan H.



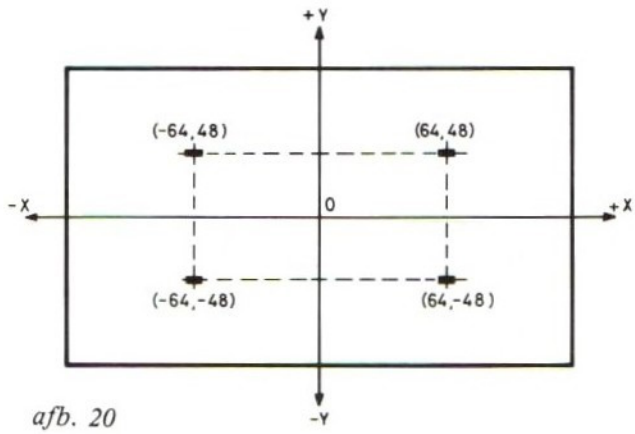
```
10 COLOR 1,15: SCREEN 3
20 A=0: H=28: C=1: GOSUB 90
30 A=1: H=72: C=4: GOSUB 90
40 A=2: H=112: C=6: GOSUB 90
50 A=3: H=140: C=10: GOSUB 90
60 A=4: H=92: C=12: GOSUB 90
70 A=5: H=40: C=13: GOSUB 90
80 GOTO 80
90 X1=44+A*28: X2=X1+16
100 FOR Y=180 TO 180-H STEP -4
110 FOR X=X1 TO X2 STEP 4
120 PSET(X,Y),C: NEXT X: NEXT Y
130 RETURN
```

In deze mode is het ook mogelijk om toevalsfiguren op het scherm te laten tekenen.

```
10 CLS: PRINT"druk op een toets"
20 A=RND(1)
30 A$=INKEY$: IF A$="" THEN 20
40 COLOR 1,15: SCREEN 3
50 FOR N=0 TO 600
60 X=INT(256*RND(1))
70 Y=INT(192*RND(1))
80 C=INT(14*RND(1)+1)
90 PSET(X,Y),C: NEXT N
100 A=RND(1)
110 A$=INKEY$: IF A$="" THEN 100
120 GOTO 40
```

Door dit programma worden 600 blokjes op het scherm geplaatst. De achtergrond is wit (regel 40). De X- en de Y-coördinaten worden met een toevalsgetal bepaald (regels 60 en 70) evenals de kleur (regel 80). Dit is een "echt"toevalsgetal (regels 20 en 30). Nadat alle blokjes op het scherm zijn geplaatst wordt in de regels 100 en 110 gewacht totdat weer op een toets wordt gedrukt voor een volgend toevalsfiguur.

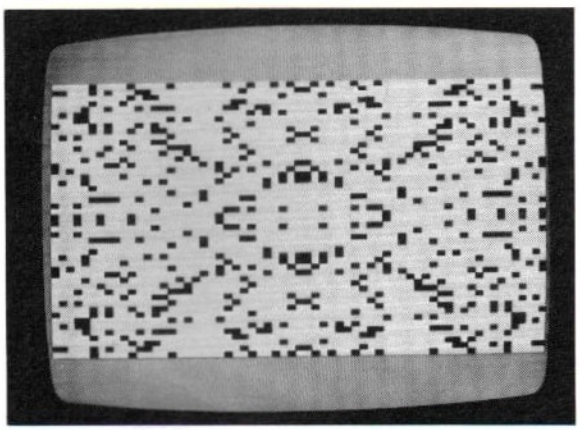
Nu zijn dit soort toevalsfiguren in het algemeen het aankijken niet waard. Veel interessanter wordt het als we er symmetrische figuren van kunnen maken. Daarvoor moeten we zowel de X-as als de Y-as over het midden van het scherm laten lopen. Wat betreft de X-as hebben we dat al eerder ontmoet. we weten dat we in dat geval de waarde voor de Y-coördinaat moeten invoeren met Y1 = 95-Y. Als we



afb. 20

daarbij ook de Y-as over het midden van het scherm laten lopen (afb. 20) dan moet de X-coördinaat worden ingevoerd met $X1 = X + 127$. Om een symmetrische figuur te maken moet elk blokje dat op het scherm is geplaatst vier keer op het scherm voorkomen, waarbij elk blokje een spiegeling van een ander is. Hiertoe worden de coördinaten van het blokje in het eerste kwadrant bepaald met $+X$ en $+Y$. Daarna wordt het blokje getekend met $+Y$ en $-X$, met $-Y$ en $+X$ en ook met $-Y$ en $-X$. De absolute waarden van X en Y zijn in alle kwadranten uiteraard gelijk. Zie hiervoor ook het punt in afb. 20.

```
10 'schakelt de stoptoets uit
20 ON STOP GOSUB 300
30 STOP ON
40 'maak RND(1) tot toevalsgetal
50 CLS: PRINT"druk op een toets"
60 A=RND(1)
70 A$=INKEY$: IF A$="" THEN 60
80 'zwarte karakters, witte achtergrond
90 COLOR 1,15: SCREEN 3
100 'voor 150 stippen
110 FOR N=0 TO 150
120 'bepaal X-waarde 1e kwadrant
130 X1=INT(32*RND(1))*4+1
140 'bepaal Y-waarde 1e kwadrant
150 Y1=INT(24*RND(1))*4+1
160 'bepaal kleurnummer voor de stip
170 C=INT(14*RND(1)+1)
180 'voor een pos. en neg. Y-waarde
190 FOR AY=1 TO -1 STEP -2
200 'voor een pos. en neg. X-waarde
210 FOR AX=1 TO -1 STEP -2
220 X=X1*AX: Y=Y1*AY
230 PSET(X+127,95-Y),C
240 NEXT AX: NEXT AY: NEXT N
```



```
250 'wacht op het indrukken v.e. toets
260 A=RND(1)
270 A$=INKEY$: IF A$="" THEN 260
280 'voor het volgende figuur
290 GOTO 90
300 'subr. voor uitschakelen stoptoets
310 SCREEN 0
320 PRINT "onderbreking opgeheven"
330 'vertraging
340 FOR N=0 TO 2000: NEXT N
350 'start het programma opnieuw
360 RETURN 50
```

In regel 90 van dit programma zorgen we voor een witte achtergrond en in regel 10 dat niet meer dan 4*150 blokjes op het scherm komen. In de regels 130 en 150 worden de coördinaten van het blokje in kwadrant 1 bepaald door echte toevalsgetallen (regels 60 en 70). Omdat door de regels 190 en 210 zowel AY als AX systematisch van teken worden veranderd worden in regel 220 de X- en de Y-coördinaten van het blokje na elkaar positief en negatief. Daardoor worden in regel 230 vier blokjes symmetrisch ten opzichte van de assen op het scherm getekend. Deze werking herhaalt zich 150 keer. Nadat een willekeurige toets is ingedrukt wordt een volgend figuur getekend.

In hoofdstuk 7 heb ik u aangeraden om elk programma direct na het invoeren in de computer, voor de zekerheid naar de cassette te schrijven. Hebt u deze raad bij dit programma niet nagevolgd dan hebt u nu pech. als u het hebt gestart met RUN dan is het niet meer op de normale manier te onderbreken (met CTRL en STOP). Deze zijn uitgeschakeld in de regels 20 en 30 van het programma. Dit is weer een statement dat een interrupt doet optreden als aan een bepaalde voorwaarde is voldaan. In dit geval is de voorwaarde het indrukken van de CTRL en STOP toetsen. Dan wordt naar de subroutine op regel 300 gesprongen die u op het scherm laat weten dat deze manier van onderbreken is opgeheven en daarna weer rustig het programma opnieuw bij regel 50 laat beginnen. De enige manier om het programma te laten stoppen is die van de RESET toets of door het uitschakelen van de computer. Staat het programma niet op de band dan bent u het nu kwijt.