BASIC-PROGRAMMA'S VOOR

ZX SPECTRUM

PROGRAMMEURS



Wessel Akkermans

Kluwer Software-reeks

BASIC-programma's voor ZX Spectrum-programmeurs

Wessel Akkermans

BASIC-programma's voor ZX Spectrum-programmeurs



Kluwer Technische Boeken B.V.
Deventer-Antwerpen

Omslag: W. Niessink

ISBN 90 201 1731 9
D/1984/0108/190

© 1984 Kluwer Technische Boeken B.V. - Deventer

1e druk 1984

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

No part of this book may be reproduced in any form, by print, photoprint, microfilm or any other means without written permission from the publisher.

Ondanks alle aan de samenstelling van de tekst bestede zorg, kan noch de redactie noch de uitgever aansprakelijkheid aanvaarden voor eventuele schade, die zou kunnen voortvloeien uit enige fout, die in deze uitgave zou kunnen voorkomen.

Woord vooraf

Bij het in BASIC programmeren van de ZX Spectrum, blijkt al gauw dat er een aantal voor het programmeren onontbeerlijke commando's ontbreken.
Zo kent de Spectrum bijvoorbeeld niet het commando RENUM, voor het hernummeren van de BASIC-regels. Ook zou het erg handig zijn de beschikking te hebben over de mogelijkheid een aantal opeenvolgende BASIC-regels in één keer uit het programma te wissen, of om bijvoorbeeld alle REM-instructies uit een programma te kunnen wissen zonder zelf regel voor regel te moeten zoeken en verwijderen.

Al dit soort functies kan men zelf programmeren. Om deze programma's echter redelijk kort te houden en om ze snel te laten zijn, is het wel nodig om in die programma's gebruik te maken van de reeds in iedere standaard ZX Spectrum aanwezige ROM-routines.
Het gebruik van ROM-routines vereist echter tevens het gebruik van enkele zeer korte machinetaalroutines. Daar het invoeren van machinetaal niet zonder meer mogelijk is, besloot ik een zo universeel mogelijk machinetaalinvoerprogramma te schrijven. Met behulp van dit programma kunt u dan op iedere door u gewenste geheugenlocatie uw machinetaalroutine invoeren, verbeteren en uitbreiden. Daar u dit machinetaalinvoerprogramma voor bijna alle daarna volgende programma's nodig hebt, begint dit boek met het beschrijven van dat invoerprogramma.

Hopelijk zult u net als ikzelf bij het gebruik van de daarna volgende programma's versteld staan van de mogelijkheden en de soms onverwacht hoge snelheid van uw ZX Spectrum.
Ook hoop ik dat u niet alleen nuttig gebruik van de in dit boek geboden programma's kunt maken, maar dat u er bovendien door zult worden aangespoord om zelf ook meer gebruik te gaan maken van de in de ZX Spectrum aanwezige mogelijkheden.

Inhoud

Inleiding 9

1. Aanwijzingen en tips 11

2. Opslaan en teruglezen van programma's 15

3. Machinetaalinvoerprogramma 19
3.1 Programmabeschrijving 19
3.2 Machinetaal programmeren 20
3.3 Uitvoeren machinetaalprogramma 29
3.4 Op cassette bewaren 30
3.5 Van cassette lezen 31
3.6 Back-up van machtaal 31

4. Afdrukken van geselecteerde BASIC-regels 40
4.1 ROM-routines voor het afdrukken van een regel 40
4.2 Het programma 40
4.3 De printer-versie 43

5. Het verwijderen van een blok BASIC-regels 44

6. Cassettelabelinformatie lezen 49
6.1. Het programma 'labelscrn' 53
6.2 Het programma 'labelprint' 56

7. Selectief wissen van BASIC-regels 62

8. Hernummeren van BASIC-regelnummers 68

9. Het kopiëren van BASIC-regels 76

10. Het verplaatsen van BASIC-regels 88

11. Opsomming van de gebruikte ROM-routines 96

Appendix A Literatuurlijst 98
Appendix B Conversietabel hexadecimaal-decimaal 99

Inleiding

Alvorens te beginnen met de programmabeschrijvingen, is het goed een paar algemene opmerkingen vooraf te maken.
In dit boek wordt veelvuldig gebruik gemaakt van combinaties van machinetaal- en BASIC-programma's. Tussen het programmeren in machinetaal en in BASIC ligt een aantal grote verschillen.
Het eerste verschil is de manier van programmeren. Programmeren in machinetaal is over het algemeen moeilijker dan programmeren in BASIC. Het schrijven van een machinetaalprogramma zal dan ook meer tijd in beslag nemen dan het schrijven van een BASIC-programma.
Het tweede, minstens even belangrijke, verschil zit in de manier waarop de twee soorten programma's in het geheugen van de ZX Spectrum worden opgeborgen.
Een BASIC-programma wordt in het BASIC-programmageheugen opgeslagen. Dit deel van het geheugen is volledig onder controle van het BASIC-systeem. Dit BASIC-systeem controleert alle door u of door het programma uit te voeren handelingen. Doet u iets verkeerd, of doet uw programma iets verkeerd, dan zult u daarvan op de hoogte worden gesteld door middel van een systeemboodschap op het scherm. U kunt dan uw fout herstellen en opnieuw starten.
Een machinetaalprogramma kan op verschillende plaatsen binnen het geheugen worden opgeslagen. De meest voor de hand liggende plaats is echter een plaats die buiten het bereik van het BASIC-systeem ligt. Op die manier voorkomt men namelijk dat, door het dynamische geheugengebruik van het BASIC-systeem, het machinetaalprogramma wordt overschreven.
Men doet er dus goed aan zijn machinetaalprogramma's boven het in de systeemvariabele RAMTOP aangegeven adres te plaatsen. Dit adres wordt aangeduid met RAMTOP, we kunnen dus ook zeggen dat we ons programma boven RAMTOP plaatsen.
Het BASIC-systeem heeft zoals gezegd geen toegang tot dit deel van het geheugen. Dit houdt dan tevens in dat het BASIC-systeem geen controle op ons machinetaalprogramma kan uitoefenen. Wij zijn helemaal zelf verantwoordelijk voor wat er met de routines boven RAMTOP gebeurt. Zit er een fout in zo'n routine, dan is er geen BASIC-systeem dat daarvan een melding geeft. In tegendeel, het programma wordt gewoon verder uitgevoerd alsof er niets aan de hand is. Dit kan evenwel tot gevolg hebben dat er de meest vreemde dingen gaan gebeuren. Het ergste wat zou kunnen gebeuren is dat het systeem in een 'hang up' komt. Dit wil zeggen dat het systeem helemaal niets meer doet. Het luistert niet meer naar onze opdrachten, maar het voert ook geen programma-instructies meer uit. Het enige wat er voor ons nog op zit is het uit- en weer aanschakelen van de ZX Spectrum.
Het uit- en weer aanschakelen van de ZX Spectrum heeft echter tot gevolg dat alles wat in het RAM-geheugen stond nu verdwenen is.
Daarom is het van het grootste belang dat u er altijd voor zorgt dat u uw machinetaalprogramma's eerst op cassette opslaat en ze pas daarna door de computer laat uitvoeren. Heeft u dan een fout gemaakt waardoor de computer blijft hangen, dan hoeft u niet het hele machinetaalprogramma opnieuw in te tikken, doch kunt u het gewoon weer vanaf de cassette laden.
Het hoofdstuk 'Opslaan en teruglezen van programma's' geeft een paar nuttige aanwijzingen die u bij het opslaan en teruglezen van combinaties van BASIC- en machinetaalpro-

gramma's kunt volgen. Leest u dat hoofdstuk maar eens door voordat u aan het programmeren slaat. U zult het zichzelf dan veel gemakkelijker kunnen maken om programma's zonder fouten op te slaan en weer terug te lezen.
Ten slotte nog het volgende:
Programmafouten kunnen nooit de oorzaak zijn van het defect raken van uw computer. Hooguit raakt het BASIC-systeem de kluts kwijt. Dit is echter vrij eenvoudig te herstellen, uit- en aanschakelen. Wees daarom niet bang zelf ook te experimenteren met het gebruik van systeemvariabelen, POKE-instructies en machinetaalprogramma's. Deze experimenten zullen namelijk laten zien dat uw Spectrum ongekende mogelijkheden heeft.

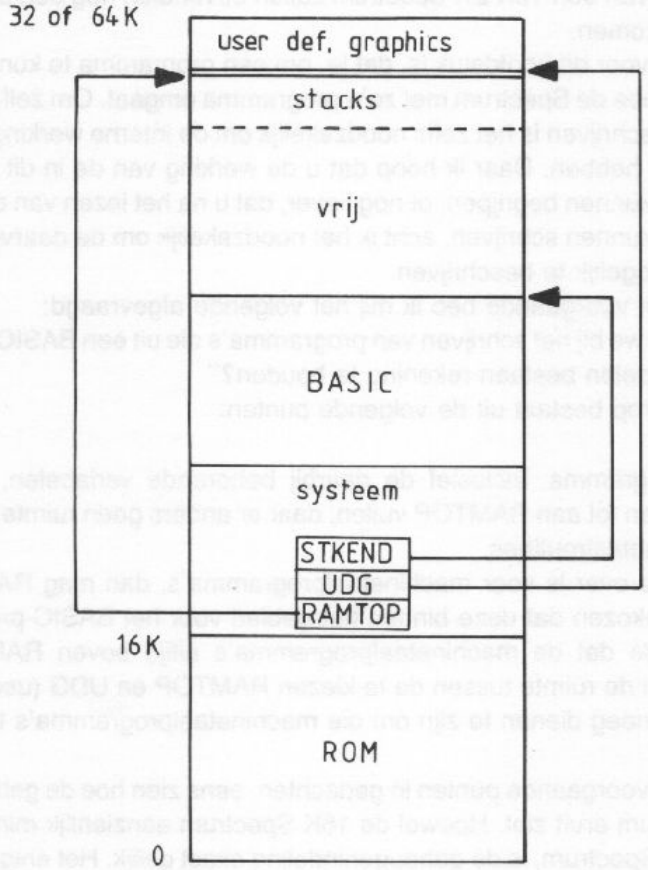
1. Aanwijzingen en tips

Voordat we met de beschrijving van de programmeer-utilities beginnen lijkt het mij goed nog even een aantal algemene aspecten, die daarbij komen kijken, onder de loep te nemen. De bezitters van een 16K ZX Spectrum zullen bovendien nog een aantal voor hen nuttige tips tegenkomen.
Mijn uitgangspunt voor dit hoofdstuk is, dat je, om een programma te kunnen begrijpen, goed moet weten hoe de Spectrum met zo'n programma omgaat. Om zelf programmeer-utilities te kunnen schrijven is het zelfs noodzakelijk om de interne werking van de Spectrum goed door te hebben. Daar ik hoop dat u de werking van de in dit boek gegeven programma's zult kunnen begrijpen, of nog liever, dat u na het lezen van dit boek zelf uw eigen utilities zult kunnen schrijven, acht ik het noodzakelijk om de daarvoor benodigde kennis zo goed mogelijk te beschrijven.
In verband met het voorgaande heb ik mij het volgende afgevraagd:
"Waarmee dienen we bij het schrijven van programma's die uit een BASIC-deel en één of meer machinetaaldelen bestaan rekening te houden?"
Het antwoord daarop bestaat uit de volgende punten:

a. Het BASIC-programma, inclusief de daarbij behorende variabelen, mag niet het gehele geheugen tot aan RAMTOP vullen, daar er anders geen ruimte meer overblijft voor de machinetaalroutines.
b. Indien er ruimte over is voor machinetaalprogramma's, dan mag RAMTOP niet zo laag worden gekozen dat deze binnen het gebied voor het BASIC-programma valt.
c. Ervan uitgaande dat de machinetaalprogramma's altijd boven RAMTOP worden geschreven, zal de ruimte tussen de te kiezen RAMTOP en UDG (user defined graphics) groot genoeg dienen te zijn om die machinetaalprogramma's te bevatten.

We zullen nu, met voorgaande punten in gedachten, eens zien hoe de geheugenindeling van de ZX Spectrum eruit ziet. Hoewel de 16K Spectrum aanzienlijk minder geheugen heeft dan de 48K Spectrum, is de geheugenindeling exact gelijk. Het enige verschil zit in de hoeveelheid vrije geheugenruimte. Die vrije geheugenruimte is echter voor beide machines variabel.
Stel, u hebt een BASIC-programma geschreven en in het geheugen geladen. Onmiddellijk na het laden, doch voor het starten van het programma zou u kunnen uitzoeken hoeveel geheugenruimte er nog vrij is. Zou u echter het programma gestart hebben en nogmaals uitzoeken hoeveel vrije geheugenruimte er is, dan maakt u een goede kans dat die geheugenruimte aanzienlijk minder groot is geworden.
Afbeelding 1-1 geeft een overzicht van de geheugenindeling. Wat daarin met systeem is aangeduid wil zeggen, het gebied waarin het beeldschermgeheugen, het attributengeheugen, het printbuffer, de systeemvariabelen enz. zijn opgeslagen. Het blok waar 'BASIC' in staat is het geheugengebied waarin het BASIC-programma, het variabelengeheugen, het EDIT- en werkgeheugen en de reken-stack zijn opgeslagen. Daarboven bevindt zich het deel van het geheugen dat niet wordt gebruikt. Vanaf het einde van het geheugen tot een variabel aantal bytes voor RAMTOP worden twee stacks opgebouwd tijdens het uitvoeren van een BASIC-programma, de GO SUB stack en een Z80-reken-stack.
Het zou voor ons interessant zijn om te weten hoe groot de vrije geheugenruimte is, om

zodoende te kunnen bepalen hoever we de RAMTOP naar omlaag kunnen verplaatsen, zonder de werking van het BASIC-programma te verstoren. Daartoe staan ons twee pointers ter beschikking en wel:
systeemvariabele STKEND en
systeemvariabele UDG.



Afb. 1-1 Geheugenindeling na aanschakelen

Het gebied 'stacks' varieert van moment tot moment, zodat we hiervoor een waarde zullen schatten. Laten we aannemen dat dit gebied 200 bytes lang kan zijn. Om dan de werkelijke vrije ruimte te kunnen bepalen kijken we wat het adres in STKEND is en wat het adres in UDG is. Nu trekken we de in STKEND gevonden waarde af van de waarde gevonden in UDG. Van het resultaat trekken we bovendien nog de 200 bytes voor 'stacks' af. De nu gevonden waarde geeft redelijk weer hoe groot de vrije geheugenruimte is. Hierbij dient nog wel te worden opgemerkt dat de gevonden waarde alleen geldt voor dit moment. Het variabelengeheugen en de werkruimte zijn namelijk ook variabel van lengte. Een variabele wordt pas in het variabelengeheugen opgenomen nadat die variabele in het programma is gebruikt. De werkruimte wordt onder andere gebruikt voor het tijdelijk opslaan van INPUT-gegevens en cassettelabelinformatie.
Uit het voorgaande zal u duidelijk zijn geworden dat het nauwelijks mogelijk is om te bepalen hoe groot de vrije ruimte minimaal zal zijn. We zullen dan ook genoegen nemen met het maken van een zo goed mogelijke schatting. Indien u de vrije ruimte gaat bepalen op het moment dat alle in het programma voorkomende variabelen zijn gebruikt, dan zal een

marge van 200 bytes in vrijwel alle gevallen voldoende zijn om het programma goed te laten verlopen.
Dan zullen we nu nog een voorbeeldje bekijken van de bepaling van de vrije geheugenruimte voor een denkbeeldig programma. Het programma is geladen en het is eenmaal uitgevoerd. Hierdoor zijn alle in dat programma voorkomende variabelen gebruikt en komen ze in het variabelengeheugen voor.
Nu lezen we de inhoud van de systeemvariabele UDG als volgt uit:

```
PRINT PEEK 23675+256*PEEK 23676
```

Van de afgedrukte waarde gaan we de inhoud van de systeemvariabele STKEND aftrekken. Deze waarde verkrijgen we als volgt:

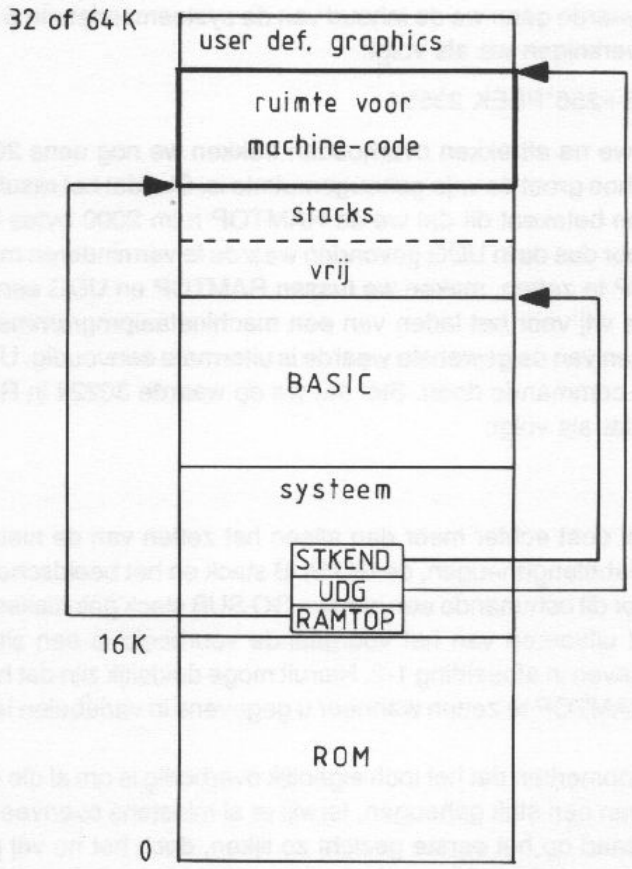
```
PRINT PEEK 23653+256*PEEK 23654
```

Van de waarde die we na aftrekken overhouden trekken we nog eens 200 af. Hiermee hebben we bepaald hoe groot de vrije geheugenruimte is. Stel dat het resultaat in dit voorbeeld 2044 was. Dan betekent dit dat we de RAMTOP ruim 2000 bytes beneden UDG kunnen plaatsen. Door dus de in UDG gevonden waarde te verminderen met 2000, en dat resultaat in RAMTOP te zetten, maken we tussen RAMTOP en UDG een geheugengebied van 2000 bytes vrij voor het laden van een machinetaalprogramma.
Het in RAMTOP zetten van de gewenste waarde is uitermate eenvoudig. U kunt dit namelijk met een CLEAR-commando doen. Stel dat we de waarde 30224 in RAMTOP willen schrijven, dan kan dat als volgt:

```
CLEAR 30224
```

De CLEAR-opdracht doet echter meer dan alleen het zetten van de nieuwe RAMTOP. Ook worden het variabelengeheugen, de GO SUB stack en het beeldscherm gewist. Bovendien wordt er door dit commando een nieuwe GO SUB stack geïnitialiseerd vlak onder RAMTOP. Door het uitvoeren van het voorgaande voorbeeld is een situatie ontstaan zoals die is weergegeven in afbeelding 1-2. Hieruit moge duidelijk zijn dat het niet is aan te raden een nieuwe RAMTOP te zetten wanneer u gegevens in variabelen hebt staan die u niet kwijt wilt.
Nu zult u misschien opmerken dat het toch eigenlijk overbodig is om al die moeite te doen voor het vrijmaken van een stuk geheugen, terwijl er al minstens evenveel geheugen vrij was. Dit mag inderdaad op het eerste gezicht zo lijken, doch het nu vrij gemaakte stuk geheugen kan niet door het BASIC-systeem worden overschreven, terwijl het vrije geheugen van voorheen juist wel door het BASIC-systeem werd gebruikt voor het uitbreiden van werkruimte en dergelijke. Het door ons gecreëerde vrije stuk geheugen boven RAMTOP is dus een zeer veilige plaats voor een machinetaalroutine.
We hebben nu gezien waar we een machinetaalroutine in het geheugen kunnen plaatsen, en hoe we die ruimte vrij kunnen maken. Met het programma 'machtaal' wordt die ruimte vrijgemaakt door een startadres op te geven. Dat startadres moeten we echter wel zelf bepalen, op de hiervoor aangegeven manier. Hebben we dat gedaan en hebben we vervolgens een routine ingegeven, dan is het nog zaak om die routine op de juiste plaats op een cassette te krijgen. Hoewel dit erg eenvoudig is, wil ik er toch nog een paar regels aan wijden.
Wanneer u een programma dat uit een BASIC- en uit een machinetaaldeel bestaat op cassette wilt schrijven, schrijft u dan eerst het BASIC-programma op die cassette. Bent u daarmee klaar, laadt u dan het programma 'machtaal', geeft u het machinetaalprogram-

ma in, en schrijf dit met de daartoe in 'machtaal' aanwezige functie weg naar de cassette waarop u zojuist het BASIC-programma had geschreven.
Probeert u het BASIC-deel en het machinetaaldeel zo dicht mogelijk na elkaar op de cassette te zetten. U zult dan zien dat, indien u in uw BASIC-programma het automatisch laden van machinetaalroutines hebt ingebouwd, het laden van het complete programma niet veel meer tijd in beslag neemt dan het laden van het BASIC-programma zelf.
Ten slotte wil ik u er nog op wijzen dat hoofdstuk 2 een aantal nuttige tips bevat over het schrijven naar en terugladen van cassette.



Afb. 1-2 Geheugenindeling na wijzigen RAMTOP

2. Opslaan en teruglezen van programma's

De hoge kwaliteit van de cassette-interface van de Spectrum draagt er niet alleen toe bij dat programma's en gegevens snel en foutloos naar cassette kunnen worden geschreven en weer teruggelezen. Het maakt ook de gebruiker van die interface onzorgvuldiger. Het gaat altijd goed, dus waarom zou je nog controleren of het wel goed is gegaan?
Enige tijd geleden had ik een programma, waaraan ik ongeveer 6 uur had gewerkt, nog gauw even op cassette geschreven. In verband met het late uur (en de hoge betrouwbaarheid van de cassette-interface) besloot ik geen 'VERIFY' uit te voeren. Toen ik de volgende dag mijn programma wilde afmaken bleek er echter helemaal niets op de cassette te staan. Ik was vergeten de MIC-plug in de cassetterecorder te steken.
U kunt zich voorstellen wat mijn eerste reactie was. Een gevolg was in ieder geval dat ik besloot mij in het vervolg aan bepaalde regels te houden. Van dat besluit heb ik inmiddels veel plezier gehad. Het is mij sindsdien niet meer overkomen dat ik het werk van vele uren in enkele seconden kwijtraakte.
Bovendien was één en ander voor mij aanleiding om eens na te gaan wat er verder met SAVE- en LOAD-statements mogelijk is om niet alleen veilig maar ook gemakkelijk te werken. De rest van dit hoofdstuk is een verslag van mijn ondervindingen. Hopelijk kan hiermee worden voorkomen dat u de door mij ondervonden problemen alsnog zult ondervinden.

Probleem:
Een programma is naar cassette weggeschreven, doch na verloop van tijd blijkt dat dit programma niet of niet goed op cassette staat.

Oplossing:
Voer altijd een 'VERIFY' uit, nadat u iets op cassette hebt geschreven, voordat u de computer uitzet of een NEW-commando geeft. Nog beter is het om zowel de SAVE- als VERIFY-opdracht in het programma op te nemen. U kunt dan op het moment dat u uw programma op cassette wilt zetten volstaan met een GO TO-statement.

Het in het programma opnemen van SAVE-, VERIFY- en LOAD-opdrachten is altijd aan te bevelen. Zeker, zoals we later zullen zien, wanneer er meer dan alleen een BASIC-programma moet worden weggeschreven of geladen. U doet er dan goed aan om ieder programma dat u schrijft te beginnen met een aantal regels waarmee programma's kunnen worden weggeschreven. Kiest u hiervoor een vast beginregelnummer, bijvoorbeeld regelnummer 9990. Het volgende probleem laat hiervan een voorbeeld zien.

Probleem:
Bij het naar cassette schrijven van een BASIC-programma + een machinetaalprogramma moet ik steeds uitzoeken wat ook al weer het startadres en de lengte van dat machinetaaldeel was.

Oplossing:
Door alle benodigde SAVE-statements zoals hiervoor beschreven in het programma op te nemen, met daarbij voor de machinetaaldelen het beginadres en de lengte gespecificeerd, hoef je niets meer te onthouden. Ga gewoon met een GO TO 9990 naar de routine die alles voor je op cassette schrijft.

Ter verduidelijking volgt hier een voorbeeld. Een programma dat uit een BASIC-deel en twee machinetaaldelen bestaat wil ik naar cassette wegschrijven. Het eerste machinetaaldeel staat in het geheugen vanaf adres 60000 en heeft een lengte van 50 bytes. Het tweede deel start op adres 61000 en heeft een lengte van 120 bytes. In het BASIC-programma heb ik dan de volgende regels opgenomen:

```
9990 CLS: PRINT AT 5,0;"Verwijde
r de EAR-plug"'"Steek de MIC-plu
g in de cassetterecorder"'"Start
de cassette op opname"
9991 SAVE "basprog" LINE 9995
9992 SAVE "machcode1" CODE 60000
,50
9993 SAVE "machcode2" CODE 61000
,120
9994 STOP
```

Waarschijnlijk is het u al opgevallen dat er in regel 9991 aan het SAVE-statement LINE 9995 is toegevoegd, terwijl regel 9995 nog helemaal niet bestaat. Het volgende probleem zal duidelijk maken wat ik op regel 9995 wil gaan doen.

Probleem:
Als ik een BASIC-programma van cassette lees weet ik niet of er machinetaalroutines bij dat programma behoren, en zo ja, hoeveel routines.

Oplossing:
Programmeer alle benodigde LOAD-statements in het BASIC-programma en zorg ervoor dat het eerste wat het BASIC-programma doet, nadat het is geladen, het laden van de machinetaaldelen is. Om absoluut zeker te zijn dat het laden als eerste gebeurt kun je het BASIC-programma wegschrijven met de 'LINE'-toevoeging.
Hierdoor start het programma automatisch met het uitvoeren van de achter LINE opgegeven regel.

Ook voor dit probleem zegt een voorbeeldje meer dan duizend woorden. Om het voorbeeldje compleet te houden herhaal ik nog even de programmaregels uit het vorige voorbeeld, waarna die regels worden uitgebreid met een paar nieuwe regels.

```
9990 CLS: PRINT AT 5,0;"Verwijde
r de EAR-plug"'"Steek de MIC-plu
g in de cassetterecorder"'"Start
de cassette op opname"
9991 SAVE "basprog" LINE 9995
9992 SAVE "machcode1" CODE 60000
,50
9993 SAVE "machcode2" CODE 61000
,120
9994 STOP
9995 LOAD "machcode1" CODE
9996 LOAD "machcode2" CODE
9997 GO TO 10
```

Doordat het programma zo was weggeschreven dat het automatisch startte op regel 9995 nadat het weer was ingelezen, en doordat op regel 9995 het laden van de machinetaaldelen begint, worden alle programmadelen keurig geladen, zonder tussenkomst van de gebruiker. Zoals regel 9997 laat zien ben ik ervan uitgegaan dat het werkelijke begin van het programma op regelnummer 10 is.
De oplettende lezer heeft waarschijnlijk al opgemerkt dat in de tot nu toe gegeven voorbeelden geen rekening is gehouden met het eerste probleem, het vergeten van de VERIFY. Die heb ik voor de duidelijkheid van het betoog even vergeten. In werkelijkheid zou na iedere SAVE-instructie een VERIFY-instructie moeten worden opgenomen.

Probleem:
Soms wil ik afwijken van mijn standaardregel om altijd een VERIFY te doen na een SAVE. Daar ik echter tussen de SAVE en de daaropvolgende VERIFY niets in het geheugen mag wijzigen kan ik geen INPUT-instructie uitvoeren waarmee ik kan laten vragen of een VERIFY wel of niet wordt gewenst.

Oplossing:
De INPUT-instructie moet worden vervangen door een PRINT plus een INKEY$. De bij INPUT ingegeven waarde wordt in een variabele opgeslagen, waardoor de geheugeninhoud zich wijzigt; de op INKEY$ ingegeven waarde wordt alleen in de werkruimte gebruikt, zonder iets te wijzigen aan het BASIC- of variabelengeheugen.

Ook hier zal een voorbeeldje meer duidelijk maken dan een stuk tekst:

```
  10 STOP
9990 CLS
9991 PRINT "Verwijder de EAR-plu
g"'"Steek MIC-plug in recorder"
9992 SAVE "basprog" LINE 10
9993 CLS
9994 PRINT "VERIFY gewenst? (j/n
)"
9995 IF INKEY$="" THEN GO TO 999
5
9996 IF INKEY$<>"j" THEN STOP
9997 PRINT "EAR-plug in recorder
"'"Spoel cassette terug"'"Start
cassetterecorder"
9998 VERIFY "basprog"
9999 STOP
```

Na het geven van de opdracht GO TO 9990 wordt het BASIC-programma naar cassette geschreven. Zodra dit is gedaan zal het programma op regel 9994 vragen of een VERIFY gewenst is. Is mijn antwoord daarop 'j', dan word ik door regel 9997 erop opmerkzaam gemaakt dat ik de connectors moet verwisselen, de cassette moet terugspoelen en de cassette moet starten. Regel 9998 voert dan de VERIFY uit.

Probleem:
Na het laden van een programma vergeet ik vaak de cassetterecorder te stoppen, waarschijnlijk omdat ik zo graag wil beginnen met het uitvoeren van het programma.

Oplossing:
Programmeer behalve een LOAD-instructie ook een hoorbaar waarschuwingssignaal.

Ook hiervan nog een voorbeeldje, waarbij ik u erop opmerkzaam maak dat alleen de LOAD-instructies met de daaropvolgende BEEP in het voorbeeld zijn opgenomen.

```
9990 LOAD "a" CODE 60000,500
9991 LOAD "b" CODE 60300,5
9992 CLS
9993 PRINT FLASH 1;"STOP DE RECO
RDER"
9994 FOR a=1 TO 5
9995 BEEP 1,1
9996 PAUSE 20
9997 NEXT a
9998 CLS
9999 GO TO 10
```

Hopelijk heeft u nu, aan de hand van de door mij ondervonden problemen en de daarbij gegeven suggesties voor oplossingen daarvan, een aantal ideeën opgedaan voor de oplossing van uw eigen problemen op het gebied van laden en teruglezen van cassette. Had u toch al geen problemen, dan kunt u misschien uw programma's nog wat uitbreiden met enkele van de hier gegeven oplossingen. U zult dan ook in de toekomst geen problemen ondervinden.

3. Machinetaalinvoerprogramma

De meeste programma's in de volgende hoofdstukken maken gebruik van een aantal korte machinetaalroutines. Deze routines zullen op de één of andere manier ingegeven moeten worden. Om ze als hexadecimale codes te kunnen ingeven, is een speciaal programma nodig.
Misschien hebt u al een dergelijk programma en bent u over de werking van dat programma tevreden. Als dat zo is, dan hoeft u het in dit hoofdstuk te behandelen programma niet te gebruiken. Leest u echter toch dit hoofdstuk eens. Misschien ontdekt u nog een paar aardige routines, die u ook in uw eigen programma kunt gebruiken.
Indien u echter nog niet de beschikking hebt over een programma waarmee u een machinetaalroutine in het geheugen kunt zetten, dan zal het programma 'machtaal' u zeker van pas komen.
De in dit boek gebruikte machinetaalroutines zijn over het algemeen zo kort dat het niet nodig is voor deze routines een assembler te gebruiken. Het gaat vaak om slechts enkele instructies. Bovendien zijn het bijna altijd dezelfde instructies, zodat u al na enkele routines uit het hoofd nieuwe routines kunt maken.

3.1 Programmabeschrijving

Met het programma 'machtaal' kunt u machinetaalroutines schrijven, uitbreiden en veranderen. Het programma bestaat uit vijf hoofddelen. Hierna volgt een korte beschrijving van ieder van die hoofddelen.

Machinetaal programmeren
Dit is het meest uitgebreide deel van het programma. Dit deel staat u toe hexadecimale codes in te voeren. Ook kunt u reeds in het geheugen staande bytes zien en eventueel veranderen. U kunt bytes tussenvoegen en weghalen. Sommige functies zijn doordat ze in BASIC zijn geschreven tamelijk langzaam. Over het geheel genomen werkt dit programma echter snel genoeg. Bedieningsgemak stond bij het ontwerpen van dit programma voorop.

Uitvoeren van het machinetaalprogramma
Dit deel van het programma draagt er zorg voor dat het machinetaalprogramma dat u zojuist hebt geschreven wordt uitgevoerd.

Op cassette bewaren
Deze functie zal het zojuist door u ingegeven machinetaalprogramma op de juiste wijze naar cassette schrijven.

Van cassette inlezen
Het teruglezen van machinetaalprogramma's die ooit op cassette waren geschreven wordt met dit deel van het programma gedaan.

Back-up van 'machtaal'
Deze functie schrijft het hele programma 'machtaal' naar cassette. Hiermee kunt u op eenvoudige wijze een back-up van uw programma maken.

Het programma 'machtaal' dient te worden gestart op regel 10. U wordt dan gevraagd het startadres voor de machinetaalroutine die u wilt gaan schrijven in te geven. De machinetaalroutine zal boven RAMTOP worden weggeschreven. Daar RAMTOP waarschijnlijk op een ander adres wijst dan het adres waar u uw programma wilt zetten, zal RAMTOP op een andere waarde dienen te worden gezet. Dit kan vrij eenvoudig worden gedaan met een CLEAR-instructie. Deze instructie zal echter alle variabelen wissen, dus ook het even daarvoor door u ingegeven startadres. Vandaar dat we na de CLEAR het startadres weer in de variabele 's' moeten zien te krijgen met behulp van PEEK-instructies.
Daarna volgt nog een controle of er wel genoeg geheugenruimte vrij is tussen RAMTOP en UDG. Indien die ruimte kleiner is dan 128 bytes, dan krijgt u de keus om door te gaan of een nieuw startadres te kiezen.
Is alles naar wens dan zult u in het hoofdmenu dat op regel 100 begint terechtkomen. In dat deel van het programma vindt u de hiervoor beschreven vijf hoofddelen terug. Het genereren van het hoofdmenu en het afhankelijk van het ingegeven functienummer wegspringen naar een hoofddeel van het programma is zo recht toe recht aan dat er geen verdere toelichting nodig is.

```
 100>REM **********************
 101 REM *     hoofdmenu        *
 102 REM **********************
 110 CLS
 120 PRINT "1 - machinetaalprogr
ammeren"
 130 PRINT "2 - uitvoeren mt-pro
gramma"
 140 PRINT "3 - op cassette bewa
ren"
 150 PRINT "4 - van cassette inl
ezen"
 160 PRINT "5 - back-up ""machta
al"""
 170 INPUT "kies gewenste functi
enummer ";f
 180 IF f<1 OR f>5 THEN  GO TO 1
70
 190 PRINT AT f-1,0; FLASH 1; OV
ER 1;" "
 200 PAUSE 60: CLS
 210 GO TO 3000+f*1000
```

3.2 Machinetaal programmeren

Drukt u op het hoofdmenu functienummer 1 in, dan zult u een sprong naar regel 4000 veroorzaken. Op regel 4000 begint de routine 'invoeren/wijzigen machinetaalprogramma'. Zoals u ziet is deze routine op zich tamelijk kort. Er staan echter erg veel GO SUB-instructies in.

```
4000>REM **********************
4001 REM * invoeren / wijzigen *
4002 REM * machinetaalprogramma*
4003 REM **********************
4010 CLS
4060 LET c=0: LET mc=c
4070 DIM h$(2)
4080 GO SUB 700
4090 GO SUB 1700
4130 GO SUB 2100
4135 GO SUB 2500: PRINT AT rn,kp
;h$: GO SUB 1700
4140 PRINT AT 21,16; FLASH 1;" B
YTE/OPDRACHT  "
4150 IF INKEY$<>"" THEN  GO TO 4
150
4160 IF INKEY$="" THEN  GO TO 41
60
4170 LET h$(1)=INKEY$
4180 IF h$(1)="-" THEN  GO SUB 9
00: GO TO 4140
4190 IF h$(1)="+" THEN  GO SUB 1
100: GO TO 4140
4200 IF h$(1)="t" THEN  GO SUB 2
300: GO TO 4140
4210 IF h$(1)="w" THEN  GO SUB 2
700: GO TO 4140
4220 IF h$(1)="h" THEN  GO SUB 1
900: GO TO 4140
4230 IF h$(1)="*" THEN  GO TO 10
0
4240 IF INKEY$<>"" THEN  GO TO 4
240
4250 IF INKEY$="" THEN  GO TO 42
50
4260 LET h$(2)=INKEY$
4270 GO SUB 300
4280 GO TO 4140
```

We zullen nu deze routine stap voor stap doornemen.
Allereerst wordt het scherm gewist en worden een aantal variabelen geïnitialiseerd. Om de functie van de variabelen te kunnen begrijpen moet u even weten hoe de organisatie van het programma is. Het programma neemt een stuk RAM-geheugen van 128 bytes en gaat, bij alles wat er binnen dat geheugengebied gebeurt, alleen uit van dat RAM-gebied. Het heeft daarom een pointer nodig die bijhoudt met welk byte er momenteel moet worden gewerkt. Diezelfde pointer kan ook worden gebruikt om te controleren of het gewenste byte nog binnen het gebied van 128 bytes ligt. Voor het beeldscherm wordt die pointer ook gebruikt. Om te bepalen waar de cursor moet worden geschreven, wordt de cursor-positie met een formule, waarin 'c' de variabele is, berekend.
Iedere keer dat er een byte binnen het gebied van 128 bytes wordt beschreven, wordt de waarde van 'c' naar 'mc' geschreven.

De functie van 'mc' is om bij te houden welk byte het laatste van het programma is. Zo zal 'mc' ook worden verhoogd als u een byte gaat tussenvoegen.
De array h$(2) zal worden gebruikt voor het converteren van een decimale waarde naar een hexadecimale waarde.
Nu het scherm schoon is en de variabelen zijn geïnitialiseerd, kan worden begonnen met het opbouwen van het plaatje op het beeldscherm. Dit opbouwen wordt in een aantal subroutines, die we hierna zullen behandelen, gedaan.

```
 700>REM ************************
 701 REM *   opmaak van het     *
 702 REM *     beeldscherm      *
 703 REM ************************
 710 CLS : BORDER 5
 720 PLOT 52,32: DRAW 0,143
 730 PLOT 0,167: DRAW 255,0
 740 PLOT 0,31: DRAW 255,0
 750 PRINT AT 0,0;"adres";AT 0,8
;"+0 +1 +2 +3 +4 +5 +6 +7"
 760 PRINT AT 19,0;"-=terug","+=
vooruit","t=tussenvoegen","w=weg
halen","*=hoofdmenu"
 770 RETURN
```

Eerst wordt er een aantal niet variabele gegevens naar het beeldscherm geschreven en getekend. Dit kunt u zien in de op regel 700 beginnende subroutine. Zijn deze vaste gegevens eenmaal op het scherm geschreven, dan wordt de routine op regel 1700 gestart. Met behulp van 'c' berekent deze routine de plaats van de cursor op het scherm en schrijft de cursor (twee bytes lang) op die gevonden plaats.

```
1700>REM ************************
1701 REM * nwe reg/kar-positie *
1702 REM ************************
1710 LET rn=INT (c/8)+2
1720 LET kp=(c-8*INT (c/8))*3+8
1730 PRINT AT rn,kp; FLASH 1; OV
ER 1;"  "
1740 RETURN
```

Vervolgens wordt de subroutine op adres 2100 gestart. Deze subroutine genereert de adressen die in de linker kolom van het beeldscherm moeten worden geschreven.
In het begin van het programma werd u gevraagd het startadres op te geven. Dit startadres, dat in variabele 's' werd opgeslagen, wordt nu gebruikt bij het genereren van de adressen.

```
2100>REM ************************
2101 REM * beeldschermadressen *
2102 REM ************************
2105 DIM x$(4)
2110 FOR a=0 TO 15
2120 LET h=s+a*8
```

```
2130 LET h1=INT (h/4096)
2140 LET h2=INT ((h-h1*4096)/256
)
2150 LET h3=INT ((h-h1*4096-h2*2
56)/16)
2160 LET h4=h-h1*4096-h2*256-h3*
16
2170 IF h1<=9 THEN  LET x$(1)=CH
R$ (h1+48)
2180 IF h1>9 THEN  LET x$(1)=CHR
$ (h1+55)
2190 IF h2<=9 THEN  LET x$(2)=CH
R$ (h2+48)
2200 IF h2>9 THEN  LET x$(2)=CHR
$ (h2+55)
2210 IF h3<=9 THEN  LET x$(3)=CH
R$ (h3+48)
2220 IF h3>9 THEN  LET x$(3)=CHR
$ (h3+55)
2230 IF h4<=9 THEN  LET x$(4)=CH
R$ (h4+48)
2240 IF h4>9 THEN  LET x$(4)=CHR
$ (h4+55)
2270 PRINT AT 2+a,1;x$
2280 NEXT a
2290 RETURN
```

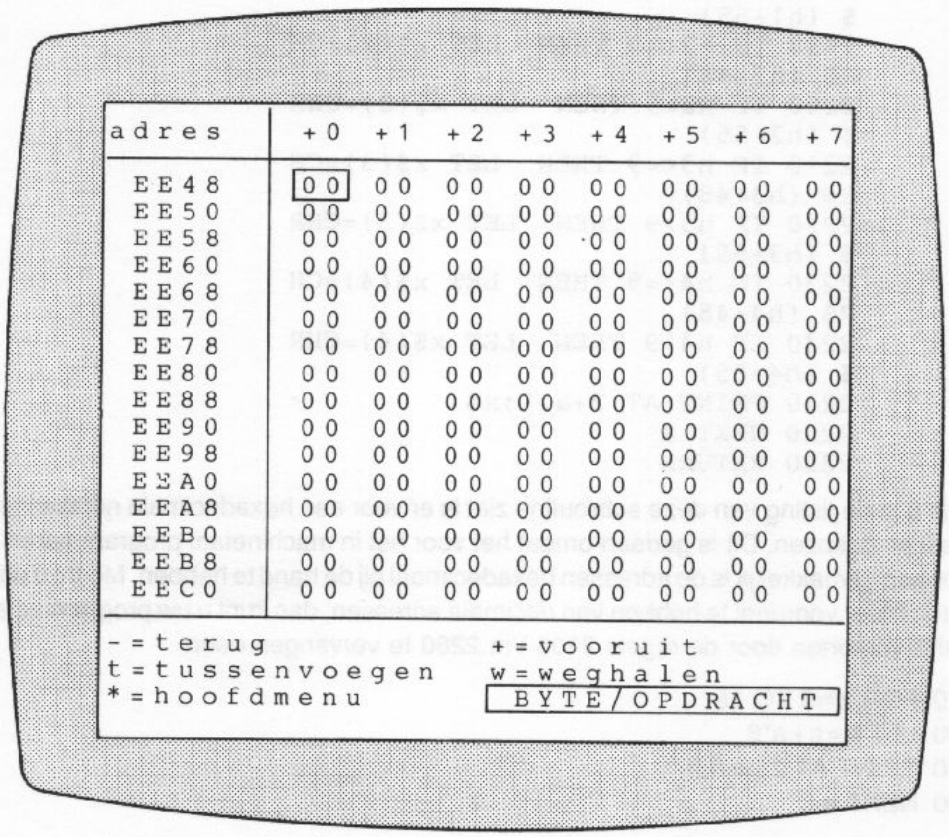
Zoals u in de listing van deze subroutine ziet is er voor een hexadecimale notatie van de adressen gekozen. Dit is gedaan omdat het voor het in machinetaal programmeren van sprongen gemakkelijk is de adressen hexadecimaal bij de hand te hebben. Mocht u echter menen meer voordeel te hebben van decimale adressen, dan kunt u uw programma aanzienlijk verkorten door de regels 2105 t/m 2280 te vervangen door:

```
2110 FOR a=0 TO 15
2120 LET h=s+a*8
2130 PRINT AT 2+a,0;h
2140 NEXT a
```

Voordat u bytes kunt gaan ingeven moet er nog een subroutine worden uitgevoerd: de routine die op regel 2500 start. Deze routine zorgt ervoor dat de bytes van het RAM-geheugen naar het beeldscherm worden geschreven. Op zichzelf is deze routine erg kort, maar er wordt vanuit deze subroutine weer een aantal andere subroutines aangeroepen. Het gevolg is dat deze routine erg langzaam is geworden. Daar deze routine echter niet vaak wordt gebruikt levert hij op het totaal weinig vertraging.

```
2500>REM **********************
2501 REM *  programma-listing  *
2502 REM **********************
2510 IF c>=127 THEN  LET c=0: RE
TURN
2520 GO SUB 500
2530 PRINT AT rn,kp;h$
2540 LET c=c+1
2550 GO SUB 1700
2560 GO TO 2510
```

Zoals u ziet wordt in deze routine een lus uitgevoerd die net zo lang doorgaat tot de cursorpositie aangeeft dat het 128e byte is bereikt. In dat geval wordt er een RETURN uitgevoerd. Is dat laatste byte nog niet bereikt, dan wordt de subroutine op regel 500 uitgevoerd. Daarin wordt het door de cursor aangewezen adres uitgelezen en als een hexadecimale waarde in h$ gezet. Regel 2530 drukt h$ af op het beeldscherm, op de plaats waar de cursor staat. Vervolgens wordt de cursor-positie 1 verhoogd. De subroutine op regel 1700 berekent de nieuwe coördinaten voor de cursor op het beeldscherm en schrijft de cursor op die nieuw berekende plaats.



Afb. 3-1 Beeldscherm na opstarten 'machtaal'

Nu wordt door regel 4140 de boodschap 'BYTE/OPDRACHT' rechts onderaan het scherm afgedrukt. U kunt nu beginnen met het invoeren van bytes (hexadecimaal) of een commando. De toegestane commando's staan onderaan het beeldscherm weergegeven. Indien u een karakter ingeeft dat niet een geldig commando is of dat niet een geldige hexadecimale waarde is, dan zult u een boodschap zien verschijnen die daarvan melding maakt.

Voor wat betreft de hexadecimale waarden nog even het volgende:

U mag zowel hoofdletters als kleine letters gebruiken. Het programma zet ze allemaal om naar hoofdletters.

De eigenlijke invoer-routine is omwille van het bedieningsgemak niet met een INPUT-instructie uitgevoerd. Er is hier gekozen voor een constructie met de functie INKEYS. U

hoeft nu de ingegeven bytes niet met ENTER af te sluiten. Zodra u de byte of het commando hebt ingetypt wordt hij geaccepteerd door het programma en naar het geheugen weggeschreven.
Indien u een commando had ingegeven wordt dit commando direct uitgevoerd. Op regel 4220 ziet u dat er een commando is dat niet op uw scherm wordt afgebeeld. Dit is het commando 'h', help. Dit commando zal later nog verder worden behandeld, maar hier wil ik alvast een kanttekening maken. Wie in het bezit is van een 16K ZX Spectrum doet er goed aan dit commando en alle daarbij behorende routines weg te laten. Dat geeft namelijk een aanzienlijke geheugenbesparing.
Laten we er even van uitgaan dat u zojuist een byte hebt ingetypt. Dat wil dan zeggen dat u in h$ twee karakters hebt geladen. De subroutine op regel 300 zal nu controleren of die twee karakters binnen het hexadecimale getallenstelsel vallen.

```
 300>REM **********************
 301 REM * controleren en con- *
 302 REM * verteren hex.input. *
 303 REM * updaten pointers en *
 304 REM * naar geheugen/beeld *
 305 REM * schrijven van data. *
 306 REM **********************
 310 LET msd=CODE h$(1): LET lsd
=CODE h$(2)
 320 IF msd>47 AND msd<58 THEN
LET msd=msd-48
 330 IF msd>64 AND msd<71 THEN
LET msd=msd-55
 340 IF msd>96 AND msd<103 THEN
 LET msd=msd-87
 350 IF lsd>47 AND lsd<58 THEN
LET lsd=lsd-48
 360 IF lsd>64 AND lsd<71 THEN
LET lsd=lsd-55
 370 IF lsd>96 AND lsd<103 THEN
 LET lsd=lsd-87
 380 IF lsd>15 OR msd>15 THEN  P
RINT AT 21,16; FLASH 1;"  FOUTE
INGAVE  ": PAUSE 100: PAUSE 100:
 RETURN
 390 POKE (s+c),msd*16+lsd
 400 GO SUB 500
 410 IF c<=127 THEN  PRINT AT rn
,kp;h$
 420 IF c<127 THEN  LET c=c+1
 430 IF mc<c THEN  LET mc=c
 440 GO SUB 1700
 450 RETURN
```

Mocht dat niet het geval zijn, dan zult u de boodschap 'FOUTE INGAVE' rechts onderaan uw beeldscherm zien verschijnen. Deze boodschap verdwijnt na een paar seconden weer om plaats te maken voor de boodschap 'BYTE/OPDRACHT'. Nu kunt u uw fout weer herstellen.
Heeft u een correcte ingave gedaan, dan zal de subroutine op regel 300 die naar het juiste

adres in RAM schrijven. Vervolgens wordt die waarde omgezet naar hexadecimale afdrukbare karakters en naar het beeldscherm geschreven.
De cursor wordt met 1 verhoogd en omdat ons programma nu een byte langer is geworden wordt ook de variabele 'mc' gelijk gemaakt met 'c'. Met de routine op regel 1700 wordt nu de cursor op de volgende beeldschermpositie afgedrukt. Hierna keren we terug uit de subroutine van regel 300 naar het invoer- en wijzigdeel van het programma. We zijn dan aangeland op regel 4280. Daar zien we een GO TO naar regel 4140, waar we weer in staat worden gesteld het volgende byte of commando in te geven.
We zullen nu eens zien wat er gebeurt als we de commando's gaan ingeven.

Het commando '-' veroorzaakt een GO SUB 900.

```
900>REM ***********************
901 REM *  opdracht = terug   *
902 REM ***********************
910 IF c<=0 THEN  RETURN
920 GO SUB 500
930 PRINT AT rn,kp;h$
940 LET c=c-1
950 GO SUB 1700
960 RETURN
```

Deze korte routine laat de cursor een positie naar links gaan. Allereerst wordt er gekeken of de cursor al helemaal linksboven is aangekomen. Indien dat zo is dan heeft het geen zin de routine verder uit te voeren en gaan we terug met een RETURN. Waren we nog niet helemaal linksboven aangekomen, dan maken we de variabele 'c' een positie lager, berekenen de nieuwe beeldschermpositie voor de cursor en gaan terug met een RETURN.

```
1100>REM ***********************
1101 REM * opdracht = vooruit  *
1102 REM ***********************
1110 IF c>=127 THEN  RETURN
1120 GO SUB 500
1130 PRINT AT rn,kp;h$
1140 LET c=c+1
1160 GO SUB 1700
1170 RETURN
```

Indien we het commando '+' hadden gegeven zouden we naar de subroutine op regel 1100 zijn gesprongen.
Daar wordt eerst gekeken of de cursor al aan het eind van het programma is aangekomen, rechts onderaan het beeldscherm. Is dit niet het geval dan wordt de cursor met 1 verhoogd. De nieuwe beeldschermpositie van de cursor wordt berekend en met RETURN wordt teruggekeerd.
Daar het hier alleen om een cursor-beweging naar rechts gaat en er geen sprake van is dat het programma langer wordt hoeft de variabele 'mc' niet bijgewerkt te worden.

Met de opdracht 't' kunt u in een reeds in het geheugen staand programma ruimte maken voor een extra byte. Deze ruimte zal ontstaan op de plaats waar de cursor zich ten tijde van het commando bevindt.

```
2300>REM ************************
2301 REM * opdracht=tussenvoeg *
2302 REM ************************
2310 IF mc>=127 THEN  GO TO 2410
2320 FOR a=mc TO c STEP -1
2340 POKE (s+a+1),PEEK (s+a)
2350 NEXT a
2360 LET sc=c: LET mc=mc+1
2370 GO SUB 2500
2380 PRINT AT rn,kp;h$: LET c=sc
: GO SUB 1700
2400 RETURN
2410 PRINT AT 21,16; FLASH 1;"
GEEN RUIMTE   "
2420 PAUSE 0: PAUSE 100
2430 PRINT AT 21,16; FLASH 1;" B
YTE/OPDRACHT  "
2440 RETURN
```

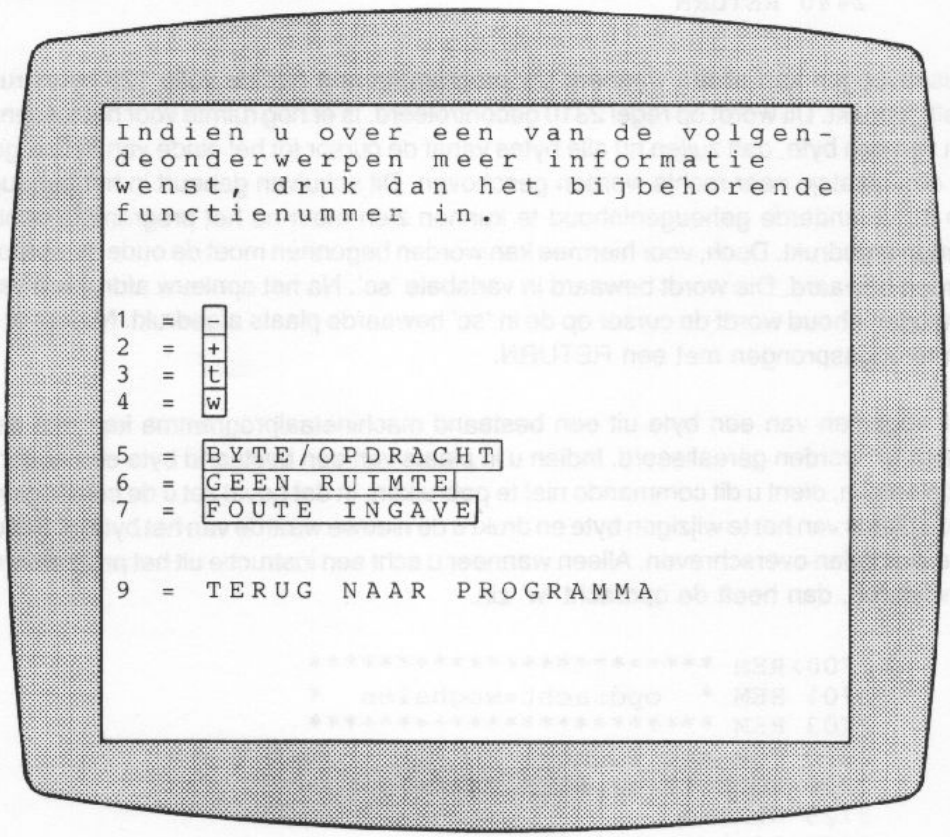
Tussenvoegen kan alleen wanneer uw programma nog niet de volle 128 bytes ruimte heeft gebruikt. Dit wordt op regel 2310 gecontroleerd. Is er nog ruimte voor het tussenvoegen van een byte, dan zullen nu alle bytes vanaf de cursor tot het einde van het programma een plaatsje naar rechts worden geschoven. Dit schuiven gebeurt in het geheugen. Om de veranderde geheugeninhoud te kunnen zien moet nu het programma opnieuw worden afgedrukt. Doch, voor hiermee kan worden begonnen moet de oude cursorpositie worden bewaard. Die wordt bewaard in variabele 'sc'. Na het opnieuw afdrukken van de geheugeninhoud wordt de cursor op de in 'sc' bewaarde plaats afgedrukt. Nu kan er worden teruggesprongen met een RETURN.

Het weghalen van een byte uit een bestaand machinetaalprogramma kan met de opdracht 'w' worden gerealiseerd. Indien u in plaats van een bestaand byte een ander byte wilt plaatsen, dient u dit commando niet te gebruiken. In dat geval zet u de cursor gewoon op de plaats van het te wijzigen byte en drukt u de nieuwe waarde van het byte in. Het oude byte wordt dan overschreven. Alleen wanneer u echt een instructie uit het programma wilt verwijderen, dan heeft de opdracht 'w' zin.

```
2700>REM ************************
2701 REM *  opdracht=weghalen  *
2703 REM ************************
2710 FOR a=c TO mc-1
2720 POKE (s+a),PEEK (s+a+1)
2725 NEXT a
2730 POKE (s+mc),00
2735 LET sc=c: LET mc=mc-1
2740 GO SUB 2500
2743 LET c=sc
2745 PRINT AT rn,kp;h$: GO SUB 1
700
2750 RETURN
```

Weghalen van bytes kan altijd. Dit wordt dan ook zonder meer gedaan. Op de regels 2710 tot en met 2725 worden alle bytes vanaf het einde van het programma tot aan de cursor een plaatsje naar links geschoven. Nu staat het laatste byte van uw programma twee keer in het geheugen, daarom wordt het laatste byte nu gewist en vervangen door '00'. De variabele 'mc' wordt met 1 verlaagd. Uw programma is immers 1 byte korter geworden. Hierna wordt de veranderde geheugeninhoud opnieuw afgedrukt op het beeldscherm. Ook hier geldt weer dat we de oude cursorpositie willen bewaren. Dit wordt weer met de variabele 'sc' gedaan.

De opdracht 'h' is speciaal ingebouwd om het programma wat gebruiksvriendelijker te maken. Om te voorkomen dat men steeds in een boek moet gaan naslaan wat ook alweer de betekenis van een bepaalde opdracht of boodschap was, kan die betekenis via het hulpcommando op het beeldscherm worden tevoorschijn getoverd. Welke informatie allemaal met 'h' kan worden opgeroepen ziet u in afbeelding 3.2 van het beeldscherm met daarop het menu dat u krijgt.



Afb. 3-2 Beeldscherm met 'help'-menu

Omdat we in de voorgaande tekst al hebben gezien wat de betekenis van de verschillende opdrachten is, en wat de oorzaak van de boodschappen is, zullen we dat hier niet herhalen. Wel zullen we eens nagaan hoe de verdere afwikkeling van het programma is als we een hulppagina opvragen.
Door het ingeven van de opdracht 'h' kwamen we in de subroutine op regel 1900 terecht. Deze subroutine drukt een menu als hiervoor weergegeven op het beeldscherm af. We kunnen nu een functie kiezen.

```
1900>REM **********************
1901 REM *    hulp-pagina's    *
1902 REM **********************
1910 CLS : PRINT "Indien u over
een van de volgen-deonderwerpen
meer informatie  wenst, druk da
n het bijbehorendefunctienummer
in."
1920 PRINT AT 7,0;"1 = -";AT 8,0
;"2 = +";AT 9,0;"3 = t";AT 10,0;
"4 = w";AT 12,0;"5 = BYTE/OPDRAC
HT";AT 13,0;"6 = GEEN RUIMTE";AT
 14,0;"7 = FOUTE INGAVE";AT 17,0
;"9 = TERUG NAAR PROGRAMMA"
1930 INPUT f
1940 GO SUB 2900+100*f
2000 LET sc=c: LET c=0
2010 GO SUB 700
2030 GO SUB 1700
2040 GO SUB 2100
2050 GO SUB 2500: PRINT AT rn,kp
;h$
2060 LET c=sc: GO SUB 1700
2070 RETURN
```

Met functienummer 9 keren we terug naar waar we vandaan kwamen. Hoe gaat dit in zijn werk? Welnu, de ingegeven 9 zorgt ervoor dat er een sprong naar de subroutine op regel 2900+100*9=3800 wordt gemaakt. Op regel 3800 staat een RETURN-instructie, zodat we weer terugkeren in de routine die op regel 1900 start. Hier wordt de oude inhoud van het beeldscherm, zoals die was op het moment dat we de opdracht 'h' ingaven, weer afgedrukt. Bij alle andere functienummers springen we op dezelfde manier uit de menu-routine naar een subroutine. Alleen staan er in die subroutines behalve een RETURN-instructie ook nog een aantal andere instructies. Die instructies zorgen ervoor dat er informatie over de gekozen functie naar het beeldscherm wordt geschreven. Na dat afdrukken wordt naar regel 3010 gesprongen, waar met een INKEY$-constructie het indrukken van de ENTER-toets wordt afgewacht. Is die ENTER-toets gesignaleerd, dan wordt met een RETURN naar de menu-routine teruggekeerd. Daar wordt weer het oude beeldscherm, zoals dat was bij het indrukken van 'h', afgedrukt.
Hierna kan men weer opdrachten of bytes ingeven.
De enige opdracht die we nog niet hebben gezien is de '*'. Hiermee kunnen we terugkeren naar het hoofdmenu. Dit wordt eenvoudig met een GO TO 100 gerealiseerd. Nu we weer terug zijn in het hoofdmenu, zullen we eens zien wat de overige mogelijkheden van het programma zijn.

3.3 Uitvoeren machinetaalprogramma

Functienummer 2 van het hoofdmenu veroorzaakt een sprong naar regel 5000 van het programma. Daar start de routine die het met de vorige functie ingegeven machinetaalprogramma uitvoert.

```
5000>REM ************************
5001 REM * uitvoeren mt-progr. *
5002 REM ************************
5010 RANDOMIZE USR s
5020 PRINT "machinetaalprogramma
 uitgevoerd.": PAUSE 100
5030 GO TO 100
```

U ziet dat dit wel een erg korte routine is. Op regel 5010 wordt de machinetaalroutine gestart. Het eerste adres van dat machinetaalprogramma is het adres dat u had opgegeven bij het starten van het programma 'machtaal'.
Zodra de routine goed is uitgevoerd, wordt dit door een boodschap op het beeldscherm te kennen gegeven. Deze boodschap blijft slechts enkele seconden staan. Daarna wordt teruggesprongen naar het hoofdmenu.
Het is misschien goed er hier nog eens de aandacht op te vestigen dat een klein foutje in een machinetaalprogramma het verlies van de hele geheugeninhoud kan veroorzaken. Om te voorkomen dat u alles opnieuw moet gaan intypen, doet u er goed aan voor het uitvoeren van het machinetaalprogramma dat programma op cassette te bewaren. Speciaal om u dat gemakkelijk te maken heb ik daarvoor een routine in dit programma ingebouwd. Deze routine kunt u met functienummer 3 van het hoofdmenu kiezen.

3.4 Op cassette bewaren

Met deze functie van het programma kunt u een door u ingevoerd machinetaalprogramma naar cassette schrijven. U roept deze functie aan door op het hoofdmenu nummer 3 in te typen.

```
6000>REM ************************
6001 REM * op cassette bewaren *
6003 REM ************************
6010 CLS : PRINT "Verwijder de E
AR-plug en steek  de MIC-plug in
 de recorder.     Zet de recorde
r op opname."
6020 SAVE "machcode"CODE s,mc
6030 CLS : PRINT "Spoel de casse
tte terug en zet  de EAR-plug we
er in de recorder,om te controle
ren of het pro-   gramma goed is
 opgenomen."
6040 VERIFY "machcode"CODE : GO
TO 100
```

Daar ik zelf nog wel eens vergeet de stekkers in de cassetterecorder te steken, leek het me een goed idee om het programma een waarschuwing te laten geven. Dit wordt op regel 6010 gedaan. Vervolgens wordt op regel 6020 het machinetaalprogramma naar cassette geschreven. Het bij het begin van het programma door u opgegeven startadres wordt ook hier als eerste adres genomen. De lengte van het weg te schrijven blok wordt uit de variabele 'mc' gehaald. Daarin hebben we immers steeds het adres van het laatste byte van het programma bewaard.

Om er zeker van te zijn dat het programma ook inderdaad goed is weggeschreven, is nog een VERIFY opgenomen. Zodra is geconstateerd dat het machinetaalprogramma goed op de band staat, wordt teruggesprongen naar het hoofdmenu.
Mocht u door een leesfout tijdens het verifiëren een systeemboodschap krijgen, dan kunt u met GO TO 100 teruggaan naar het hoofdmenu. U dient dan wel het machinetaalprogramma opnieuw weg te schrijven.
Zowel in deze functie als in de hiernavolgende functie, van cassette lezen, wordt een vaste naam voor het machinetaalprogramma gebruikt. Dit heeft tot gevolg dat al uw machinetaalprogramma's dezelfde naam krijgen. Indien u dit niet wilt, dan kunt u op eenvoudige wijze een INPUT-instructie in de routine voor regel 6020 opnemen, waarmee u naar de naam van de weg te schrijven routine vraagt. Regel 6020 zelf moet dan nog wel worden aangepast, bijv. SAVE n$ CODE s,mc.

3.5 Van cassette lezen

Functienummer 4 van het hoofdmenu veroorzaakt een sprong naar de routine op regel 7000. Met deze routine kan een eerder op cassette opgeslagen machinetaalprogramma weer terug in het geheugen worden geladen.

```
7000>REM ************************
7001 REM * van cassette lezen  *
7002 REM ************************
7010 CLS : PRINT "start de casse
tterecorder."
7020 LOAD "machcode"CODE
7030 GO TO 100
```

Na u te hebben gevraagd de cassetterecorder te starten wordt onmiddellijk begonnen met het uitlezen van de cassette. Voor regel 7020 zijn een aantal mogelijkheden. Zoals hij hier in het programma staat, zal altijd naar een routine met de naam 'machcode' worden gezocht. Door echter de tekst tussen de aanhalingstekens weg te laten zal het eerste machinetaalprogramma dat wordt gevonden worden geladen.
Een derde mogelijkheid, die te overwegen is wanneer u ook het naar cassette wegschrijven heeft aangepast, is om met een INPUT-instructie naar de te zoeken naam te vragen. Hiertoe dient u dan voor regel 7020 een INPUT te plaatsen en regel 7020 zelf aan te passen, bijv. LOAD n$ CODE.

3.6 Back-up van machtaal

De laatste functie die u van het hoofdmenu kunt kiezen is functienummer 5, het maken van een back-up van het programma 'machtaal'.

```
8000>REM ************************
8001 REM * backup van machtaal *
8003 REM ************************
8010 CLS : PRINT "verwijder de E
```

```
AR-plug en steek  de MIC-plug in
 de recorder.     Zet de recorde
r op opnemen."
8020 SAVE "machtaal" LINE 10
8030 CLS : PRINT "Spoel de casse
tte terug en zet  de EAR-plug we
er in de recorder,om te controle
ren of het pro-   gramma goed is
 opgenomen."
8040 VERIFY "machtaal": GO TO 10
0
```

Ook hier wordt weer een waarschuwing gegeven om de stekkers op de juiste manier in de cassetterecorder te steken. Vervolgens wordt op regel 8020 het programma naar cassette geschreven. Omdat het handig is dit programma onmiddellijk, na het later weer te laden, te laten starten op regel 10, is de SAVE-instructie hier voorzien van de toevoeging LINE 10.

Daar mag worden aangenomen dat men na het op cassette bewaren van het programma niets meer met het programma wenst te doen, is de instructie STOP aan de SAVE-regel toegevoegd. Mocht u in de plaats daarvan liever een andere opdracht hebben, dan kunt u het programma natuurlijk dienovereenkomstig wijzigen. Een suggestie is om er dan een GO TO 10 neer te zetten.

Hiermee is het gehele programma besproken. Daar echter in deze bespreking niet alle BASIC-regels van het programma zijn afgebeeld, volgt hierna nog een complete listing van het programma. Ook zou ik de bezitters van een 16K ZX Spectrum er nog eens op willen wijzen dat terwille van het spaarzaam geheugengebruik de hulpfunctie kan worden weggelaten. Bovendien kunnen de REM-statements uit het programma worden verwijderd.

De programmalisting

```
    1 REM ************************
    2 REM * machinetaal invoer- *
    3 REM *  print- en wijzig-  *
    4 REM *      utility.       *
    5 REM ************************
    6 REM *
    7 REM *** Naam = machtaal
    8 REM *** Copyright 1983
    9 REM
   10 CLS : PRINT "Geeft u nu het
 door u gewenste  startadres van
 het te laden of  te schrijven m
achinetaalprogram-ma in. Dit sta
rtadres zal wordengebruikt om RA
MTOP overeenkom-  stig te zetten
."
   20 PRINT AT 10,0;"Mocht u tijd
ens het invoeren vanbytes of opd
rachten meer willen weten over d
e mogelijkheden, dankunt u in pl
```

```
aats van een byte ofopdracht de
letter ""h"" indrukken"'""U kunt
 altijd weer terugkeren  naar d
e plaats waar u de ""h""    hebt
 ingedrukt."
  30 INPUT "startadres? ";s
  40 LET RAMTOP=s-1
  50 CLEAR RAMTOP
  60 LET RAMTOP=PEEK 23730+256*P
EEK 23731
  70 LET s=RAMTOP+1
  80 LET UDG=PEEK 23675+256*PEEK
 23676
  90 IF s+128>=UDG THEN  PRINT "
U hebt nog ";UDG-s;" bytes ruimt
e.","startadres wijzigen? (j/n)
": INPUT a$: IF a$="j" THEN  GO
TO 10
 100 REM **********************
 101 REM *    hoofdmenu      *
 102 REM **********************
 110 CLS
 120 PRINT "1 - machinetaalprogr
ammeren"
 130 PRINT "2 - uitvoeren mt-pro
gramma"
 140 PRINT "3 - op cassette bewa
ren"
 150 PRINT "4 - van cassette inl
ezen"
 160 PRINT "5 - back-up ""machta
al"""
 170 INPUT "kies gewenste functi
enummer ";f
 180 IF f<1 OR f>5 THEN  GO TO 1
70
 190 PRINT AT f-1,0; FLASH 1; OV
ER 1;" "
 200 PAUSE 60: CLS
 210 GO TO 3000+f*1000
 300>REM **********************
 301 REM * controleren en con- *
 302 REM * verteren hex.input. *
 303 REM * updaten pointers en *
 304 REM * naar geheugen/beeld *
 305 REM * schrijven van data. *
 306 REM **********************
 310 LET msd=CODE h$(1): LET lsd
=CODE h$(2)
 320 IF msd>47 AND msd<58 THEN
LET msd=msd-48
 330 IF msd>64 AND msd<71 THEN
LET msd=msd-55
 340 IF msd>96 AND msd<103 THEN
 LET msd=msd-87
```

```
 350 IF lsd>47 AND lsd<58 THEN
LET lsd=lsd-48
 360 IF lsd>64 AND lsd<71 THEN
LET lsd=lsd-55
 370 IF lsd>96 AND lsd<103 THEN
 LET lsd=lsd-87
 380 IF lsd>15 OR msd>15 THEN  P
RINT AT 21,16; FLASH 1;"  FOUTE
INGAVE  ": PAUSE 100: PAUSE 100:
 RETURN
 390 POKE (s+c),msd*16+lsd
 400 GO SUB 500
 410 IF c<=127 THEN  PRINT AT rn
,kp;h$
 420 IF c<127 THEN  LET c=c+1
 430 IF mc<c THEN  LET mc=c
 440 GO SUB 1700
 450 RETURN
 500 REM ************************
 501 REM * conversie decimaal  *
 502 REM * naar hexadecimaal   *
 503 REM ************************
 510 LET h=PEEK (s+c)
 520 LET h1=INT (h/16)
 530 LET h2=h-h1*16
 540 IF h1<=9 THEN  LET h$(1)=CH
R$ (h1+48)
 550 IF h1>9 THEN  LET h$(1)=CHR
$ (h1+55)
 560 IF h2<=9 THEN  LET h$(2)=CH
R$ (h2+48)
 570 IF h2>9 THEN  LET h$(2)=CHR
$ (h2+55)
 580 RETURN
 700 REM ************************
 701 REM *   opmaak van het    *
 702 REM *     beeldscherm     *
 703 REM ************************
 710 CLS : BORDER 5
 720 PLOT 52,32: DRAW 0,143
 730 PLOT 0,167: DRAW 255,0
 740 PLOT 0,31: DRAW 255,0
 750 PRINT AT 0,0;"adres";AT 0,8
;"+0 +1 +2 +3 +4 +5 +6 +7"
 760 PRINT AT 19,0;"-=terug","+=
vooruit","t=tussenvoegen","w=weg
halen","*=hoofdmenu"
 770>RETURN
 900 REM ************************
 901 REM *  opdracht = terug   *
 902 REM ************************
 910 IF c<=0 THEN  RETURN
 920 GO SUB 500
 930 PRINT AT rn,kp;h$
 940 LET c=c-1
```

```
 950 GO SUB 1700
 960 RETURN
1100 REM ************************
1101 REM * opdracht = vooruit  *
1102 REM ************************
1110 IF c>=127 THEN  RETURN
1120 GO SUB 500
1130 PRINT AT rn,kp;h$
1140 LET c=c+1
1160 GO SUB 1700
1170 RETURN
1700 REM ************************
1701 REM * nwe reg/kar-positie *
1702 REM ************************
1710 LET rn=INT (c/8)+2
1720 LET kp=(c-8*INT (c/8))*3+8
1730 PRINT AT rn,kp; FLASH 1; OV
ER 1;"  "
1740 RETURN
1900 REM ************************
1901 REM *    hulp-pagina's    *
1902 REM ************************
1910 CLS : PRINT "Indien u over
een van de volgen-deonderwerpen
meer informatie   wenst, druk da
n het bijbehorendefunctienummer
in."
1920 PRINT AT 7,0;"1 = -";AT 8,0
;"2 = +";AT 9,0;"3 = t";AT 10,0;
"4 = w";AT 12,0;"5 = BYTE/OPDRAC
HT";AT 13,0;"6 = GEEN RUIMTE";AT
 14,0;"7 = FOUTE INGAVE";AT 17,0
;"9 = TERUG NAAR PROGRAMMA"
1930 INPUT f
1940 GO SUB 2900+100*f
2000 LET sc=c: LET c=0
2010 GO SUB 700
2030 GO SUB 1700
2040 GO SUB 2100
2050 GO SUB 2500: PRINT AT rn,kp
;h$
2060 LET c=sc: GO SUB 1700
2070 RETURN
2100 REM ************************
2101 REM * beeldschermadressen *
2102 REM ************************
2105 DIM x$(4)
2110 FOR a=0 TO 15
2120 LET h=s+a*8
2130 LET h1=INT (h/4096)
2140 LET h2=INT ((h-h1*4096)/256
)
2150 LET h3=INT ((h-h1*4096-h2*2
56)/16)
2160>LET h4=h-h1*4096-h2*256-h3*
16
```

```
2170 IF h1<=9 THEN  LET x$(1)=CH
R$ (h1+48)
2180 IF h1>9 THEN  LET x$(1)=CHR
$ (h1+55)
2190 IF h2<=9 THEN  LET x$(2)=CH
R$ (h2+48)
2200 IF h2>9 THEN  LET x$(2)=CHR
$ (h2+55)
2210 IF h3<=9 THEN  LET x$(3)=CH
R$ (h3+48)
2220 IF h3>9 THEN  LET x$(3)=CHR
$ (h3+55)
2230 IF h4<=9 THEN  LET x$(4)=CH
R$ (h4+48)
2240 IF h4>9 THEN  LET x$(4)=CHR
$ (h4+55)
2270 PRINT AT 2+a,1;x$
2280 NEXT a
2290 RETURN
2300 REM ************************
2301 REM * opdracht=tussenvoeg *
2302 REM ************************
2310 IF mc>=127 THEN  GO TO 2410
2320 FOR a=mc TO c STEP -1
2340 POKE (s+a+1),PEEK (s+a)
2350 NEXT a
2360 LET sc=c: LET mc=mc+1
2370 GO SUB 2500
2380 PRINT AT rn,kp;h$: LET c=sc
: GO SUB 1700
2400 RETURN
2410 PRINT AT 21,16; FLASH 1;"
GEEN RUIMTE   "
2420 PAUSE 0: PAUSE 100
2430 PRINT AT 21,16; FLASH 1;" B
YTE/OPDRACHT  "
2440 RETURN
2500 REM ************************
2501 REM *  programma listing  *
2502 REM ************************
2510 IF c>=127 THEN  LET c=0: RE
TURN
2520 GO SUB 500
2530 PRINT AT rn,kp;h$
2540 LET c=c+1
2550 GO SUB 1700
2560 GO TO 2510
2700 REM ************************
2701 REM *  opdracht=weghalen  *
2703 REM ************************
2710 FOR a=c TO mc-1
2720 POKE (s+a),PEEK (s+a+1)
2725 NEXT a
2730 POKE (s+mc),00
2735 LET sc=c: LET mc=mc-1
```

```
2740 GO SUB 2500
2743 LET c=sc
2745>PRINT AT rn,kp;h$:GO SUB 17
00
2750 RETURN
3000 CLS : PRINT "Met - kunt u d
e cursor naar      links bewegen.
 Zodra u geheel    linksboven ben
t aangekomen stoptde cursor vanz
elf."'''"Druk op ENTER om door t
e gaan."
3010 IF INKEY$<>"" THEN  GO TO 3
010
3020 IF INKEY$="" THEN  GO TO 30
20
3030 IF INKEY$=CHR$ (13) THEN  R
ETURN
3100 CLS : PRINT "Met + kunt u d
e cursor vooruit   bewegen. Bent
u helemaal rechts-onderaan gekom
en, dan stopt de   cursor vanzelf
."'''"Druk op ENTER om door te g
aan.": GO TO 3010
3200 CLS : PRINT "Met t kunt u o
p de plaats van decursor een byt
e tussenvoegen.    Eerst wordt er
 in het geheugen   ruimte gemaakt
, daarna wordt de  nieuwe geheuge
nindeling op het   scherm afgedru
kt. De cursor      blijft op de p
laats waar u het   tussen te voeg
en karakter wilt   plaatsen stils
taan."'''"Druk op  ENTER om door
te gaan."
3210 GO TO 3010
3300 CLS : PRINT "Met w kunt u h
et byte waar met   de cursor naar
 wordt gewezen     uit uw machine
taalprogramma ver-wijderen. Nada
t alle bytes na    het verwijderd
e byte zijn terug-geschoven, wor
dt een nieuwe      listing van uw
 programma gemaakt"'''"Druk op E
NTER om door te gaan.": GO TO 30
10
3400 CLS : PRINT "Het bericht ";
 FLASH 1;"BYTE/OPDRACHT"; FLASH
0;" geeft aan dat u nu een machi
netaal     instructie of een van
de onder   in het beeld gegeven o
pdrachten kunt ingeven."'''"Druk
 op ENTER om door te gaan.": GO
TO 3010
3500 CLS : PRINT "Het bericht ";
 FLASH 1;"GEEN RUIMTE"; FLASH 0;
```

```
" geeft   aan dat u hebt getrach
t een bytetussen te voegen, doch
 dat het  laatste byte van uw pr
ogramma alhet 128-ste byte was.
Er kan dusniets meer worden tuss
engevoegd."'''"Druk op ENTER om
door te gaan.": GO TO 3010
3600>CLS :PRINT "Het bericht ";F
LASH 1;"FOUTE INGAVE";FLASH 0;"
geeft  aan dat u niet een van de
 cij-  fers 0 t/m 9, een van de
lettersA t/m F of a t/m f of een
 geldigcommando hebt ingegeven.
       Na enkele seconden kunt u
 als-  nog de juiste ingave doen
."'''"Druk op ENTER om door te g
aan.":GO TO 3010
3800 RETURN
4000 REM **********************
4001 REM * invoeren / wijzigen *
4002 REM * machinetaalprogramma*
4003 REM **********************
4010 CLS
4060 LET c=0: LET mc=c
4070 DIM h$(2)
4080 GO SUB 700
4090 GO SUB 1700
4130 GO SUB 2100
4135 GO SUB 2500: PRINT AT rn,kp
;h$: GO SUB 1700
4140 PRINT AT 21,16; FLASH 1;" B
YTE/OPDRACHT  "
4150 IF INKEY$<>"" THEN  GO TO 4
150
4160 IF INKEY$="" THEN  GO TO 41
60
4170 LET h$(1)=INKEY$
4180 IF h$(1)="-" THEN  GO SUB 9
00: GO TO 4140
4190 IF h$(1)="+" THEN  GO SUB 1
100: GO TO 4140
4200 IF h$(1)="t" THEN  GO SUB 2
300: GO TO 4140
4210 IF h$(1)="w" THEN  GO SUB 2
700: GO TO 4140
4220 IF h$(1)="h" THEN  GO SUB 1
900: GO TO 4140
4230 IF h$(1)="*" THEN  GO TO 10
0
4240 IF INKEY$<>"" THEN  GO TO 4
240
4250 IF INKEY$="" THEN  GO TO 42
50
4260 LET h$(2)=INKEY$
4270 GO SUB 300
```

```
4280 GO TO 4140
5000 REM ************************
5001 REM * uitvoeren mt-progr. *
5002 REM ************************
5010 RANDOMIZE USR s
5020 PRINT "machinetaalprogramma
 uitgevoerd.": PAUSE 100
5030 GO TO 100
6000 REM ************************
6001 REM * op cassette bewaren *
6003 REM ************************
6010>CLS :PRINT "Verwijder de EA
R-plug en steek  de MIC-plug in
de recorder.     Zet de recorder
 op opname."
6020 SAVE "machcode"CODE s,mc
6030 CLS : PRINT "Spoel de casse
tte terug en zet  de EAR-plug we
er in de recorder,om te controle
ren of het pro-   gramma goed is
 opgenomen."
6040 VERIFY "machcode"CODE : GO
TO 100
7000 REM ************************
7001 REM * van cassette lezen  *
7002 REM ************************
7010 CLS : PRINT "start de casse
tterecorder."
7020 LOAD "machcode"CODE
7030 GO TO 100
8000 REM ************************
8001 REM * backup van machtaal *
8003 REM ************************
8010 CLS : PRINT "verwijder de E
AR-plug en steek  de MIC-plug in
 de recorder.     Zet de recorde
r op opnemen."
8020 SAVE "machtaal" LINE 10
8030 CLS : PRINT "Spoel de casse
tte terug en zet  de EAR-plug we
er in de recorder,om te controle
ren of het pro-   gramma goed is
 opgenomen."
8040 VERIFY "machtaal": GO TO 10
0
```

4. Afdrukken van geselecteerde BASIC-regels

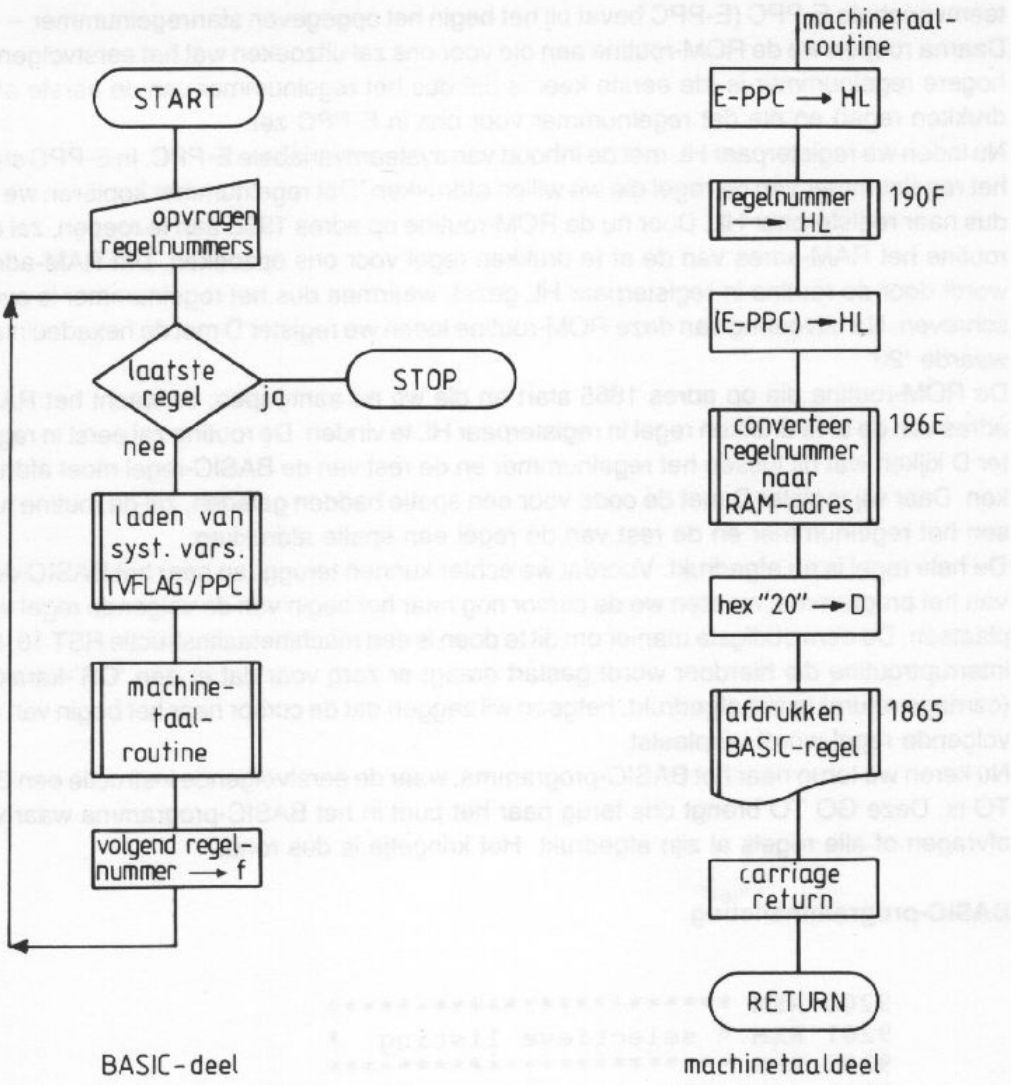
Heeft u zich ooit wel eens gewenst in staat te zijn slechts een beperkt aantal regels van uw BASIC-programma te 'listen'? Hoewel deze mogelijkheid bij het afdrukken van een stukje listing op het beeldscherm al handig kan zijn, is het voor het afdrukken op een printer van het allergrootste belang. Een printer is immers relatief langzaam en iedere regel kost papier en inktlint. Bovendien is het ook nog zo dat je een listing op het beeldscherm steeds na 22 regels kunt stoppen, terwijl een listing op een printer in zijn geheel wordt afgedrukt.
Het zou dus plezierig zijn als je de Spectrum kon opdragen om alleen de regels vanaf een bepaald startregelnummer tot en met een opgegeven laatste regelnummer af te drukken. Hoewel dit op het eerste gezicht een beetje ingewikkeld kan lijken, zult u zien dat het met een zeer kort programma te doen is.
Wat we eigenlijk willen, is dat we een regelnummer kunnen opgeven. Het programma moet dan in het BASIC-programmageheugen opzoeken op welk adres die regel in dat geheugen begint. Daarna drukken we die regel af; laten we er even van uitgaan dat we de regel op het beeldscherm afdrukken. Is de regel afgedrukt dan moeten we de carriage (cursor) naar het begin van de volgende regel verplaatsen. Daarna controleren we of alle gewenste regels zijn afgedrukt, of dat er nog meer regels moeten worden afgedrukt. Indien er nog meer regels moeten worden afgedrukt, dan herhalen we voorgaande procedure. Zijn alle gewenste regels afgedrukt dan beëindigen we het programma.
In de BASIC-ROM komen enkele routines voor die we goed kunnen gebruiken bij de verwezenlijking van ons programma. Het gebruik van die ROM-routines zal niet alleen de programmalengte sterk inkorten, doch ook de uitvoeringssnelheid zal er sterk door worden verhoogd.

4.1 ROM-routines voor het afdrukken van een regel

Op adres 190F (hexadecimaal) start een routine die de waarde, die in systeemvariabele E-PPC staat, leest. Door E-PPC voor het aanroepen van die routine te laden met een regelnummer, zal de ROM-routine in het BASIC-geheugen zoeken naar het eerstvolgende hogere regelnummer dat in het programma voorkomt. Dit regelnummer wordt door de routine in E-PPC gezet.
Op adres 196E (hexadecimaal) start een routine die uitzoekt wat het RAM-adres van de regel, waarvan het regelnummer in het Z80-registerpaar HL staat, is.
Een heel belangrijke routine start op adres 1865 (hexadecimaal). Deze routine drukt de BASIC-regel, waarvan het startadres in registerpaar HL moet staan, af op een output-device (beeldscherm of printer). In Z80-register D verwacht deze routine de 'current cursor'-indicator aan te treffen. Daar wij voor onze toepassing deze indicator niet willen hebben, zetten wij de code van een spatie (hexadecimaal 20) in dat register.

4.2 Het programma

Met onze tot nu toe opgedane kennis zullen we eerst eens bekijken hoe we de hiervoor genoemde ROM-routines voor ons doel kunnen inzetten.



Afb. 4-1 Flowcharts van BASIC- en machinetaalprogramma

Zoals u ziet bestaat de flowchart uit twee delen, een BASIC-deel en een machinetaaldeel. In het BASIC-deel vragen we de begin- en eindregelnummers voor de regels die moeten worden afgedrukt. Door van het beginregelnummer 1 af te trekken, bereiken we dat het eerste regelnummer dat door de ROM-routine op adres 190F in E-PPC zal worden geladen, het regelnummer van de eerste regel die moet worden afgedrukt is.
We zetten dus het opgegeven beginregelnummer – 1 in E-PPC. Hierna moeten we eerst controleren of we alle gewenste regels al hebben afgedrukt. Zou dit zo zijn, dan beëindigen we het programma. Indien er nog regels moeten worden afgedrukt, dan gaan we gewoon door met het programma.
Voordat we naar de machinetaalroutine springen, zetten we het regelnummer in systeemvariabele E-PPC en zetten we de systeemvariabele TVFLAG, om te bewerkstelligen dat de listing naar de bovenste 22 regels van het scherm zullen worden gestuurd. Is dit gebeurd, dan starten we de machinetaalroutine.
In de machinetaalroutine laden we allereerst het registerpaar HL met het adres van sys-

teemvariabele E-PPC (E-PPC bevat bij het begin het opgegeven startregelnummer – 1). Daarna roepen we de ROM-routine aan die voor ons zal uitzoeken wat het eerstvolgende hogere regelnummer is (de eerste keer is dat dus het regelnummer van de eerste af te drukken regel) en die dat regelnummer voor ons in E-PPC zet.
Nu laden we registerpaar HL met de inhoud van systeemvariabele E-PPC. In E-PPC staat het regelnummer van de regel die we willen afdrukken. Dat regelnummer kopiëren we nu dus naar registerpaar HL. Door nu de ROM-routine op adres 196E aan te roepen, zal die routine het RAM-adres van de af te drukken regel voor ons opzoeken. Dat RAM-adres wordt door de routine in registerpaar HL gezet, waarmee dus het regelnummer is overschreven. Na uitvoering van deze ROM-routine laden we register D met de hexadecimale waarde '20'.
De ROM-routine die op adres 1865 start en die we nu aanroepen, verwacht het RAM-adres van de af te drukken regel in registerpaar HL te vinden. De routine zal eerst in register D kijken wat hij tussen het regelnummer en de rest van de BASIC-regel moet afdrukken. Daar wij register D met de code voor een spatie hadden geladen, zal de routine tussen het regelnummer en de rest van de regel een spatie afdrukken.
De hele regel is nu afgedrukt. Voordat we echter kunnen teruggaan naar het BASIC-deel van het programma, moeten we de cursor nog naar het begin van de volgende regel verplaatsen. De eenvoudigste manier om dit te doen is een machinetaalinstructie RST 10. De interruptroutine die hierdoor wordt gestart draagt er zorg voor dat er een 'CR'-karakter (carriage return) wordt afgedrukt, hetgeen wil zeggen dat de cursor naar het begin van de volgende regel wordt verplaatst.
Nu keren we terug naar het BASIC-programma, waar de eerstvolgende instructie een GO TO is. Deze GO TO brengt ons terug naar het punt in het BASIC-programma waar we afvragen of alle regels al zijn afgedrukt. Het kringetje is dus rond.

BASIC-programmalisting

```
9200 REM ************************
9201 REM * selectieve listing  *
9202 REM ************************
9203 INPUT "Eerste regelnr.?";f
9204 INPUT "laatste regelnr.?";l
9205 LET f=f-1
9206 IF f>=l THEN  STOP
9207 GO SUB 9211
9208 RANDOMIZE USR 60019
9209 LET f=256*PEEK 23626+PEEK 2
3625
9210 GO TO 9206
9211 REM ************************
9212 REM *laden syst.variabelen*
9213 REM *   E-PPC en TVFLAG   *
9214 REM ************************
9215 LET f1=INT (f/256)
9216 LET f2=f-256*f1
9217 POKE 23626,f1
9218 POKE 23625,f2
9219 POKE 23612,0
9220 RETURN
```

```
9221 CLS : PRINT  FLASH 1;"LAAT
CASSETTE DRAAIEN"
9222 LOAD "scrncode"CODE 60019,5
0
9223 CLS : STOP
```

Machinetaallisting

adres	mnem	operands	hex.instr.	commentaar
60019	LD	HL,E-PPC	21 49 5C	adr. E-PPC --> HL
60022	CALL	190F	CD 0F 19	volgend regelnr.--> E-PPC
60025	LD	HL,(E-PPC)	2A 49 5C	inh. E-PPC --> HL
60028	CALL	196E	CD 6E 19	RAM-startadres van regel
60031	LD	D,20	16 20	spatie naar reg. D
60033	CALL	1865	CD 65 18	druk regel af
60036	RST	10	D7	carriage return
60037	RET		C9	terug naar BASIC

4.3 De printer-versie

Zoals eerder in dit hoofdstuk al werd opgemerkt, zou dit programma vooral bij gebruik van een printer grote voordelen hebben. Tot nu toe hebben we echter alleen gekeken hoe we een selectieve listing op het beeldscherm kunnen maken. Nu zullen we zien hoe eenvoudig het is om het programma aan te passen voor het afdrukken op een printer.
We behoeven niets meer te doen dan het 'printer-channel' te activeren. De ROM-routine die dat voor ons kan doen bevindt zich op adres 1601 (hexadecimaal). Deze routine verwacht de channel indicator in de accumulator (register A) aan te treffen. Als we dus register A laden met de waarde '03' (de printer channel indicator), en we roepen daarna de ROM-routine op adres 1601 aan, dan zal na het uitvoeren van die ROM-routine de printer geselecteerd zijn.
Om dus de listing op de printer te laten verschijnen behoeven we alleen de machinetaalroutine wat uit te breiden. Deze uitgebreide machinetaalroutine volgt hierna.

Printer-versie van het machinetaalprogramma

adres	mnem	operands	hex.instr.	commentaar
60019	LD	HL,E-PPC	21 49 5C	adr. E-PPC --> HL
60022	CALL	190F	CD 0F 19	volgend regelnr.--> E-PPC
60025	LD	A,03	3E 03	set printer I/O
60027	CALL	1601	CD 01 16	open printer channel
60030	LD	HL,(E-PPC)	2A 49 5C	inh. E-PPC --> HL
60033	CALL	196E	CD 6E 19	RAM-startadres van regel
60036	LD	D,20	16 20	spatie naar reg. D
60038	CALL	1865	CD 65 18	druk regel af
60041	RST	10	D7	carriage return
60042	RET		C9	terug naar BASIC

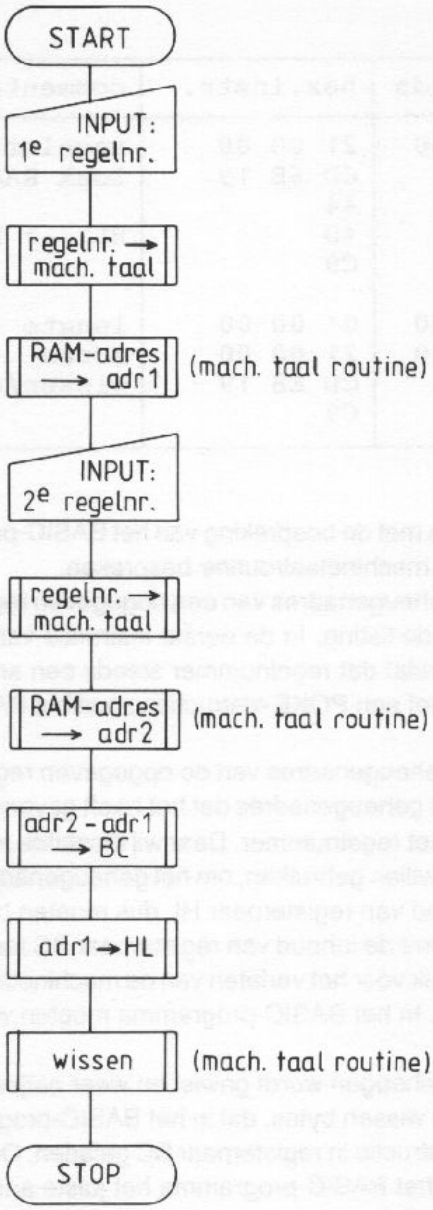
5. Het verwijderen van een blok BASIC-regels

Het is mij al vaak overkomen dat ik een groot deel van een eerder geschreven BASIC-programma opnieuw had kunnen gebruiken voor een nieuw te schrijven programma. Meestal was het dan te veel werk om alle regels die ik niet kon gebruiken stuk voor stuk uit de oude listing te verwijderen. Dit had dan vaak tot gevolg dat ik toch maar besloot de regels die ik wel kon gebruiken opnieuw in te typen.
Voorgaand probleem heeft mij dan ook doen zoeken naar een eenvoudige wijze om een heel blok van BASIC-regels in een keer uit de listing van een bestaand programma te verwijderen. Het grootste probleem was daarbij dat niet alleen de ruimte die open viel moest worden opgevuld, maar dat ook een mij onbekend aantal andere zaken, zoals systeemvariabelen en stacks, met het korter geworden programma in overeenstemming moest worden gebracht.
Daar het in de EDIT-mode mogelijk is om een BASIC-regel uit een programma te verwijderen, moet er dus wel een ROM-routine in de BASIC-ROM aanwezig zijn die ervoor zorgt dat het hele proces van het verwijderen correct verloopt. Immers, wij hoeven alleen het te verwijderen regelnummer op te geven en de ROM-routine doet de rest. Het vinden van de juiste ROM-routine was dus mijn eerste karwei.
Die ROM-routine had ik al vrij spoedig gevonden. Doch toen bleek dat die ROM-routine niet tevreden was met een regelnummer. De gevonden routine verwacht namelijk in Z80-registerpaar HL het RAM-adres van waar af het geheugen dient te worden gewist en in registerpaar BC het aantal bytes dat dient te worden gewist. Met deze twee gegevens wordt dan het geheugen gewist en worden alle BASIC-regels die na dat gewiste deel in het geheugen staan, teruggeschoven, zodat ze weer aansluiten met de BASIC-regels die voor het gewiste deel staan. Ook worden door diezelfde ROM-routine alle van belang zijnde pointers aangepast aan de nieuw ontstane situatie.
Het opgeven van een regelnummer was dus duidelijk niet genoeg. Er moest een manier zijn om het RAM-adres van een bepaald regelnummer uit te vinden. Dit is natuurlijk mogelijk met behulp van PEEK, doch daarmee kan het zoeken van een RAM-adres wel eens erg langdurig worden. In het programma 'sellist' had ik gelukkig al dezelfde ROM-routine nodig gehad als nu; een routine die voor een in registerpaar HL opgegeven regelnummer het startadres in RAM uitzoekt. Door deze routine het regelnummer van de eerste BASIC-regel die ik kwijt wil op te geven krijg ik het startadres van die regel terug. Als ik dan diezelfde ROM-routine nog eens start, maar er nu het regelnummer van de eerste regel die ik NIET kwijt wil aan opgeef, dan krijg ik het RAM-adres tot waaraan toe het geheugen moet worden gewist.
Door nu het eerste te wissen RAM-adres af te trekken van het eerste niet te wissen RAM-adres, wordt de wis-lengte verkregen. Deze wis-lengte kunnen we dan in registerpaar BC zetten, terwijl we het startadres in registerpaar HL zetten. Indien we nu de wis-routine uit de BASIC-ROM starten, zal het hele gebied, vanaf de eerste tot de laatste door ons opgegeven BASIC-regel keurig uit het BASIC-programma worden verwijderd. En doordat we bijna alles in machinetaal doen wordt dat nog ontzettend snel gedaan ook.

Machinetaallisting

adres	mnem	operands	hex.instr.	commentaar
60000	LD	HL,0000	21 00 00	regelnr. --> HL
60003	CALL	196E	CD 6E 19	Zoek RAM-adres
60006	LD	B,H	44	
60007	LD	C,L	4D	HL --> BC
60008	RET		C9	
60009	LD	BC,0000	01 00 00	lengte --> BC
60012	LD	HL,0000	21 00 00	adres --> HL
60015	CALL	19E8	CD E8 19	wissen/opvullen RAM
60018	RET		C9	

Voordat we nu verder gaan met de bespreking van het BASIC-programma wil ik nog even een aantal details van de machinetaalroutine bespreken.
De routine waarmee het geheugenadres van een opgegeven regel kan worden gevonden begint op adres 60000 in de listing. In de eerste instructie van die routine heb ik geen regelnummer ingevuld, omdat dat regelnummer steeds een andere kan zijn. We zullen dat regelnummer er dus met een POKE-instructie vanuit het BASIC-programma in moeten zetten.
De ROM-routine die het geheugenadres van de opgegeven regel uitzoekt start op adres 196E. Deze routine zet het geheugenadres dat het heeft gevonden in registerpaar HL en overschrijft daarmee dus het regelnummer. Daar wij diezelfde routine direct na de eerste keer nog een tweede keer willen gebruiken, om het geheugenadres van de laatste regel te vinden, zullen we de inhoud van registerpaar HL dus moeten bewaren. Bij terugkeer uit een machinetaalroutine komt de inhoud van registerpaar BC ter beschikking van BASIC. Dit is dan ook de reden dat ik vóór het verlaten van de machinetaalroutine registerpaar HL naar registerpaar BC laad. In het BASIC-programma moeten we dan de inhoud van BC opvangen en bewaren.
De routine waarmee het geheugen wordt gewist en weer netjes aangeschoven, start op adres 60009. Het aantal te wissen bytes, dat in het BASIC-programma moet zijn vastgesteld, wordt in de eerste instructie in registerpaar BC geladen. Ook hier is weer de waarde 0 ingevuld, dus zal vanuit het BASIC-programma het juiste aantal bytes in de instructie moeten worden ge-POKEd. Hetzelfde geldt voor de volgende instructie, hiermee wordt het adres van waar af moet worden gewist in registerpaar HL geladen. Zijn deze gegevens eenmaal in de genoemde registers aanwezig, dan kan de ROM-routine die op adres 19E8 begint worden aangeroepen. Na uitvoering van deze ROM-routine zijn alle gewenste regels gewist.
We weten nu welke ROM-routines we tot onze beschikking hebben en hoe we die routines kunnen aanroepen. We weten zelfs al dat er vanuit het BASIC-programma een aantal instructies uit het machinetaalprogramma moet worden ingevuld. Laten we dan nu eens naar dat BASIC-programma kijken.



Afb. 5-1 Flowchart van het BASIC-programma 'blokdel'

De flowchart uit afbeelding 5-1 geeft weer, hoe het programma in grote lijnen in elkaar zit. Na het opvragen van de eerste regel wordt de subroutine op regel 9500 aangeroepen. Deze subroutine converteert het opgegeven regelnummer, dat in variabele 'r' staat, naar twee getallen, een meest significant deel dat in variabele r1 wordt gezet en een minst significant deel dat in variabele r2 wordt gezet. Vervolgens worden deze nieuw verkregen getallen in de eerste instructie van het eerste deel van de machinetaalroutine ge-POKEd. Hierna wordt op regel 9050 de machinetaalroutine gestart. Na uitvoering daarvan zal de inhoud van registerpaar BC worden toegekend aan de variabele adr1. In adr1 staat nu dus de eerste te wissen RAM-locatie.

Nu wordt het regelnummer van de laatste te wissen regel opgevraagd. Om nu die laatste regel ook nog gewist te krijgen wordt in regel 9070 het opgegeven regelnummer met 1 verhoogd. Hierdoor zal straks de ROM-routine het eerstvolgende regelnummer na het opgegeven regelnummer opzoeken. De subroutine op regel 9500 wordt ook nu weer gebruikt. Na terugkomst uit de subroutine wordt dezelfde machinetaalroutine als bij het eerste regelnummer aangeroepen. De inhoud van registerpaar BC komt nu echter in variabele adr2 terecht. In adr2 staat nu dus het eerste adres dat niet meer moet worden gewist.
In regel 9100 wordt het aantal te wissen bytes bepaald. Dat aantal wordt met de subroutine die op regel 9600 start geconverteerd en in de eerste instructie van het tweede deel van de machinetaalroutine ge-POKEd. Hiermee is dus de wis-lengte aangegeven. Een overeenkomstige actie wordt gestart voor het opgeven van het adres waar vanaf moet worden gewist. Dit wordt met de subroutine op regel 9700 in de tweede instructie van het machinetaalprogramma ingevuld.
Na al deze voorbereidingen wordt met regel 9140 de tweede machinetaalroutine gestart. Nadat deze routine ervoor heeft gezorgd dat alle gewenste regels uit het programma zijn verwijderd, komen we terug in het BASIC-programma. Daar wordt dan nog het beeldscherm schoongemaakt en wordt u op de hoogte gebracht van het feit dat uw wens is vervuld. Hierna stopt de uitvoering van het programma op regel 9150.
Zoals u zich wellicht realiseert kunt u na een STOP-instructie door op de ENTER-toets te drukken een listing van het in het geheugen staande BASIC-programma krijgen. U kunt dus nu direct controleren of het programma het werk goed heeft uitgevoerd. Waarschijnlijk zult u dat in het begin inderdaad steeds controleren, omdat er tussen het moment waarop u het laatste regelnummer in geeft en het moment waarop het bericht verschijnt, dat de regels zijn verwijderd, slechts een fractie van een seconde ligt.

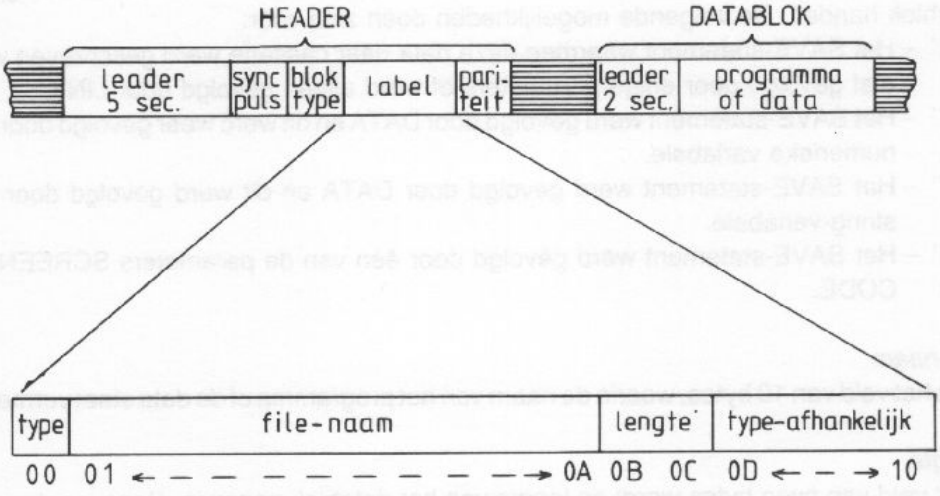
Programmalisting 'blockdel'

```
9000 REM ************************
9005 REM *    BLOCK DELETE      *
9010 REM ************************
9015 REM prognaam=blokdel
9020 REM copyright 1983
9025 REM
9030 INPUT "eerste regel? ";r
9040 GO SUB 9500
9050 LET adr1=USR 60000
9060 INPUT "laatste regel? ";r
9070 LET r=r+1
9080 GO SUB 9500
9090 LET adr2=USR 60000
9100 LET bc=adr2-adr1
9110 GO SUB 9600
9120 LET hl=adr1
9130 GO SUB 9700
9140 RANDOMIZE USR 60009
9150 PRINT "De opgegeven regels
zijn uit de listing verwijderd."
'''"Druk op ENTER voor een listin
g.": STOP
9480 REM ************************
9485 REM *  regelnr --> adres   *
```

```
9490 REM ***********************
9500 LET r1=INT (r/256)
9510 LET r2=r-256*r1
9520 POKE 60001,r2
9530 POKE 60002,r1
9540 RETURN
9580 REM ***********************
9585 REM *  wislengte --> BC   *
9590 REM ***********************
9600 LET bc1=INT (bc/256)
9610 LET bc2=bc-256*bc1
9620 POKE 60010,bc2
9630 POKE 60011,bc1
9640 RETURN
9680 REM ***********************
9685 REM *  startadres --> HL  *
9690 REM ***********************
9700 LET hl1=INT (hl/256)
9710 LET hl2=hl-256*hl1
9720 POKE 60013,hl2
9730 POKE 60014,hl1
9740 RETURN
9890 REM ***********************
9900 PRINT  FLASH 1;"LAAT DE CAS
SETTE DRAAIEN.     DE MACHINET
AALROUTINE MOET NOG  WORDEN GELA
DEN."
9910 LOAD "delcode"CODE
9920 CLS : PRINT "STOP DE CASSET
TE"
9930 FOR a=1 TO 5
9940 BEEP 1,a
9950 NEXT a
9960 STOP
```

6. Cassettelabelinformatie lezen

U hebt nu reeds enkele machinetaalprogramma's ingegeven en op cassette geschreven. Mocht u nu geen goede administratie hebben bijgehouden van al die routines, hun startadressen en hun lengte, dan loopt u in de toekomst gevaar meerdere machinetaalroutines over elkaar heen te laden of een bepaalde routine vanuit uw BASIC-programma aan te roepen op een verkeerd startadres.
Om van ieder programma of gegevensbestand dat u op cassette hebt geladen precies te weten wat voor soort programma het is en waar het in het geheugen van de Spectrum zal terechtkomen, dient u in staat te zijn de gegevens uit het cassettelabel te lezen en deze gegevens op een overzichtelijke manier naar het beeldscherm of naar de printer te sturen. De programma's in dit hoofdstuk stellen u daartoe in staat.
Hoewel de gegeven programma's hun werk voortreffelijk doen, kan ik mij voorstellen dat u uw eigen specifieke wensen aan de output van de programma's hebt. Om het u wat gemakkelijker te maken de programma's naar uw eigen inzichten aan te passen, heb ik dan ook besloten de programmabeschrijvingen te laten voorafgaan door een stukje algemene informatie over de indeling van cassettelabels voor de ZX Spectrum.
Of u nu een BASIC-programma, een machinetaalprogramma, het beeldschermgeheugen of gewoon gegevens naar cassette schrijft, het gevolg zal altijd zijn dat datgene wat u naar cassette hebt geschreven zal worden voorafgegaan door een 'label'. U hebt dat al wel gemerkt bij het wegschrijven van een programma. Eerst wordt er een langdurige toon weggeschreven, dan hoort u even niets en even later volgt er een kortere toon gevolgd door een hoop lawaai, het programma.
In afbeelding 6-1 is een stukje van de cassetteband weergegeven met daarop getekend wat er op die cassette staat. U ziet dat er duidelijk sprake is van twee delen: het eerste deel, de header, waarvan het belangrijkste deel het label is en het tweede deel, waarvan het belangrijkste deel het programma of de gegevens is. Het label uit het eerste deel is daarbij in wat meer detail uitgewerkt.



Afb. 6-1 Indeling van de cassetteband

De verschillende velden die in de afbeelding worden genoemd zullen nu aan een nadere beschouwing worden onderworpen:

Leader 5 sec
Dit veld stelt de fluittoon van ongeveer 5 seconden voor, die het begin van een nieuw programma of gegevensblok voorafgaat. De functie van deze fluittoon is onder meer om cassetterecorders die een automatische volumeregeling hebben in staat te stellen hun volumeregeling op de juiste sterkte af te stellen. Dit alles natuurlijk tijdens het opnemen van een programma. Bij het terugladen van programma's kunt u deze toon gebruiken om de afspeel-volumeregelaar op uw cassetterecorder op de juiste sterkte af te stellen. Een goede afregeling is bereikt wanneer u donkere en lichte balken van gelijke grootte hebt verkregen.

Sync puls
Op deze puls kan de Spectrum zich, bij het laden van een programma, op het begin van een byte synchroniseren. Dit veld komt voor in zowel de header als in het daaropvolgende 'datablok'. Immers, in beide gevallen moet er iets van cassette worden gelezen.

Bloktype
Dit is een veld van slechts 1 byte. De inhoud van dit byte kan alleen hexadecimaal '00' of 'FF' zijn (decimaal 0 of 255). Indien de waarde van dit byte 0 is, dan geeft dit aan dat het momenteel gelezen blok een header is. Is de waarde 255, dan wordt het datablok gelezen.

Label
Het label wordt tijdens het wegschrijven van de data in het werkgeheugen van de Spectrum samengesteld aan de hand van het ingegeven SAVE-commando. Het label zelf bestaat weer uit een aantal velden, zoals in de afbeelding is weergegeven. Een verdere uitwerking van deze label-velden is als volgt:

Type
Dit veld van 1 byte geeft aan om wat voor soort 'data' het zich in het daarna volgende datablok handelt. De volgende mogelijkheden doen zich voor:
"00" – Het SAVE-statement waarmee deze data naar cassette werd geschreven werd niet gevolgd door enige parameters of werd alleen gevolgd door LINE.
"01" – Het SAVE-statement werd gevolgd door DATA en dit werd weer gevolgd door een numerieke variabele.
"02" – Het SAVE-statement werd gevolgd door DATA en dit werd gevolgd door een string-variabele.
"03" – Het SAVE-statement werd gevolgd door één van de parameters SCREEN$ of CODE.

File-naam
Dit is het veld van 10 bytes, waarin de naam van het programma of de data staat vermeld.

Lengte
In dit veld van twee bytes wordt de lengte van het datablok gegeven. Deze lengte moet afhankelijk van de soort data op de volgende manier worden gelezen:

CODE – De in het SAVE-statement achter CODE opgegeven lengte.
SCREEN$ – De totale lengte van het beeldscherm- en attributengeheugen samen (hexadecimaal '1B00' is decimaal 6912).
DATA – De lengte van de in het SAVE-statement opgegeven variabele.

Type-afhankelijk
De betekenis van deze vier bytes hangt geheel af van de soort data die met het SAVE-statement naar cassette werd geschreven. Per soort data volgt hier de betekenis:
CODE – Op de posities '0D' en '0E' staat het startadres van de CODE zoals die met het SAVE-statement werd opgegeven.
DATA – Op positie '0E' vindt u de variabelenaam van de met het SAVE-statement weggeschreven variabele.
LINE – Indien u een programma had weggeschreven met een LINE-functie, dan staat op de posities '0D' en '0E' het regelnummer waarop het programma automatisch zal worden gestart.
SCREEN$ – Het startadres van het beeldschermgeheugen zal op posities '0D' en '0E' staan.
Indien er bij het SAVE-statement geen verdere parameters werden gespecificeerd, dan zult u op positie '0E' de hexadecimale waarde '80' (= decimaal 128) aantreffen.

Pariteit
Dit is het pariteits-byte dat wordt gegenereerd tijdens het naar cassette schrijven van de data. Tijdens een LOAD, MERGE en VERIFY wordt ook een pariteits-byte gegenereerd en wel van de ingelezen data. Het nu gegenereerde pariteits-byte wordt vergeleken met het van cassette ingelezen pariteits-byte en indien er een ongelijkheid wordt geconstateerd, zal een bericht worden gegeven dat er een leesfout is geconstateerd.

Leader 2 sec
Dit is de fluittoon die het datablok voorafgaat. De functie is dezelfde als voor 'Leader 5 sec'.

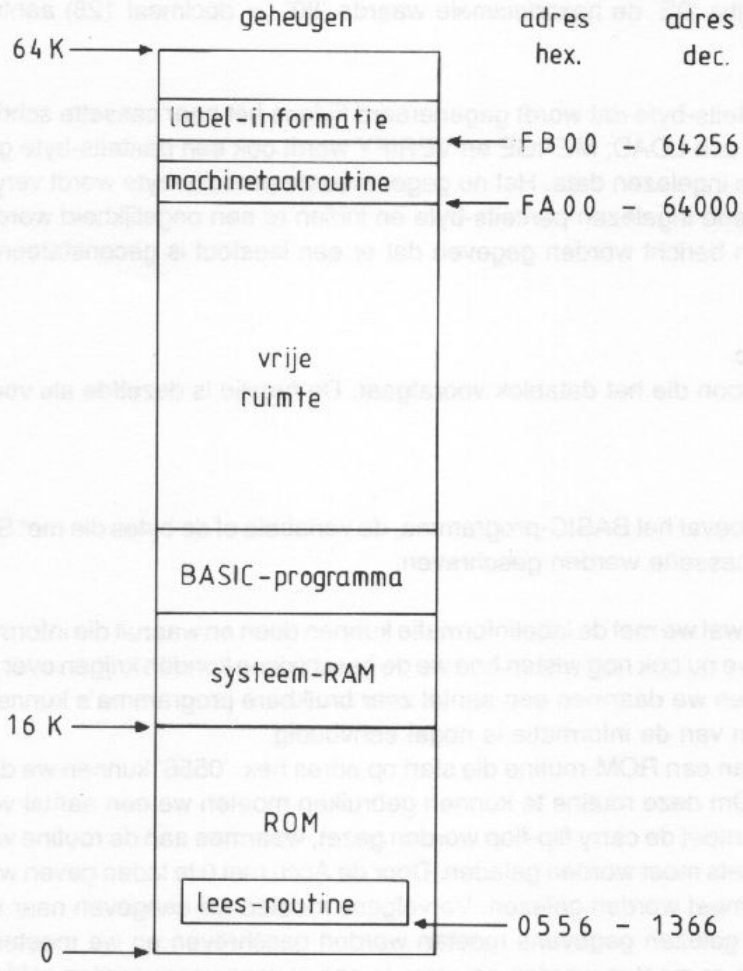
Datablok
Het datablok bevat het BASIC-programma, de variabele of de bytes die met SCREEN$ of CODE naar cassette werden geschreven.

We weten nu wat we met de labelinformatie kunnen doen en waaruit die informatie precies bestaat. Als we nu ook nog wisten hoe we de beschikking konden krijgen over die informatie, dan zouden we daarmee een aantal zeer bruikbare programma's kunnen schrijven. Dat verkrijgen van de informatie is nogal eenvoudig.
Met behulp van een ROM-routine die start op adres hex. '0556' kunnen we data van cassette lezen. Om deze routine te kunnen gebruiken moeten we een aantal voorwaarden vervullen. Zo moet de carry flip-flop worden gezet, waarmee aan de routine wordt aangegeven dat er iets moet worden geladen. Door de Accu met 0 te laden geven we aan dat er header-data moet worden gelezen. Vervolgens moeten we aangeven naar welk geheugenadres de gelezen gegevens moeten worden geschreven en we moeten aangeven hoeveel bytes er moeten worden gelezen. Is aan al deze voorwaarden voldaan dan kan de ROM-routine worden gestart.

Machinetaalroutine "casread"

adres	mnem	operands	hex.instr.	commentaar
64000	SCF		37	set: LOAD
64001	LD	A,00	3E 00	set: lees header
64003	LD	IX,FB00	DD 21 00 FB	adres voor headerdata
64007	LD	DE,11	11 11 00	lengte van headerdata
64010	CALL	0556	CD 56 05	ROM-routine die de cassette leest.
64013	RET		C9	terug naar BASIC

In de machinetaal-listing ziet u dat de plaats waarnaartoe de gelezen data moet worden geschreven in het indexregister IX is gezet. Het resultaat van de ROM-routine zal dus zijn dat de gelezen bytes vanaf adres hex. 'FB00' in het geheugen komen te staan. Hebt u een 16 K Spectrum dan zult u een lager adres moeten kiezen, bijvoorbeeld '7B00'. Verder zal het u duidelijk zijn geworden uit de machinetaal-listing dat het aantal te lezen



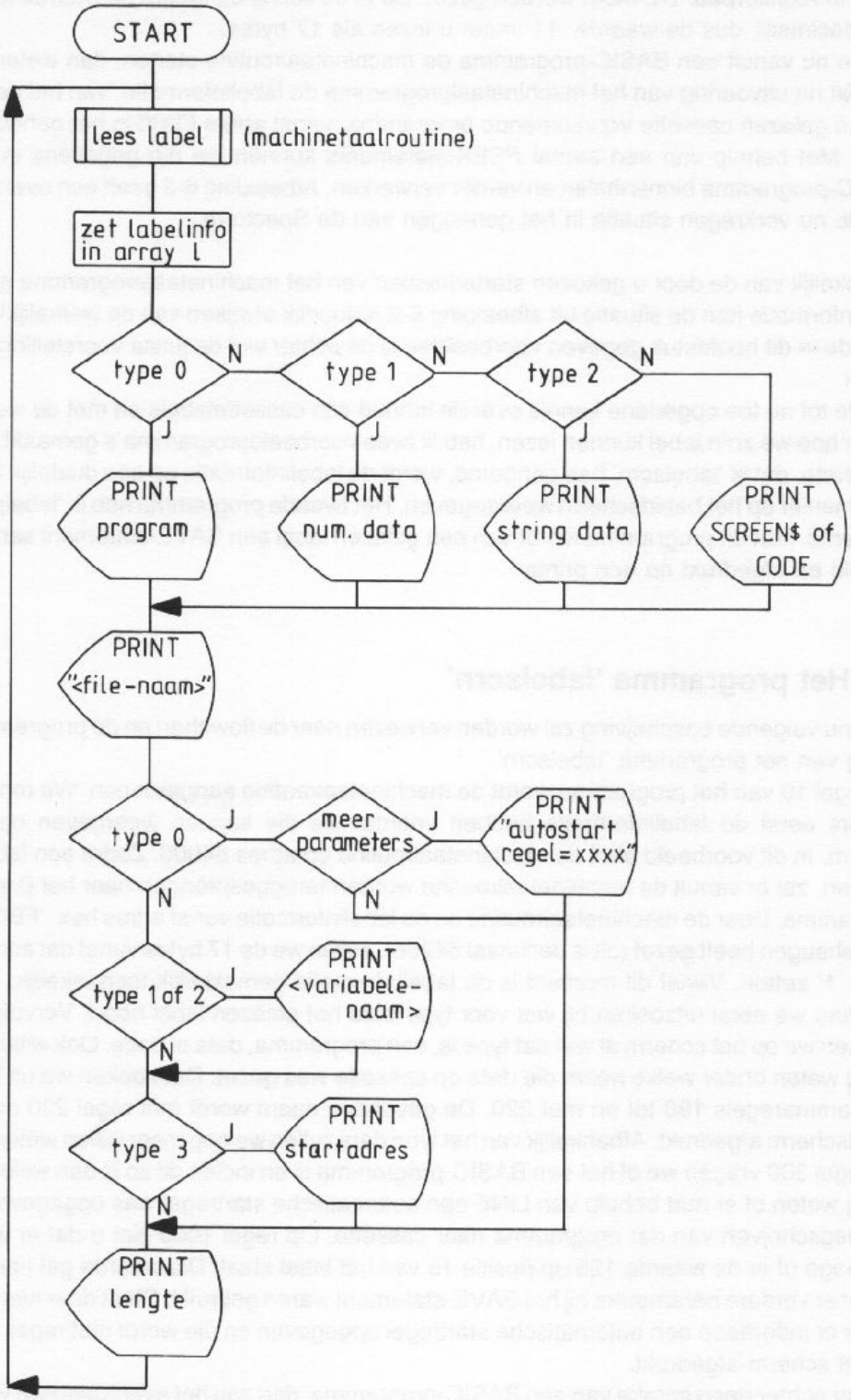
Afb. 6-2 Geheugenindeling na laden labelinformatie

bytes in registerpaar DE moet worden gezet. De in de listing aangegeven waarde is wel hexadecimaal, dus de waarde '11' moet u lezen als 17 bytes.
Als we nu vanuit een BASIC-programma de machinetaalroutine starten, dan weten we dus dat na uitvoering van het machinetaalprogramma de labelinformatie, van het eerste op een gelezen cassette voorkomende programma, vanaf adres FB00 in het geheugen staat. Met behulp van een aantal PEEK-statements kunnen we die gegevens in ons BASIC-programma binnenhalen en verder verwerken. Afbeelding 6-2 geeft een overzicht van de nu verkregen situatie in het geheugen van de Spectrum.

Afhankelijk van de door u gekozen startadressen van het machinetaalprogramma en de labelinformatie kan de situatie uit afbeelding 6-2 natuurlijk afwijken van de werkelijkheid. Voor de in dit hoofdstuk gegeven voorbeelden is dit echter wel de juiste voorstelling van zaken.
Met de tot nu toe opgedane kennis over de inhoud van cassettelabels en met de wetenschap hoe we zo'n label kunnen lezen, heb ik twee voorbeeldprogramma's gemaakt. Met het eerste, dat ik 'labelscrn' heb genoemd, wordt de labelinformatie op een duidelijk leesbare manier op het beeldscherm weergegeven. Het tweede programma heb ik 'labelprint' genoemd. Met dit programma wordt van een gelezen label een SAVE-statement samengesteld en afgedrukt op een printer.

6.1 Het programma 'labelscrn'

In de nu volgende beschrijving zal worden verwezen naar de flowchart en de programmalisting van het programma 'labelscrn'.
Op regel 10 van het programma wordt de machinetaalroutine aangeroepen. We moeten immers eerst de labelinformatie hebben voordat we die kunnen weergeven op het scherm. In dit voorbeeld start de machinetaalroutine op adres 64000. Zodra een label is gelezen, zal er vanuit de machinetaalroutine worden teruggesprongen naar het BASIC-programma. Daar de machinetaalroutine nu de labelinformatie vanaf adres hex. 'FB00' in het geheugen heeft gezet (dit is decimaal 64256), zullen we de 17 bytes vanaf dat adres in array '1' zetten. Vanaf dit moment is de labelinformatie gemakkelijk toegankelijk.
Nu gaan we eerst uitzoeken bij wat voor type data het gelezen label hoort. Vervolgens drukken we op het scherm af wat dat type is, een programma, data of code. Ook willen we graag weten onder welke naam die data op cassette was gezet. Dat zoeken we uit in de programmaregels 190 tot en met 220. De gevonden naam wordt met regel 230 op het beeldscherm afgedrukt. Afhankelijk van het type data zullen we nog meer willen weten.
Op regel 300 vragen we of het een BASIC-programma is en indien dit zo is dan willen we graag weten of er met behulp van LINE een automatische startregel was opgegeven bij het wegschrijven van dat programma naar cassette. Op regel 1000 ziet u dat er wordt gevraagd of er de waarde 128 op positie 15 van het label staat. Die waarde gaf immers aan of er verdere parameters bij het SAVE-statement waren gebruikt. Staat daar niet 128, dan is er inderdaad een automatische startregel opgegeven en die wordt met regel 1010 op het scherm afgedrukt.
Was er echter geen sprake van een BASIC-programma, dan zou het eventueel een variabele hebben kunnen zijn. Dit wordt op regel 400 afgevraagd. Is het inderdaad een variabele, dan wordt de naam van die variabele met de regels 800 tot en met 850 afgedrukt. Daar afhankelijk van het type variabele de mogelijkheid bestaat dat de meest significante bits



Afb. 6-3 Flowchart van het BASIC-programma 'labelscrn'

van de variabelenaam gezet zijn, moeten deze bits eruit gezeefd worden. Dat is de reden dat er zoveel regels zijn gebruikt voor het afdrukken van de variabelenaam.
Indien het label noch bij een BASIC-programma, noch bij een variabele hoort, dan moet het CODE zijn. Op regel 500 wordt dit afgevraagd. Blijkt het inderdaad CODE te zijn, dan wordt het startadres van die code afgedrukt. Vervolgens wordt met regel 600 de lengte afgedrukt. Deze regel wordt zowel voor CODE als voor de andere soorten data gebruikt. U weet dus altijd hoe lang het op cassette staande programma is.
Na het afdrukken van alle gegevens wordt u met regel 9080 gevraagd of u door wilt gaan. Zo ja, dan wordt weer naar het begin van het programma gesprongen.
Misschien is u inmiddels opgevallen dat er nog een aantal regels niet zijn besproken, de regels 9000 tot en met 9070. Deze regels dienen voor het automatisch laten laden van het machinetaalprogramma nadat u het BASIC-programma hebt geladen. Het BASIC-programma moet u dan wel zo wegschrijven dat het automatisch wordt gestart op regel 9000. Daar over dit onderwerp in hoofdstuk 2 al diepgaand is gesproken, zullen we er nu niet meer verder op ingaan.

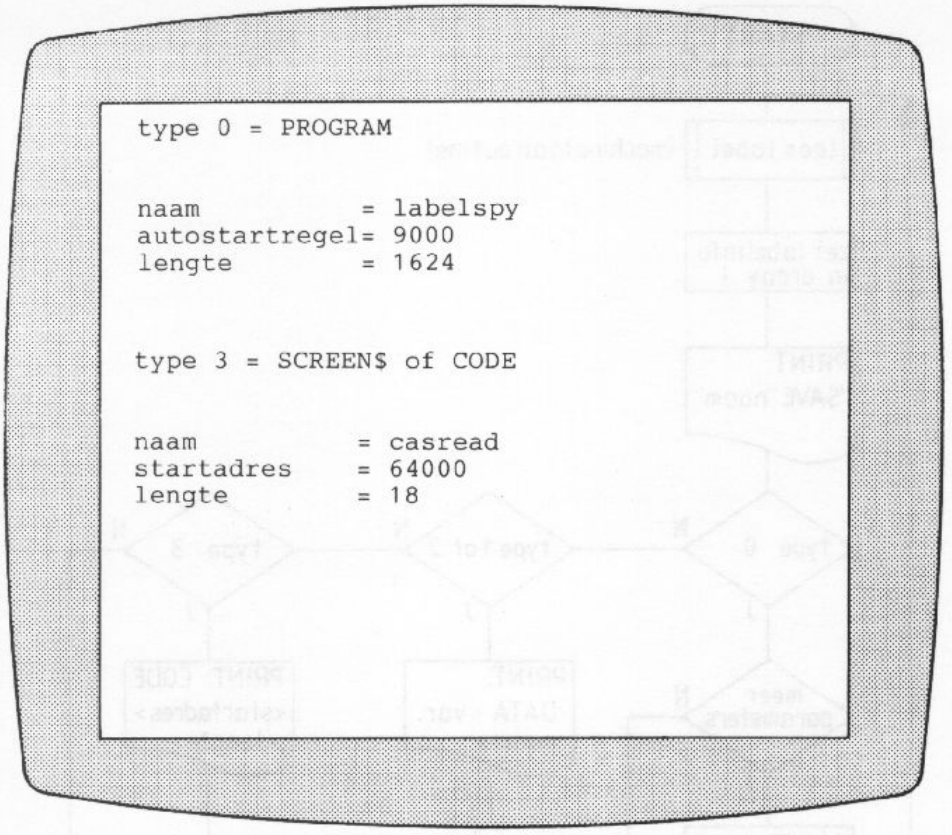
```
 10 RANDOMIZE USR 64000
 20 DIM l(17): CLS
 30 FOR a=1 TO 17
 40 LET l(a)=PEEK (64255+a)
 50 NEXT a
 60 LET type=l(1)
100 IF type=0 THEN  PRINT "type
0 = PROGRAM"
110 IF type=1 THEN  PRINT "type
1 = numerieke DATA"
120 IF type=2 THEN  PRINT "type
2 = string DATA"
130 IF type=3 THEN  PRINT "type
3 = SCREEN$ of CODE"
180 PRINT ''
190 LET n$=""
200 FOR a=2 TO 11
210 LET n$=n$+CHR$ (l(a))
220 NEXT a
230 PRINT "naam          =",n$
300 IF type=0 THEN  GO TO 1000
400 IF type=1 OR type=2 THEN  G
O TO 800
500 IF type=3 THEN  PRINT "star
tadres    =",256*l(15)+l(14)
600 PRINT "lengte         =",256
*l(13)+l(12)
700 GO TO 9080
800 IF l(15)<=95 THEN  PRINT "v
ariabelenaam =",CHR$ l(15)
810 IF l(15)>95 AND l(15)<127 T
HEN  PRINT "variabelenaam =",CHR
$ (l(15)-32)
820 IF l(15)>127 AND l(15)<159
```

```
THEN  PRINT "variabelenaam =",CH
R$ (l(15)-64)
 830 IF l(15)>159 AND l(15)<191
THEN  PRINT "variabelenaam =",CH
R$ (l(15)-96)
 840 IF l(15)>191 AND l(15)<223
THEN  PRINT "variabelenaam =",CH
R$ (l(15)-128)
 850 IF l(15)>223 THEN  PRINT "v
ariabelenaam =",CHR$ (l(15)-160)
 860 GO TO 500
1000 IF l(15)=128 THEN  GO TO 60
0
1010 PRINT "autostartregel=",l(1
5)*256+l(14)
1020 GO TO 600
9000 CLEAR 63999
9010 PRINT "Laat de cassette dra
aien, het   machinetaalprogramma
 moet nog   worden geladen."
9020 LOAD ""CODE
9030 CLS : PRINT  FLASH 1;"stop
de cassetterecorder"
9040 FOR a=1 TO 5
9050 BEEP 1,a
9060 NEXT a
9070 CLS
9080 INPUT "starten? j/n ";a$
9090 IF a$="j" THEN  GO TO 10
9100 GO TO 9080
```

Het resultaat van het uitvoeren van het programma 'labelscrn' is te zien in afbeelding 6-4. Voor deze afbeelding heb ik de cassette waarop het BASIC-programma 'labelspy', zoals ik het zojuist beschreven programma ook wel eens pleeg te noemen, en het bijbehorende machinetaalprogramma 'casread', waarmee je labels van cassettes kunt lezen, gebruikt als input voor datzelfde programma. U ziet dus dat het BASIC-programma slechts 1624 bytes lang is en dat daar dan alleen nog 18 bytes voor het machinetaaldeel bijkomen.

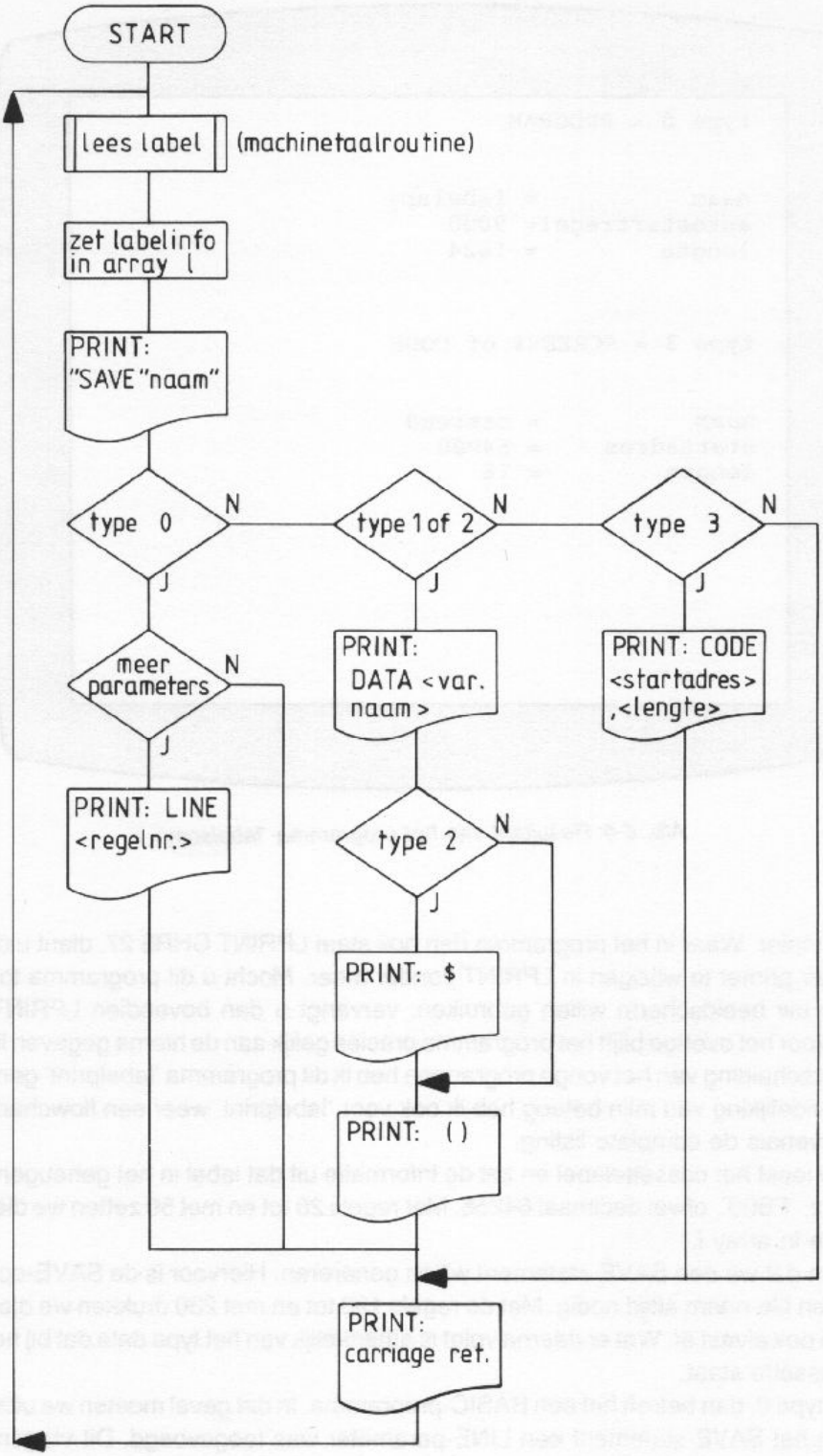
6.2 Het programma 'labelprint'

Voortbordurend op dezelfde basisprincipes heb ik nog een programma gemaakt. Daarbij was de achterliggende gedachte dat ik het programma toch eigenlijk alleen maar wilde gebruiken om van een mij van te voren onbekend programma een kopie te maken. Wat ik dus eigenlijk van mijn programma verwachtte is, dat het de daarvoor benodigde SAVE-statements zou genereren. Omdat op een aantal van mijn cassettes, C60-cassettes, een zeer groot aantal programma's staat, was het waarschijnlijk dat de van het programma verkregen SAVE-statements niet allemaal op een scherm zouden passen. Daar er aan mijn Spectrum een printer is gekoppeld, heb ik dan ook besloten dit programma geschikt te maken voor afdrukken op de printer.
Om te voorkomen dat het programma het op uw computer niet doet, moet mij nog het volgende van het hart. De op mijn Spectrum aangesloten printer is niet de standaard



Afb. 6-4 Resultaat van het programma 'labelscrn'

Sinclair printer. Waar in het programma dan ook staat LPRINT CHR$ 27, dient u dit voor de Sinclair printer te wijzigen in LPRINT zonder meer. Mocht u dit programma toch gewoon op uw beeldscherm willen gebruiken, vervangt u dan bovendien LPRINT door PRINT. Voor het overige blijft het programma precies gelijk aan de hierna gegeven listing.
Ter onderscheiding van het vorige programma heb ik dit programma 'labelprint' genoemd. Ter verduidelijking van mijn betoog heb ik ook voor 'labelprint' weer een flowchart bijgevoegd, evenals de complete listing.
Regel 10 leest het cassettelabel en zet de informatie uit dat label in het geheugen vanaf adres hex. 'FB00', ofwel decimaal 64256. Met regels 20 tot en met 50 zetten we die labelinformatie in array I.
We weten dat we een SAVE-statement willen genereren. Hiervoor is de SAVE-opdracht zelf en een file-naam altijd nodig. Met de regels 190 tot en met 230 drukken we die gegevens dan ook alvast af. Wat er daarna volgt is afhankelijk van het type data dat bij het label op de cassette staat.
Was het type 0, dan betreft het een BASIC-programma. In dat geval moeten we uitzoeken of er aan het SAVE-statement een LINE-parameter was toegevoegd. Dit vragen we in regel 1000. Was er geen LINE toegevoegd, dan is het statement compleet. We geven dan nog de opdracht een carriage return te printen en gaan terug naar het begin van het programma om op regel 10 het volgende label te lezen. In het andere geval drukken we achter het eerder afgedrukte deel van het SAVE-statement de parameter LINE en het bijbe-



Afb. 6-5 Flowchart van het BASIC-programma 'labelprint'

horende regelnummer af. Hiermee is het statement compleet en springen we na het afdrukken van een carriage return terug naar het begin van het programma.
Indien het niet om een BASIC-programma ging, dan kijken we met regel 400 of het soms een variabele betreft. Is dit zo, dan wordt in regels 800 tot en met 850 de tekst DATA afgedrukt en wordt de variabelenaam uitgezocht, geconverteerd naar een normale karakter-code, en afgedrukt. Daar een stringvariabele de toevoeging '$' vereist, controleren we met regel 860 of het om een stringvariabele gaat. Zo ja, dan drukken we het $-teken af. Hierna moet voor zowel een numerieke als string data nog haakjes open en haakjes sluiten '()' worden afgedrukt. Hiermee is het SAVE-statement compleet en wordt na het printen van carriage return teruggesprongen naar regel 10, om het volgende label van de cassette te lezen.
Als geen van beide hiervoor genoemde mogelijkheden van toepassing was, dan moet het zich handelen om een stuk machinetaal ofwel CODE. Daar uit het label niet duidelijk wordt of het een machinetaalprogramma danwel gewoon een aantal bytes (bijvoorbeeld het beeldschermgeheugen) is, printen we met regel 500 altijd CODE, ook als het een SCREEN$ betreft. Omdat dit het enige geval is waarbij we in een startadres en de lengte van de code zijn geïnteresseerd, drukken we direct na CODE het startadres, een komma en de lengte af. Hierna is het SAVE-statement compleet en rest ons alleen nog een carriage return (op regel 9080), waarna we naar regel 10 terugspringen om het volgende label te lezen.
Of u nu een C60-cassette of een C120-cassette in uw recorder afspeelt, 'labelprint' zal net zolang doorgaan tot u het zelf door middel van BREAK en STOP beëindigt.

```
 10 RANDOMIZE USR 64000
 20 DIM l(17): CLS
 30 FOR a=1 TO 17
 40 LET l(a)=PEEK (64255+a)
 50 NEXT a
 60 LET type=l(1)
190 LET n$=""
200 FOR a=2 TO 11
210 LET n$=n$+CHR$ (l(a))
220 NEXT a
230 LPRINT CHR$ 27;"SAVE """;n$
;" """;
300 IF type=0 THEN  GO TO 1000
400 IF type=1 OR type=2 THEN  G
O TO 800
500 IF type=3 THEN  LPRINT CHR$
27;" CODE ";256*l(15)+l(14);","
;256*l(13)+l(12);
700 GO TO 9080
800 IF l(15)<=95 THEN  LPRINT C
HR$ 27;" DATA ";CHR$ l(15);
810 IF l(15)>95 AND l(15)<127 T
HEN  LPRINT CHR$ 27;" DATA ";CHR
$ (l(15)-32);
820 IF l(15)>127 AND l(15)<159
```

```
THEN  LPRINT CHR$ 27;" DATA ";CH
R$ (l(15)-64);
 830 IF l(15)>159 AND l(15)<191
THEN  LPRINT CHR$ 27;" DATA ";CH
R$ (l(15)-96);
 840 IF l(15)>191 AND l(15)<223
THEN  LPRINT CHR$ 27;" DATA ";CH
R$ (l(15)-128);
 850 IF l(15)>223 THEN  LPRINT C
HR$ 27;" DATA ";CHR$ (l(15)-160)
;
 860 IF type=2 THEN  LPRINT CHR$
 27;"$";
 870 LPRINT CHR$ 27;"()";
 880 GO TO 500
1000 IF l(15)=128 THEN  GO TO 60
0
1010 LPRINT CHR$ 27;" LINE ";l(1
5)*256+l(14);
1020 GO TO 700
9000 CLEAR 63999
9010 PRINT  FLASH 1;"Laat de cas
sette draaien, het  machinetaal
programma moet nog  worden gela
den."
9020 LOAD "printcode"CODE
9030 RANDOMIZE USR 65150
9034 LOAD "casread"CODE
9036 PRINT  FLASH 1;"STOP CASSET
TE"
9040 BEEP 3,3
9050 CLS
9060 INPUT "aantal karakters per
 regel? ";k
9070 POKE 64869,k
9080 LPRINT : PAUSE 100
9090 GO TO 10
9500 SAVE "labelprint" LINE 9000
: SAVE "printcode"CODE 64736,632
: SAVE "casread"CODE 64000,18
```

Afbeelding 6-6 geeft een voorbeeld van de output van het programma 'labelprint'.

```
SAVE "labelprint " LINE 9000
SAVE "printcode  " CODE 64736,632
SAVE "casread    " CODE 64000,18

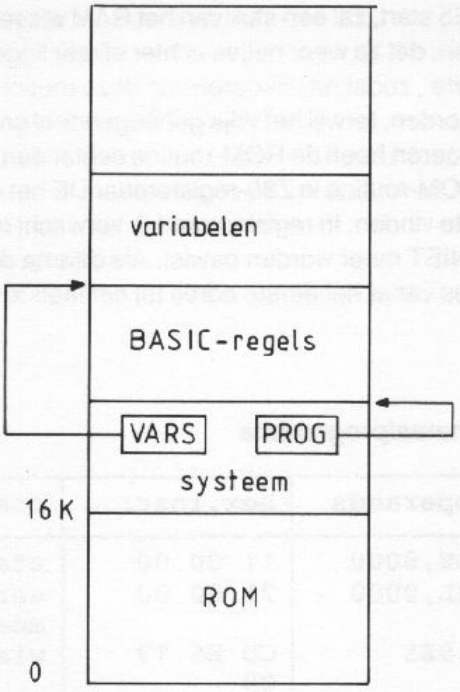
SAVE "array x    " DATA X()
SAVE "array y    " DATA Y$()

SAVE "hexlader+  "
SAVE "machtaal   "
```

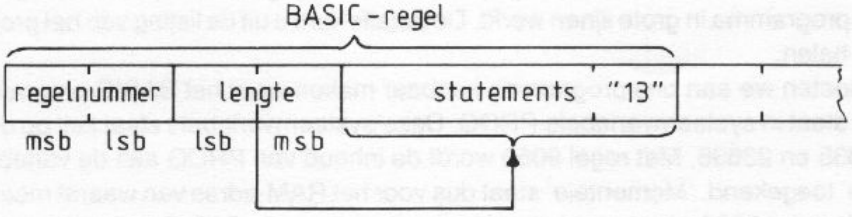
Afb. 6-6 Resultaat van het programma 'labelprint'

7. Selectief wissen van BASIC-regels

Een REM-statement heeft alleen het voordeel dat het een listing van een BASIC-programma kan verduidelijken. Staan er echter in een programma dat wordt uitgevoerd nog steeds REM-statements, dan zullen die ervoor zorgen dat het programma trager wordt uitgevoerd dan wanneer die REM-statements er niet in staan. Deze wetenschap bracht mij er toe een programma te ontwikkelen waarmee je zonder veel moeite alle REM-regels uit een in het geheugen staand programma kunt verwijderen.
Om de werking van dit programma te kunnen begrijpen is enige achtergrondkennis van het geheugengebruik en de lay-out van BASIC-regels nodig. Hebt u die kennis eenmaal, dan zult u ook in staat zijn zelf nieuwe programma's te schrijven die aan uw specifieke wensen voldoen.
Het in dit hoofdstuk te beschrijven programma heb ik 'seldel' genoemd; 'sel' van selective en 'del' van delete, hetgeen Engels is voor 'verwijderen'. Om het programma eenvoudig en snel te houden heb ik ook nu weer gebruik gemaakt van een in de Spectrum aanwezige ROM-routine. Doch voor ik hierop verder inga, wil ik u eerst wat algemene informatie over de plaats van een BASIC-programma binnen het geheugen van de Spectrum geven en wil ik de lay-out van een BASIC-regel met u doornemen.
De Spectrum bestaat in twee versies, 16K en 48K. Dit wil zeggen dat er machines zijn met 16K RAM en met 48K RAM. Doch beide hebben ze daarnaast nog 16K ROM. De hoeveelheid RAM die op een gegeven moment in gebruik is, is geheel afhankelijk van het geladen programma en van de configuratie van de hardware. Dit laatste behoeft misschien enige toelichting. Welnu, zoals u weet kun je met de standaard ZX Spectrum alleen I/O plegen met het beeldscherm, het toetsenbord en de cassetterecorder. Indien de Spectrum wordt uitgebreid met een 'Interface 1', dan is er ook I/O mogelijk met de microdrives en de serie-interface (RS232 en LAN). In het systeemdeel van het RAM wordt voor ieder I/O-device een aantal bytes gebruikt, voor administratie en buffers. Een BASIC-programma start dan ook niet bij iedere Spectrum op het zelfde RAM-adres. Dit hangt af van het feit of er al dan niet een hardware uitbreiding op die Spectrum is aangesloten.
U ziet dus dat er binnen de Spectrum praktisch geen vaste adressen zijn aan te geven. Vandaar dat er om een bepaald geheugengebied te vinden gebruik wordt gemaakt van zogenaamde pointers. De meeste van deze pointers, wijzers naar een bepaald gebied, staan in het direct op het ROM volgende geheugengebied.
In afbeelding 7-1 ziet u vlak boven het ROM-gebied, dat van adres 0 tot 16K gaat, het systeemdeel. In dat deel bevinden zich onder meer het beeldschermgeheugen, de attributen en de systeemvariabelen. Om die laatste gaat het nu.
In het programma 'seldel' worden twee systeemvariabelen gebruikt, PROG en VARS. PROG bevat het adres van het eerste byte van de eerste BASIC-regel in het geheugen. VARS bevat het adres van het eerste byte van het variabelengeheugen. Het adres in PROG is zoals hiervoor al werd opgemerkt afhankelijk van de configuratie. VARS is afhankelijk van diezelfde configuratie, maar bovendien afhankelijk van de lengte van het BASIC-programma.
Zoals gezegd, wijst PROG naar het eerste byte van de eerste BASIC-regel. Wij weten dus waar het begin van een BASIC-regel is. Afbeelding 7-2 laat de indeling van een BASIC-regel zien. Iedere regel bestaat uit vier delen. De eerste twee bytes bevatten het regelnummer. Dit regelnummer mag maximaal 9999 zijn, zodat het nodig was om hiervoor twee bytes te reserveren. In een byte kan immers maximaal slechts 255 staan. Het eerste



Afb. 7-1 Indeling van het BASIC-geheugen



Afb. 7-2 Opbouw van een BASIC-regel

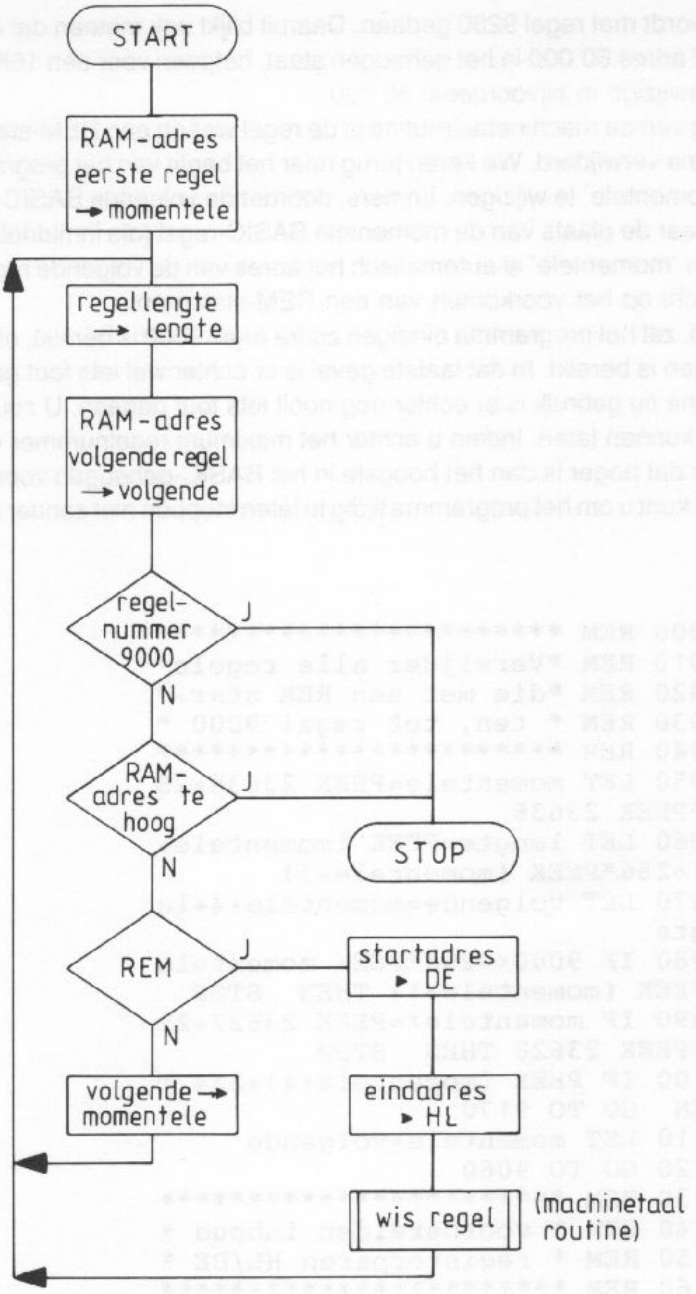
byte van het regelnummer is het meest significant (msb). Het tweede byte is het minst significant (lsb). Het derde en vierde byte van de BASIC-regel bevatten de lengte van de rest van de regel. In tegenstelling tot het regelnummer is hier het eerste byte van de lengte het minst significant (lsb), terwijl het tweede byte van de lengte het meest significant is (msb). Na het regelnummer en de lengte komen de statements. Er kunnen meerdere statements in een BASIC-regel staan, van elkaar gescheiden door een dubbele punt. Het einde van een BASIC-regel wordt aangegeven met de code 13 (hexadecimaal '0D'). Dit is de code voor de ENTER-toets. Hierna volgt een volgende BASIC-regel, of, indien dit de laatste regel was, het variabelengeheugen.
Gewapend met deze kennis zullen we het programma eens onder de loep nemen. Zoals in het begin al opgemerkt, gaat het hier weer om een programma dat uit een BASIC- en een machinetaaldeel bestaat. Om de zin van enkele van de BASIC-statements te kunnen inzien, is het goed eerst eens te kijken wat het machinetaalprogramma precies doet. De machinetaalroutine roept op een gegeven ogenblik een ROM-routine aan. Deze ROM-

routine, die op adres 19E5 start, zal een stuk van het RAM wissen en vervolgens de gegevens zodanig verschuiven, dat ze weer netjes achter elkaar liggen. Ook worden alle pointers waar nodig 'ge-update', zodat na uitvoeren van deze routine het stuk in gebruik zijnde geheugen kleiner is geworden, terwijl het vrije geheugendeel groter is geworden. Om zijn taak goed te kunnen uitvoeren heeft de ROM-routine echter een aantal ingangsgegevens nodig. Zo verwacht de ROM-routine in Z80-registerpaar DE het adres van het eerste byte dat moet worden gewist te vinden. In registerpaar HL verwacht de ROM-routine het adres van het eerste byte dat NIET moet worden gewist. Als daarna de ROM-routine wordt gestart, dan zullen alle bytes vanaf het eerste adres tot het laatste uit het geheugen worden verwijderd.

Listing van het machinetaalprogramma

```
adres   mnem    operands    hex.instr.    commentaar
60000   LD      DE,0000     11 00 00      startadres --> DE
60003   LD      HL,0000     21 00 00      eerste adres dat niet
                                          moet worden gewist.
60006   CALL    19E5        CD E5 19      wis-routine in ROM
60009   RET                 C9
```

Dan komen we nu tot de beschrijving van het BASIC-programma. In afbeelding 7-3 ziet u hoe het programma in grote lijnen werkt. De details kunt u uit de listing van het programma 'seldel' halen.
Eerst moeten we aan ons programma kenbaar maken waar het BASIC-programma begint. Dit staat in systeemvariabele PROG. Deze systeemvariabele staat zelf op de adressen 23635 en 23636. Met regel 9050 wordt de inhoud van PROG aan de variabele 'momentele' toegekend. 'Momentele' staat dus voor het RAM-adres van waaraf moet worden gewist. In regel 9060 zetten we de lengte van de rest van de BASIC-regel die op het adres 'momentele' start in variabele 'lengte'. Met deze twee gegevens zijn we in staat om het beginadres van de volgende BASIC-regel uit te zoeken. Dit wordt in regel 9070 gedaan. De variabele 'volgende' wordt daar geladen met het startadres van de volgende BASIC-regel.
In feite zouden we nu voldoende weten om de machinetaalroutine te starten. Echter, er kunnen zich problemen voordoen. Stel dat het regelnummer van de te wissen BASIC-regel 9000 of hoger is. Zoals u ziet begint ons programma op regel 9000. Daar het niet de bedoeling is dat we het programma 'seldel' gaan wissen, moet er een beveiliging worden ingebouwd om dat tegen te gaan. Deze beveiliging zit in regel 9080. Zodra regel 9000 is bereikt stopt het programma. Het zou ook mogelijk zijn dat, door welke oorzaak dan ook, het programma in de war raakt. Om nu te voorkomen dat er gekke dingen gaan gebeuren zit er in regel 9090 een tweede beveiliging. Hier wordt namelijk gekeken waar het variabelengeheugen begint en zodra het adres van de te wissen BASIC-regel binnen het variabelengeheugen valt, wordt de verdere uitvoering van het programma gestopt.
Na deze beveiligingen komen we aan de vraag waar het eigenlijk om gaat, namelijk: is er sprake van een REM-statement? De decimale code van REM is 234 en u ziet dat in regel 9100 wordt gechecked of er sprake is van de REM-code of niet. Staat er op de vijfde posi-



Afb. 7-3 Flowchart van het programma 'seldel'

tie van de BASIC-regel (momentele+4) geen REM, dan zetten we het adres van de volgende regel in 'momentele' en keren we terug naar het begin van het programma op regel 9060, waar het hele verhaal opnieuw begint. Was er wel sprake van een REM-statement, dan gaan we naar regel 9170.

Met de regels 9170 tot en met 9240 worden de adressen van 'momentele' en 'volgende' in het machinetaalprogramma ge-POKEd. Hiervoor is een eenvoudige conversie nodig, omdat we een waarde van meer dan 255 in twee adressen moeten plaatsen. Zodra deze adressen in de machinetaalroutine zijn ingevuld, kan die machinetaalroutine zelf worden

gestart. Dit wordt met regel 9290 gedaan. Daaruit blijkt ook meteen dat die machinetaalroutine vanaf adres 60 000 in het geheugen staat, hetgeen voor een 16K Spectrum dient te worden gewijzigd in bijvoorbeeld 30 000.
Na uitvoering van de machinetaalroutine is de regel waarin een REM-statement stond uit het programma verwijderd. We keren terug naar het begin van het programma zonder de variabele 'momentele' te wijzigen. Immers, doordat de volgende BASIC-regel nu is terug geschoven naar de plaats van de momentele BASIC-regel (die inmiddels is verdwenen), is het adres in 'momentele' al automatisch het adres van de volgende regel die moet worden onderzocht op het voorkomen van een REM-statement.
Zoals gezegd, zal het programma eindigen zodra regel 9000 is bereikt, of zodra het variabelengeheugen is bereikt. In dat laatste geval is er echter wel iets fout gegaan. Zolang ik het programma nu gebruik is er echter nog nooit iets fout gegaan. U zou regel 9090 dan ook wel weg kunnen laten. Indien u echter het maximum regelnummer verhoogt tot een regelnummer dat hoger is dan het hoogste in het BASIC-geheugen voorkomende regelnummer, dan kunt u om het programma tijdig te laten stoppen niet zonder regel 9090.

```
9000 REM ************************
9010 REM *Verwijder alle regels*
9020 REM *die met een REM star-*
9030 REM * ten, tot regel 9000 *
9040 REM ************************
9050 LET momentele=PEEK 23635+25
6*PEEK 23636
9060 LET lengte=PEEK (momentele+
2)+256*PEEK (momentele+3)
9070 LET volgende=momentele+4+le
ngte
9080 IF 9000<=256*PEEK momentele
+PEEK (momentele+1) THEN  STOP
9090 IF momentele>=PEEK 23627+25
6*PEEK 23628 THEN  STOP
9100 IF PEEK (momentele+4)=234 T
HEN  GO TO 9170
9110 LET momentele=volgende
9120 GO TO 9060
9130 REM ************************
9140 REM * voorbereiden inhoud *
9150 REM * registerparen HL/DE *
9160 REM ************************
9170 LET de1=INT (momentele/256)
9180 LET de2=momentele-256*de1
9190 POKE 60001,de2
9200 POKE 60002,de1
9210 LET hl1=INT (volgende/256)
9220 LET hl2=volgende-256*hl1
9230 POKE 60004,hl2
9240 POKE 60005,hl1
9250 REM ************************
9260 REM * verwijder momentele *
9270 REM * regel tot volgende. *
9280 REM ************************
```

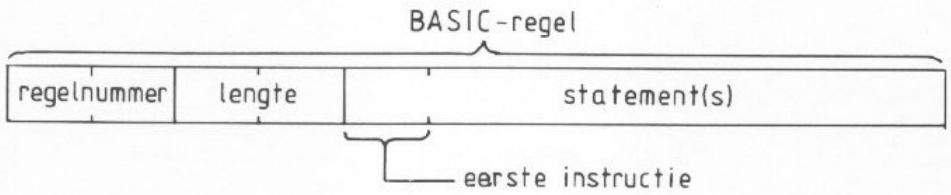
```
9290 RANDOMIZE USR 60000
9300 GO TO 9060
9900 CLEAR 59999: CLS : PRINT "L
AAT DE CASSETTE DRAAIEN.       D
E MACHINETAALROUTINE MOET NOG  W
ORDEN GELADEN."
9910 LOAD "seldelcode"CODE
9920 CLS : PRINT  FLASH 1;"STOP
DE CASSETTERECORDER."
9930 FOR a=1 TO 5
9940 BEEP 1,a
9950 NEXT a
9960 STOP
```

8. Hernummeren van BASIC-regelnummers

Indien u de programma's uit de voorgaande hoofdstukken hebt gebruikt om uw eigen BASIC-programma's te wijzigen, of gewoon om de REM-statements eruit te halen, dan zal de regelnummering van die programma's niet meer netjes oplopen, met steeds eenzelfde verschil in regelnummer per regel. Hoewel dit niet netjes staat, kan het geen kwaad. Anders wordt dat wanneer u een programma was begonnen met steeds een verschil van 10 tussen iedere regel, en wanneer u naderhand tot de ontdekking kwam dat er nog een aantal regels moesten worden tussengevoegd. In dat geval kon u slechts 9 regels tussenvoegen, doch daarna ontstonden problemen.
Daarom zullen we nu een programma onder de loep nemen waarmee we op zeer eenvoudige wijze de regelnummering van ons programma kunnen wijzigen. De Spectrum mist jammer genoeg zo'n hernummer-mogelijkheid. Het programma 'hernummer' geeft u echter alle voor het hernummeren gewenste mogelijkheden.
Voordat we het programma gaan bespreken, wil ik eens op een rijtje zetten wat onze eisen aan zo'n hernummerprogramma zijn.

1. Het moet niet alleen mogelijk zijn een heel programma te hernummeren, het moet ook mogelijk zijn een willekeurig deel uit een programma te hernummeren.
2. Indien wordt gekozen voor het hernummeren van een heel programma, dan mag het aantal regels bij een opgegeven verhogingsfactor niet groter zijn dan voor de Spectrum regelnummering is toegestaan. (Het hoogste toegestane regelnummer is 9999.)
3. Indien wordt gekozen voor het hernummeren van een deel van het programma, dan mogen de nieuw te genereren regelnummers de bestaande regelnummers niet overlappen.
4. Indien hernummering door één van de hiervoor gestelde eisen niet mogelijk is, dan moet de operator daarop worden gewezen en moet hem de mogelijkheid worden geboden het hernummeren opnieuw te starten met andere regelnummers of verhogingsfactoren.
5. Indien een regel waarin een GO TO of een GO SUB statement staat wordt hernummerd, dan wil ik daarvan een melding krijgen, opdat ik later waar nodig deze instructies kan aanpassen aan de nieuw verkregen regelnummers.

Het in dit hoofdstuk te beschrijven hernummerprogramma voldoet aan alle hiervoor gestelde eisen. Hoewel het mogelijk was geweest het hele programma in BASIC te schrijven (het eerste hernummerprogramma dat ik ooit schreef was inderdaad geheel in BASIC), kan de verwerkingssnelheid aanzienlijk worden verbeterd door gebruik te maken van machinetaalroutines die op hun beurt weer ROM-routines aanroepen. Dat er ook in het hier gegeven programma weer ROM-routines worden gebruikt, heeft dan ook een verhoogde 'performance' tot gevolg.



Afb. 8-1 Indeling van een BASIC-regel

Afbeelding 8-1 laat nog eens zien hoe een BASIC-regel in het geheugen er uitziet. Daar wij de regelnummers van de BASIC-regels gaan wijzigen, zullen we ons goed moeten realiseren wat we precies aan het doen zijn. U ziet dat een BASIC-regel begint met een regelnummer. Dat regelnummer staat in twee bytes opgeslagen. Het eerste byte, het meest links in de tekening, heeft de meeste significantie. Dit wil zeggen dat als er in dat byte een waarde 5 zou staan, dat deze waarde dan met een factor 256 moet worden vermenigvuldigd om de werkelijke waarde te verkrijgen. Het tweede byte heeft de minste significantie. Dat wil zeggen dat indien hier de waarde 20 zou staan dat dan ook werkelijk 20 wordt bedoeld. Als we de zojuist genoemde waarden als voorbeeld nemen, dan zou het regelnummer dus zijn:

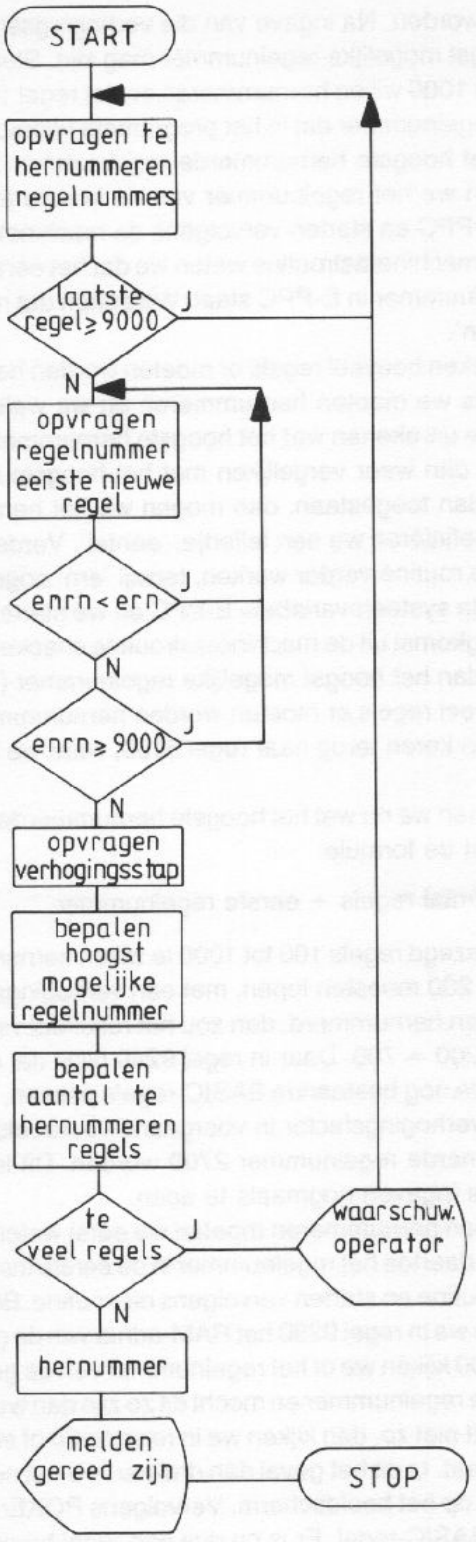
5 * 256 + 20 = 1280 + 20 = 1300.

Als dit duidelijk is, dan wil ik verder nog even wijzen op de plaats waar een instructie begint. Zoals u bekend zal zijn, wordt een instructie in de Spectrum opgeslagen in één byte. Iedere instructie heeft een specifieke code. Die code kan liggen tussen 0 en 255. In ons programma zijn we alleen geïnteresseerd in de GO TO- en GO SUB-instructies. Deze instructies zullen in het geheugen staan opgeslagen als een waarde 236 voor GO TO en een waarde 237 voor een GO SUB.
De eerste instructie in een BASIC-regel staat altijd in het vijfde byte van die regel. Dit is ook aangegeven in afb. 8-1. Het is weliswaar mogelijk om meerdere instructies in een regel te zetten, doch in het hier beschreven programma heb ik alle volgende instructies in eenzelfde regel buiten beschouwing gelaten. Mocht u zelf het programma zodanig willen uitbreiden dat het alle instructies vindt, dan dient u de volgende mogelijkheden in aanmerking te nemen.
Instructies staan òf aan het begin van een regel (op het vijfde byte), òf achter een dubbele punt. De code voor een dubbele punt is 58. Het is echter niet zo dat achter iedere dubbele punt een instructie staat. Immers als die dubbele punt deel uitmaakt van een tekst-string, dan kan er heel goed gewoon een letter achter staan. Bovendien is het ook nog zo dat er na de eerste instructie andere instructies kunnen voorkomen die niet achter een dubbele punt staan. Denk hierbij maar aan de constructie IF...THEN <instructie>. U merkt het al, het zoeken naar bijv. GO TO's en GO SUB's, voor zover die niet als eerste instructie van de regel voorkomen, is een ingewikkelde zaak. Het is dan ook de vraag of het de moeite waard is om iedere instructie op te zoeken, omdat hierdoor de verwerkingssnelheid sterk wordt vertraagd.
Vertraging is nu juist het tegengestelde van wat we wensen. Om wat meer vaart in het programma te krijgen heb ik dan ook gezocht naar passende ROM-routines. Ik heb er twee gevonden die goed te gebruiken waren bij het hernummeren. Voordat we nu het hele programma al kennen, wil ik toch even iets over die routines vertellen.
De eerste ROM-routine die we zullen gebruiken heeft de volgende functie. Indien wij deze ROM-routine opdragen het regelnummer uit de systeemvariabele E-PPC te lezen, dan zal de routine in het BASIC-programmageheugen kijken wat het eerste regelnummer is dat onmiddellijk volgt op het in E-PPC opgegeven regelnummer. Om deze ROM-routine de systeemvariabele E-PPC te laten gebruiken moeten wij met behulp van een machinetaalroutine het adres van E-PPC in register HL plaatsen. In de listing van de machinetaalroutine 'hernumcode' ziet u dat in de eerste instructie van het eerste deel van die routine gebeuren.

adres	mnem	operands	hex.code	commentaar
60000	LD	HL,5C49	21 49 5C	adres van E-PPC --> HL
60003	CALL	190F	CD 0F 19	zoek volgend regelnr. (resultaat in E-PPC)
60006	RET		C9	
60007	LD	HL,0000	21 00 00	hier wordt regelnr. in ge-POKEd.
60010	CALL	196E	CD 6E 19	zoek RAM-adres
60013	LD	B,H	44	HL --> BC
60014	LD	C,L	4D	
60015	RET		C9	

De tweede routine die we gaan gebruiken, zoekt van het regelnummer dat we hem in register HL aanbieden het startadres op. Dit startadres staat na uitvoering van de ROM-routine in registerpaar HL. Om ons in staat te stellen dit adres op eenvoudige wijze in ons BASIC-programma op te nemen en verder te verwerken, zou het prettig zijn als het in registerpaar BC had gestaan. Daar valt echter wel wat aan te doen. In de machinetaal listing ziet u op adressen 60013 en 60014 twee instructies staan die de inhoud van register H naar B en de inhoud van L naar C overbrengen. Hiermee hebben we er dus voor gezorgd dat het door de ROM-routine gevonden adres in registerpaar BC staat op het moment dat we terugkeren naar het BASIC-programma.

Laten we dan nu het BASIC-programma eens aan een nader onderzoek onderwerpen. Afbeelding 8-2 is de flowchart van dat programma. U kunt dat gebruiken om de grote lijnen van het programma vast te houden. Tevens zullen we zo nu en dan in enige technische details duiken, waarbij ik de regelnummers uit de aan het eind van dit hoofdstuk gegeven BASIC-programmalisting zal gebruiken.
Eerst moet worden uitgezocht wat moet worden hernummerd. In regels 9010 tot en met 9030 wordt het eerste te hernummeren regelnummer aan de operator gevraagd. Dit nummer wordt netjes op het beeldscherm weergegeven. Daarna wordt om het laatste te hernummeren regelnummer gevraagd. Voordat dit nummer op het beeldscherm kan worden weergegeven moet worden gecontroleerd of het nummer wel lager dan 9000 is. Immers het hernummerprogramma start zelf op regelnummer 9000 en we willen alleen het daarvoor staande programma hernummeren, niet het hernummerprogramma zelf. Als het laatste regelnummer beneden de 9000 is, dan zal dat nummer ook netjes op het beeldscherm worden afgedrukt.
Nu bekend is welke regels moeten worden hernummerd, moet nog worden uitgezocht wat de nieuwe regelnummers daarvoor moeten worden. De vraag 'Eerste nieuwe regelnummer?' verschijnt. Indien u nu een regelnummer ingeeft dat lager is dan het laagste regelnummer van de te hernummeren regels, dan zal dat niet worden geaccepteerd. U mag een ander nummer ingeven. Pas wanneer u een regelnummer opgeeft dat hoger is dan of gelijk is aan de laagste te hernummeren regel, wordt dat geaccepteerd. Doch voordat het helemaal wordt goedgekeurd wordt nog gecontroleerd of het wel lager dan 9000 is. Is ook dat het geval dan wordt het opgegeven nieuwe regelnummer op het beeldscherm afgedrukt.
Er blijft nu nog één vraag over, namelijk wat de verhogingsstap tussen de nieuw te num-



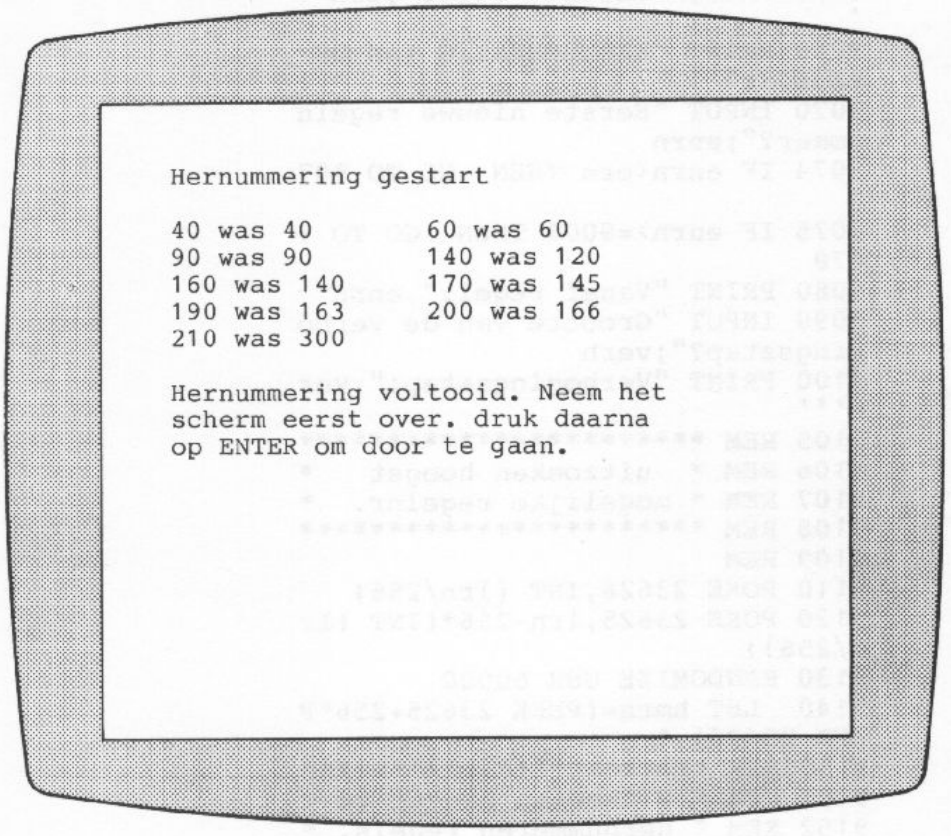
Afb. 8-2 Flow chart van het programma 'hernummer'

meren regels moet worden. Na ingave van die verhogingsstap gaat het programma uitzoeken wat het hoogst mogelijke regelnummer mag zijn. Stel dat we hadden opgegeven dat we regels 100 tot 1000 willen hernummeren en dat regel 1000 niet voorkomt, doch dat het eerstvolgende regelnummer dat in het programma voorkomt 3000 is. In dat geval zou na hernummeren het hoogste hernummerde regelnummer 2999 mogen zijn. In regels 9110 en 9120 zetten we het regelnummer van de laatste te hernummeren regel in de systeemvariabele E-PPC en starten vervolgens de machinetaalroutine op adres 60000. Bij terugkeer van de machinetaalroutine weten we dat het eerstvolgende regelnummer na het opgegeven regelnummer in E-PPC staat. We zetten dat regelnummer nu van E-PPC in de variabele 'hmrn'.
Nu moeten we uitzoeken hoeveel regels er moeten worden hernummerd. Als we namelijk weten hoeveel regels we moeten hernummeren en we weten de verhogingsfactor per regel, dan kunnen we uitrekenen wat het hoogste hernummerde regelnummer wordt. Dit nummer kunnen we dan weer vergelijken met het hoogste regelnummer dat is toegestaan. Is het hoger dan toegestaan, dan mogen we het hernummeren niet starten.
Voor dat uitzoeken definiëren we een tellertje, 'aantal'. Verder zetten we 'ern' in 'rn'. Met 'rn' zullen we in deze routine verder werken, terwijl 'ern' ongewijzigd zal blijven bestaan. Nu zetten we 'rn' in de systeemvariabele E-PPC en we starten de machinetaalroutine op adres 60000. Bij terugkomst uit de machinetaalroutine checken we of het gevonden regelnummer al groter is dan het hoogst mogelijke regelnummer ('rn'>'hmrn'). Is dat zo, dan staat in 'aantal' hoeveel regels er moeten worden hernummerd. Is dat niet zo, dan tellen we 1 op bij 'aantal' en keren terug naar regel 9160, waar we het volgende regelnummer gaan opzoeken.
In regel 9230 berekenen we nu wat het hoogste hernummerde regelnummer zou worden. Dit wordt gedaan met de formule:

verhogingsfactor * aantal regels + eerste regelnummer.

Stel dat we hadden gezegd regels 100 tot 1000 te willen hernummeren, waarbij de nieuwe regelnummers vanaf 200 moesten lopen, met een verhogingsfactor van 10, terwijl er 50 regels moesten worden hernummerd, dan zou het resultaat van de hiervoor gegeven formule zijn: 10 * 50 + 200 = 700. Daar in regel 9240 blijkt dat deze hernummerde regels zonder meer tussen de nog bestaande BASIC-regels passen, zal het hernummeren worden gestart. Zou de verhogingsfactor in voorgaand voorbeeld 50 zijn geweest, dan zou het hoogste hernummerde regelnummer 2700 worden. Dit is te hoog en dus gaan we terug naar af, om alle ingaven nogmaals te doen.
Om een regel te kunnen hernummeren moeten we eerst weten op welk adres in RAM die regel start. We zetten daartoe het regelnummer in de eerste instructie van het tweede deel van de machinetaalroutine en starten vervolgens die routine. Bij terugkeer uit de machinetaalroutine ontvangen we in regel 9290 het RAM-adres van de gevraagde regel in variabele 'adres'. In regel 9300 kijken we of het regelnummer van de gevonden regel hoger is dan het hoogst toegestane regelnummer en mocht dit zo zijn dan weten we dat het hernummeren voltooid is. Was dit niet zo, dan kijken we in regel 9320 of er in de gevonden regel een GO TO of GO SUB staat. Is dit het geval dan drukken we het nieuwe regelnummer en het oude regelnummer af op het beeldscherm. Vervolgens POKEn we het nieuwe regelnummer in de gevonden BASIC-regel. Er is nu dus een regel hernummerd. Om de volgende regel te kunnen hernummeren moeten we het adres daarvan zien te vinden. Dit wordt met regel 9350 gedaan. Met regel 9360 wordt dan nog het regelnummer met de verhogingsfactor verhoogd, zodat de volgende hernummerde regel het juiste nummer zal krijgen.

Om u een indruk te geven van het resultaat van dit programma, heb ik een voorbeeld van de output naar het beeldscherm bijgevoegd (zie afbeelding 8-3).



Afb. 8-3 Resultaat van het programma 'hernummer'

BASIC-programmalisting 'hernummer'

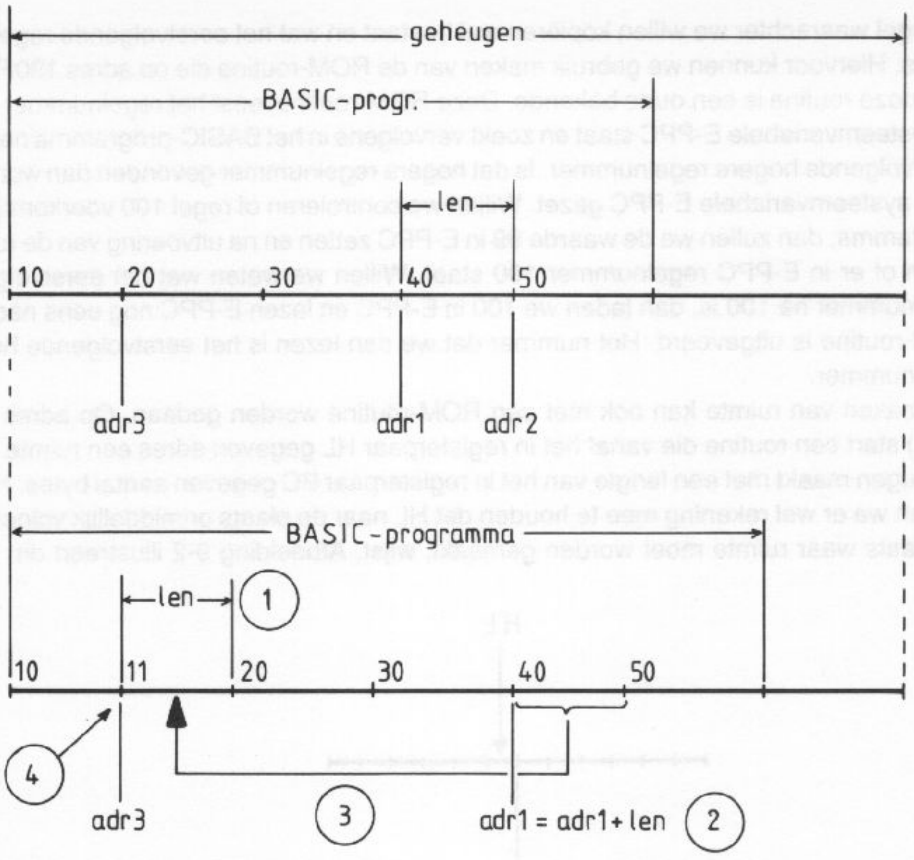
```
9000 REM ************************
9001 REM *  regelhernummering   *
9002 REM ************************
9003 REM *ern =eerste regelnr. *
9004 REM *lrn =laatste regelnr.*
9005 REM *enrn=eerste nieuwe rn*
9006 REM *verh=verhogingsstap  *
9007 REM ************************
9008 REM
9009 CLS
9010 PRINT "TE HERNUMMEREN REGEL
S:"''
9020 INPUT "Eerste te hernummere
n regel?";ern
9030 PRINT "Eerste regel",ern
9040 INPUT "Laatste te hernummer
en regel?";lrn
```

```
9045 IF lrn>=9000 THEN  GO TO 90
40
9050 PRINT "Laatste regel",lrn''
'
9060 PRINT "HERNUMMERING VAN DE
REGELS:"''
9070 INPUT "Eerste nieuwe regeln
ummer?";enrn
9074 IF enrn<ern THEN  GO TO 907
0
9075 IF enrn>=9000 THEN  GO TO 9
070
9080 PRINT "Vanaf regel:",enrn
9090 INPUT "Grootte van de verho
gingsstap?";verh
9100 PRINT "Verhogingsstap:",ver
h'''
9105 REM **********************
9106 REM *  uitzoeken hoogst   *
9107 REM * mogelijke regelnr.  *
9108 REM **********************
9109 REM
9110 POKE 23626,INT (lrn/256)
9120 POKE 23625,lrn-256*(INT (lr
n/256))
9130 RANDOMIZE USR 60000
9140  LET hmrn=(PEEK 23625+256*P
EEK 23626)-1
9150>REM **********************
9151 REM * uitzoeken aantal te *
9152 REM * hernummeren regels. *
9153 REM **********************
9154 REM
9155 LET aantal=0: LET rn=ern
9160 POKE 23626,INT (rn/256)
9170 POKE 23625,rn-256*(INT (rn/
256))
9180 RANDOMIZE USR 60000
9190 LET rn=PEEK 23625+256*PEEK
23626
9200 IF rn>hmrn THEN  GO TO 9230
9210 LET aantal=aantal+1
9220 GO TO 9160
9230 LET hrn=verh*aantal+enrn
9240 IF hrn>hmrn THEN  PRINT "He
t hoogste gehernummerde regel-nu
mmer zou ";hrn;" worden."''"Daar
 dit te hoog is moet u de  verh
ogingsstap kleiner maken."''"Dru
k ENTER om door te gaan.": PAUSE
 0: GO TO 9000
9250 REM **********************
9251 REM * Hernummeren mag nu  *
9252 REM *   worden gestart    *
9253 REM **********************
```

```
9254 REM * eerst RAM-adres van *
9255 REM *eerste te hernummeren*
9256 REM *   regel opzoeken.   *
9257 REM **********************
9258 REM
9259 CLS : PRINT "Hernummering g
estart"'''
9260 LET rn=ern
9270 POKE 60009,INT (rn/256)
9280 POKE 60008,rn-256*(INT (rn/
256))
9290 LET adres=USR 60007
9295 LET rn=enrn
9300 LET orn=256*PEEK adres+PEEK
 (adres+1)
9310 IF orn>hmrn THEN  GO TO 980
0
9320 IF PEEK (adres+4)=236 OR PE
EK (adres+4)=237 THEN  PRINT rn;
" was ";orn,
9330 POKE adres,INT (rn/256)
9340 POKE (adres+1),rn-256*(INT
(rn/256))
9350 LET adres=adres+(PEEK (adre
s+2)+256*PEEK (adres+3))+4
9360 LET rn=rn+verh
9370 GO TO 9300
9800 PRINT '''"Hernummering volto
oid. Neem het scherm eerst over.
 druk daarna  op ENTER om door t
e gaan.": PAUSE 0
9810 STOP
9900 LOAD ""CODE
9910 CLS : PRINT  FLASH 1;"STOP
CASSETTERECORDER"
9920 BEEP 3,1
9930 STOP
```

9. Het kopiëren van BASIC-regels

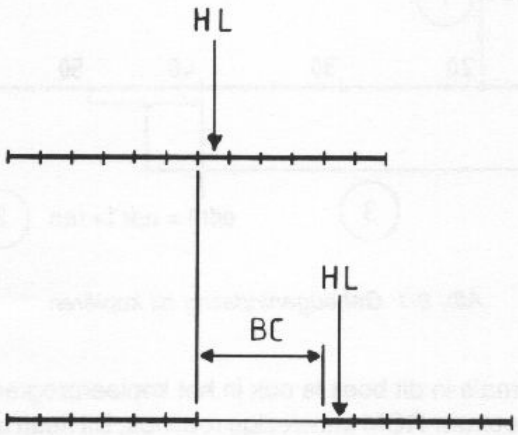
Bij het schrijven van BASIC-programma's komt het nogal eens voor dat een bepaalde routine op verschillende plaatsen in dat programma nodig is. Doordat er, afhankelijk van de functie van die routines kleine onderlinge verschillen in de routines optreden kunnen ze niet door één subroutine worden vervangen. Dit heeft dan tot gevolg dat men soms grote stukken van het programma meermalen moet intikken, soms met als enig verschil een paar variabelenamen. Dit bracht mij op het idee een programma te maken waarmee ik kan aangeven welke regels ik gekopieerd wil hebben en waar die regels in mijn programma naartoe gekopieerd moeten worden.
Het idee was eenvoudig, de uitvoering iets ingewikkelder. Voordat we het programma gaan bekijken zullen we eens het probleem gaan analyseren. In afbeelding 9-1 is het geheugen tweemaal weergegeven. Bovenin de afbeelding zien we dat er een programma in het geheugen staat. Dat programma bestaat uit 5 regels, de regels 10 tot en met 50. Stel nu dat het mijn bedoeling is om regel 40 te kopiëren naar de plaats onmiddellijk volgend op regel 10. Dan zal ik moeten weten hoe lang regel 40 is, waar hij begint en waar hij naartoe moet. Als ik nu weet op welk adres regel 40 begint en op welk adres regel 50 begint, dan kan ik uitrekenen hoe lang regel 40 is door het startadres van regel 40 (adr1) van dat van regel 50 (adr2) af te trekken. Het startadres van regel 20 (volgt onmiddellijk op regel 10) bewaren we zolang in de variabele adr3.
Om te voorkomen dat ik met het kopiëren van regel 40 regel 20 overschrijf, zal ik er nu eerst voor moeten zorgen dat er na regel 10 (dus vanaf adr3) voldoende ruimte vrij komt om er regel 40 naartoe te schrijven. In het onderste deel van afbeelding 9-1 is deze actie aangegeven met de omcirkelde 1. Wat er dus eigenlijk gebeurde is, dat alle BASIC-regels vanaf regel 20 een aantal plaatsen naar rechts zijn geschoven. Dat aantal plaatsen is precies zo groot dat regel 40 tussen regel 10 en 20 in past.
Nu valt echter op dat de pointer adr1 (die is gewoon blijven staan) niet meer naar het begin van regel 40 wijst, maar ergens midden in regel 30. Door nu de lengte van regel 40 (is de lengte van de gecreëerde ruimte) bij adr1 op te tellen, wijst deze pointer weer precies naar het begin van regel 40. Deze laatste actie is met de omcirkelde 2 aangegeven.
Nu kunnen we zonder meer regel 40 naar de vrijgekomen ruimte kopiëren, hetgeen in afbeelding 9-1 is aangegeven met de omcirkelde 3. Is die regel echter gekopieerd, dan valt onmiddellijk een volgend probleem op. We hebben nu twee regels 40, waarvan er één op de verkeerde plaats staat. Er zal dus niets anders opzitten dan de gekopieerde regel 40 te hernummeren. Dit is aangegeven met de omcirkelde 4. U ziet dat de gekopieerde regel 40 het nieuwe regelnummer 11 heeft gekregen.
We kennen nu de belangrijkste problemen die we bij het kopiëren van BASIC-regels binnen één en hetzelfde programma zullen tegenkomen. Dat er nog wel meer problemen kunnen ontstaan moge blijken uit het volgende voorbeeld. Stel dat u een programma hebt waarin u 23 BASIC-regels naar een andere plaats wilt kopiëren. Laten we zeggen dat u die regels achter regel 350 wilt kopiëren. De eerstvolgende regel na regel 350 is echter regel 360. Het kopiëren is geen probleem. Het hernummeren zal echter wel een probleem worden. Er is immers maar plaats voor 9 regels tussen regel 350 en 360. Maar u wilt er 23 regels tussen zetten. Het gevolg zal zijn dat er een aantal dubbele regelnummers ontstaan. Straks zullen we zien dat het programma alleen gaat kopiëren als het aantal te kopiëren programmaregels niet groter is dan er nieuwe regelnummers kunnen worden gemaakt.



Afb. 9-1 Geheugenindeling bij kopiëren

Zoals bij alle programma's in dit boek is ook in het kopieerprogramma weer gebruik gemaakt van de in de Spectrum-ROM aanwezige routines. Dit heeft tot gevolg dat, ondanks het grote aantal acties dat moet worden ondernomen om één of meer regels te kopiëren, de uitvoeringstijd is teruggebracht tot bijna niets. Alleen het hernummeren kost enige tijd. Daar het bij het gebruik van dit programma bijna altijd om slechts een beperkt aantal regels zal gaan, zal die tijd nooit meer dan enkele seconden zijn.
Zoals uit de voorgaande analyse van het probleem al was duidelijk geworden zullen we op een gegeven moment het startadres van een regel moeten uitzoeken. Hiervoor kunnen we gebruik maken van een ROM-routine die we ook al in één van de vorige programma's hebben gebruikt. Het is de ROM-routine die start op adres 196E (hex.). Deze ROM-routine zoekt het startadres van de regel waarvan het regelnummer in registerpaar HL is gegeven, op en zet dat startadres vervolgens in datzelfde registerpaar HL. Indien het in HL opgegeven regelnummer niet in het BASIC-programma voorkomt, dan zal het startadres van het eerstvolgende hogere regelnummer in HL worden gezet, terwijl het startadres van het eerstvorige lagere regelnummer in registerpaar DE wordt gezet. Met deze routine zullen we in staat zijn om de pointers (adr1, adr2 en adr3) te laden met de juiste startadressen.
Het aantal regels dat kan worden gekopieerd is afhankelijk van het aantal vrije regelnummers tussen de regels waartussen de regels moeten worden gekopieerd. Om uit te vinden hoeveel vrije regelnummers er zijn, moeten we eerst controleren of het regelnummer van

de regel waarachter we willen kopiëren wel bestaat en wat het eerstvolgende regelnummer is. Hiervoor kunnen we gebruik maken van de ROM-routine die op adres 190F start. Ook deze routine is een oude bekende. Deze ROM-routine leest het regelnummer dat in de systeemvariabele E-PPC staat en zoekt vervolgens in het BASIC-programma naar het eerstvolgende hogere regelnummer. Is dat hogere regelnummer gevonden dan wordt dat in de systeemvariabele E-PPC gezet. Willen we controleren of regel 100 voorkomt in het programma, dan zullen we de waarde 99 in E-PPC zetten en na uitvoering van de routine kijken of er in E-PPC regelnummer 100 staat. Willen we weten wat het eerstvolgende regelnummer na 100 is, dan laden we 100 in E-PPC en lezen E-PPC nog eens nadat de ROM-routine is uitgevoerd. Het nummer dat we dan lezen is het eerstvolgende hogere regelnummer.
Het maken van ruimte kan ook met een ROM-routine worden gedaan. Op adres 1655 (hex.) start een routine die vanaf het in registerpaar HL gegeven adres een ruimte in het geheugen maakt met een lengte van het in registerpaar BC gegeven aantal bytes. Hierbij dienen we er wel rekening mee te houden dat HL naar de plaats onmiddellijk volgend op de plaats waar ruimte moet worden gemaakt, wijst. Afbeelding 9-2 illustreert dit.



Afb. 9-2 Registergebruik bij ruimte maken

In de probleem-analyse aan het begin van dit hoofdstuk zagen we al welke pointers we nodig hadden. We kunnen nu de volgende relaties leggen: Pointer adr3 komt overeen met registerpaar HL in de ROM-routine en de lengte (len) komt overeen met registerpaar BC in de ROM-routine.
Voor het kopiëren van de regels heb ik geen gebruik gemaakt van een ROM-routine. De Z80 beschikt over een zeer krachtige instructie die we voor dat kopiëren uitstekend kunnen gebruiken. De instructie LDIR (Load Immediate Repeated) stelt ons namelijk in staat om in één keer elk gewenst aantal bytes van de ene naar een andere plaats te kopiëren. Het maximum aantal bytes dat in één keer kan worden gekopiëerd is 65535. Zoveel zullen we echter nooit nodig hebben. Om LDIR zijn werk te laten doen moeten we een aantal registers van te voren met bepaalde gegevens laden.
In registerpaar HL zetten we het 'source'-adres,
in registerpaar DE zetten we het 'destination'-adres en
in registerpaar BC zetten we het aantal bytes dat van de source naar de destination moet worden gekopieerd.Als we daarna de LDIR-instructie uitvoeren, dan zal de kopie in een fractie van een seconde worden gemaakt.

adres	mnem	operands	hex. code	commentaar
60000 60003 60006	LD CALL RET	HL,5C49 190F	21 49 5C CD 0F 19 C9	adres van EPPC --> HL zoek volgend regelnummer
60007 60010 60013 60014 60015	LD CALL LD LD RET	HL,0000 196E B,H C,L	21 00 00 CD 6E 19 44 4D C9	regelnummer --> HL zoek RAM-adres van het in HL gegeven regelnummer } HL --> BC
60016 60019 60022 60025	LD LD CALL RET	BC,0000 HL,0000 1655	01 00 00 21 00 00 CD 55 16 C9	lengte --> BC startadres --> HL maak ruimte in RAM
60026 60029 60032 60035 60037	LD LD LD LDIR RET	HL,0000 DE,0000 BC,0000	21 00 00 11 00 00 01 00 00 ED B0 C9	source adres --> HL destination adres --> DE aantal bytes --> BC copieer source --> dest.

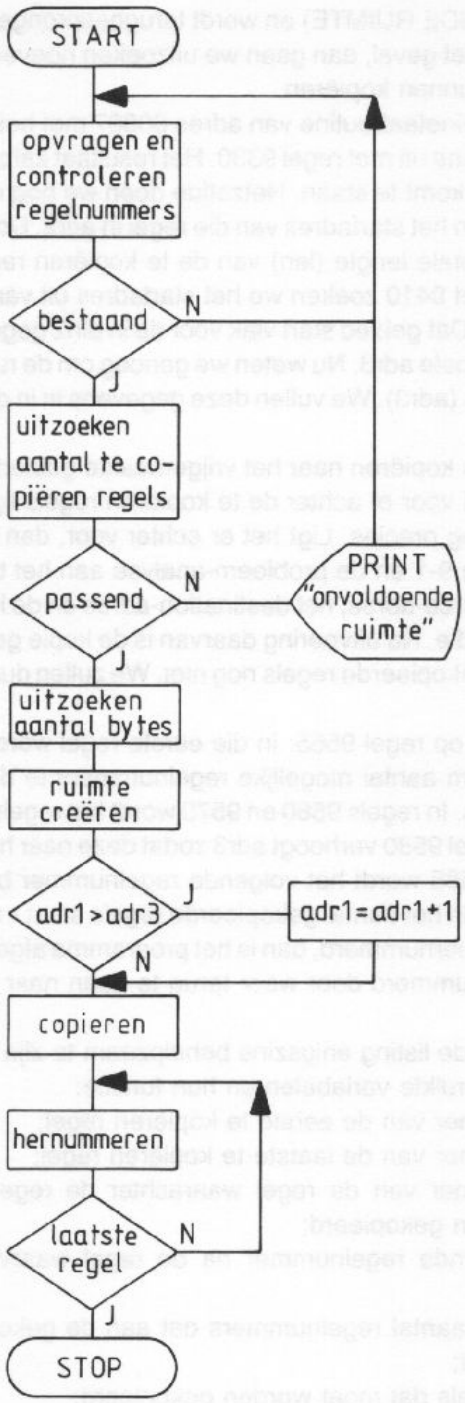
Nu we weten welke ROM-routines we nodig hebben en wat die ROM-routines zelf aan input nodig hebben zullen we eens zien hoe we ze met behulp van korte machinetaalroutines kunnen starten.
Op adres 60000 start een machinetaalroutine waarmee de ROM-routine die op adres 190F begint kan worden gestart. Met de LD-instructie laden we het adres van E-PPC in registerpaar HL. Vervolgens starten we de ROM-routine met de CALL-instructie en keren we terug naar het BASIC-programma met de RET-instructie. We weten nu dat het eerstvolgende regelnummer in systeemvariabele E-PPC staat.
Op adres 60007 start de machinetaalroutine waarmee we de ROM-routine die op adres 196E begint starten. Met de eerste instructie laden we het regelnummer waarvan we het startadres in het geheugen willen weten in registerpaar HL. Daar we echter pas weten van welk regelnummer het startadres moet worden opgezocht als we met het programma bezig zijn, heb ik hier de waarde 0 ingevuld. Vanuit het BASIC-programma zullen we het werkelijke regelnummer in deze machinetaalinstructie moeten POKEn. Met de instructies op adressen 60013 en 60014 wordt de inhoud van HL naar registerpaar BC gekopieerd. Na uitvoering van de ROM-routine bevat HL het gezochte RAM-adres. Dat adres zetten we nu dus in BC en zoals bekend, wordt de inhoud van BC doorgegeven aan het BASIC-programma via de functie USR. Door in het BASIC-programma LET adr=USR 60007 te programmeren zal na terugkeer uit de machinetaalroutine 'adr' de waarde van het BC registerpaar krijgen, ofwel het gezochte adres.
Op adres 60016 start de machinetaalroutine die de ROM-routine aanroept waarmee ruimte in het geheugen wordt gemaakt. Met de eerste instructie zetten we de lengte in registerpaar BC, met de daaropvolgende instructie het adres waarvoor de ruimte moet worden gemaakt in HL. Ook hier geldt weer dat het adres noch de lengte bekend zijn. Ze zijn

afhankelijk van de door de 'operator' op te geven regelnummers. We zullen deze instructies dus weer vanuit het BASIC-programma moeten completeren met behulp van POKE-instructies.
De routine die op adres 60026 start roept geen ROM-routine aan. Met dit machinetaalroutinetje kan ieder gewenst aantal bytes worden gekopieerd. Daar we niet weten hoeveel bytes, waar vandaan en waar naartoe moeten worden gekopieerd is ook hier weer de waarde 0 achter de eerste drie instructies gezet. Vanuit BASIC moeten hier weer de juiste waarden in gePOKEd worden.
Nu zijn we voldoende gewapend om het BASIC-programma aan een wat nauwkeuriger onderzoek te onderwerpen. Bij dit onderzoek zullen we de flowchart van afbeelding 9-3 en de listing van het BASIC-programma gebruiken.
Het programma begint met het opvragen van de te kopiëren regelnummers. In regel 9030 wordt naar het eerst te kopiëren regelnummer gevraagd. In regel 9040 controleren we of het opgegeven regelnummer wel 1 of hoger, doch niet hoger dan 9000 is. Vanaf regelnummer 9000 start namelijk ons kopieerprogramma en daarin willen we niets kopiëren. Vervolgens springen we naar de subroutine op regel 9800.
In die subroutine zetten we het opgegeven regelnummer in de systeemvariabele E-PPC en voeren we de machinetaalroutine die op adres 60000 start uit. We kijken nu of het opgegeven regelnummer in het programma voorkomt en zetten de variabele 'ok' op 1 als dat zo is. Hierna keren we terug naar regel 9060. Bestaat het gevraagde regelnummer niet dan springen we terug naar regel 9030, waar naar een nieuw regelnummer wordt gevraagd. Bestaat het gevraagde regelnummer wel, dan wordt dat nummer in de variabele sln1 opgeslagen.
Nu wordt het laatste te kopiëren regelnummer gevraagd. In regel 9100 wordt gekeken of het ingegeven regelnummer wel hoger is dan het eerste regelnummer, doch niet hoger dan regelnummer 9000. Is dat het geval dan wordt weer naar de subroutine op regel 9800 gesprongen, waar weer wordt gekeken of het opgegeven regelnummer wel bestaat. Bestaat het niet dan kunt u weer een nieuw (wel bestaand) regelnummer ingeven, bestaat het wel dan wordt dat regelnummer opgeslagen in variabele sln2.
Het programma weet nu welke regels moeten worden gekopieerd, doch het weet nog niet waar die regels naartoe moeten. Dat wordt in regel 9160 gevraagd. Indien de plaats van bestemming niet tussen regelnummer 1 en 8999 ligt, dan kunt u een nieuw regelnummer opgeven. Ligt het nummer daar wel tussen, dan wordt weer met de subroutine op regel 9800 uitgezocht of dat regelnummer bestaat. Ook nu krijgt u weer een herkansing als dat regelnummer niet mocht bestaan.
Vanaf regel 9200 gaan we uitzoeken wat het volgende regelnummer is na de regel waarachter we een kopie willen tussenvoegen. We hebben het regelnummer waarachter moet worden tussengevoegd inmiddels in variabele dln1 opgeslagen. Het daaropvolgende regelnummer slaan we op in variabele dln 2. In regel 9230 berekenen we het aantal regelnummers dat tussen de in dln1 en dln2 opgeslagen regelnummers past. Dit aantal wordt in variabele 'pos' opgeslagen.
Nu moeten we uitzoeken hoeveel regels er moeten worden gekopieerd, om te zien of er wel voldoende regelnummers ter beschikking staan. Vanaf regel 9255 wordt het aantal regels bepaald door in een lus het volgende regelnummer op te zoeken, beginnende bij het in sln1 bewaarde regelnummer, eindigende wanneer het regelnummer dat in sln2 was bewaard is bereikt. Het totaal aantal te kopiëren regels staat dan in de variabele nrl. In regel 9295 kijken we of het aantal te kopiëren regels wel kleiner of gelijk is aan het maximum aantal toe te kennen regelnummers. Is dat niet het geval dan wordt daarvan melding

gemaakt (ONVOLDOENDE RUIMTE) en wordt teruggesprongen naar het begin van het programma. Is dat wel het geval, dan gaan we uitzoeken hoeveel ruimte we moeten maken om alle regels te kunnen kopiëren.
We prepareren de machinetaalroutine van adres 60007 met het gewenste regelnummer (sln1) en voeren die routine uit met regel 9330. Het resultaat zal zijn dat het startadres van de regel uit sln1 in adr1 komt te staan. Hetzelfde doen we nog eens voor de regel die in sln2 staat en we bewaren het startadres van die regel in adr2. Door nu adr1 van adr2 af te trekken vinden we de totale lengte (len) van de te kopiëren regels.
In regels 9390 tot en met 9410 zoeken we het startadres uit van het gebied waar we de ruimte moeten creëren. Dat gebied start vlak voor de in dln2 gegeven regel. We slaan dat startadres op in de variabele adr3. Nu weten we genoeg om de ruimte te creëren: de lengte (len) en het startadres (adr3). We vullen deze gegevens in in de machinetaalroutine en starten die.
Nu kunnen we de regels kopiëren naar het vrijgemaakte gebied. Heel belangrijk is nu of het vrijgemaakte gebied voor of achter de te kopiëren regels ligt. Ligt het erachter, dan kloppen alle pointers nog precies. Ligt het er echter voor, dan moeten we pointer adr1 bijstellen (zie afbeelding 9-1 en de probleem-analyse aan het begin van dit hoofdstuk). Hierna vullen we het source-adres, het destination-adres en de lengte in de machinetaalroutine in en starten we die. Na uitvoering daarvan is de kopie gemaakt, doch kloppen de regelnummers van de gekopieerde regels nog niet. We zullen dus nog een hernummering moeten starten.
Het hernummeren start op regel 9555. In die eerste regel wordt de verhogingsstap bepaald door het maximum aantal mogelijke regelnummers te delen door het werkelijke aantal te kopiëren regels. In regels 9560 en 9570 wordt het regelnummer in de gekopieerde regel gePOKEd. Regel 9580 verhoogt adr3 zodat deze naar het begin van de volgende regel wijst. Met regel 9585 wordt het volgende regelnummer bepaald en in regel 9590 wordt de variabele waarin het aantal gekopieerde regels staat met 1 verlaagd. Indien alle gekopieerde regels zijn hernummerd, dan is het programma afgelopen. Zo niet, dan wordt de volgende regel hernummerd door weer terug te gaan naar regel 9560.

Om u bij het lezen van de listing enigszins behulpzaam te zijn volgt hier nog een korte opsomming van de gebruikte variabelen en hun functie:

- sln1 – het regelnummer van de eerste te kopiëren regel;
- sln2 – het regelnummer van de laatste te kopiëren regel;
- dln1 – het regelnummer van de regel waarachter de regels sln1 tot en met sln2 moeten worden gekopieerd;
- dln2 – het eerstvolgende regelnummer na de regel waarvan het regelnummer in dln1 staat;
- pos – het maximum aantal regelnummers dat aan de gekopieerde regels kan worden toegekend;
- nrl – het aantal regels dat moet worden gekopieerd;
- adr1 – het startadres van de eerste te kopiëren regel;
- adr2 – het startadres van de regel na de laatste te kopiëren regel;
- adr3 – het startadres van het gebied waar de regels naartoe moeten worden gekopieerd;
- len – de totale lengte van alle te kopiëren regels (adr2-adr1);
- incr – de verhogingsstap voor het hernummeren (pos/nrl).



Afb. 9-3 Flow chart van het programma 'copylines'

Het BASIC-programma:

```
9001 REM ***********************
9002 REM * Copieren van BASIC- *
9003 REM * regels.             *
9004 REM ***********************
9005 REM * Het opvragen van de *
9006 REM * te copieren regels. *
9007 REM ***********************
9010 CLS
9020 PRINT "TE COPIEREN BASIC-RE
GELS:"''
9030 INPUT "Eerste regelnummer?
";ln
9040 IF ln>=9000 OR ln=0 THEN  G
O TO 9030
9050 GO SUB 9800
9060 IF ok=0 THEN  GO TO 9030
9070 LET sln1=ln
9080 PRINT "Copie van de regels
";sln1
9090 INPUT "Laatste regelnummer?
 ";ln
9100 IF ln<sln1 OR ln>=9000 OR l
n=0 THEN  GO TO 9090
9110 GO SUB 9800
9120 IF ok=0 THEN  GO TO 9090
9130 LET sln2=ln
9140 PRINT "tot en met
";sln2''
9150 PRINT "PLAATS VAN BESTEMMIN
G:"''
9160 INPUT "Achter welke regel?"
;ln
9170 IF ln=0 OR ln>=9000 THEN  G
O TO 9160
9180 GO SUB 9800
9185 IF ok=0 THEN  GO TO 9160
9190 PRINT "Tussenvoegen na rege
l ";ln
9200 LET ln=ln+1: LET dln1=ln
9210 GO SUB 9800
9220 LET dln2=(PEEK 23625+256*PE
EK 23626)-1
9230 LET pos=dln2-dln1+1
9240 PRINT "max. aantal regels
  ";pos
9250 REM ***********************
9251 REM * uitzoeken aantal te *
9252 REM * copieren regels     *
9253 REM ***********************
9255 LET ln=sln1: LET nr1=1
9260 POKE 23626,INT (ln/256)
9265 POKE 23625,ln-256*(INT (ln/
256))
```

```
9270 RANDOMIZE USR 60000
9275 LET ln=PEEK 23625+256*PEEK
23626
9280 IF sln2<ln THEN  GO TO 9295
9285 LET nrl=nrl+1
9290 GO TO 9260
9295 IF nrl<=pos THEN  PRINT "We
rk. aantal regels   ";nrl
9300 IF nrl>pos THEN  PRINT  FLA
SH 1;"ONVOLDOENDE RUIMTE": PAUSE
 100: GO TO 9010
9310>REM ************************
9311 REM *  uitzoeken hoeveel  *
9312 REM * ruimte er nodig is. *
9313 REM ************************
9315 POKE 60009,INT (sln1/256)
9320 POKE 60008,sln1-256*(INT (s
ln1/256))
9330 LET adr1=USR 60007
9335 LET sln2=sln2+1
9340 POKE 60009,INT (sln2/256)
9350 POKE 60008,sln2-256*(INT (s
ln2/256))
9360 LET adr2=USR 60007
9370 LET len=adr2-adr1
9380 REM ************************
9381 REM * ruimte maken (len)  *
9382 REM * tussen de regels    *
9384 REM * dln1 en dln2        *
9385 REM ************************
9390 POKE 60009,INT (dln2/256)
9400 POKE 60008,dln2-256*(INT (d
ln2/256))
9410 LET adr3=USR 60007
9420 POKE 60018,INT (len/256)
9430 POKE 60017,len-256*(INT (le
n/256))
9440 POKE 60021,INT (adr3/256)
9450 POKE 60020,adr3-256*(INT (a
dr3/256))
9460 RANDOMIZE USR 60016
9470 REM ************************
9471 REM *   copieren regels   *
9472 REM ************************
9475 IF adr1>adr3 THEN  LET adr1
=adr1+len
9480 POKE 60028,INT (adr1/256)
9490 POKE 60027,adr1-256*(INT (a
dr1/256))
9500 POKE 60031,INT (adr3/256)
9510 POKE 60030,adr3-256*(INT (a
dr3/256))
9520 POKE 60034,INT (len/256)
9530 POKE 60033,len-256*(INT (le
n/256))
```

```
9540 RANDOMIZE USR 60026
9550 REM ***********************
9551 REM *  hernummeren van de *
9552 REM *  gecopieerde regels *
9553 REM ***********************
9555 LET incr=INT (pos/nrl)
9560 POKE adr3,INT (dln1/256)
9570 POKE (adr3+1),dln1-256*(INT
 (dln1/256))
9580 LET adr3=adr3+(PEEK (adr3+2
)+256*PEEK (adr3+3))+4
9585 LET dln1=dln1+incr
9590 LET nrl=nrl-1
9600 IF nrl<=0 THEN  STOP
9620 GO TO 9560
9800>REM ***********************
9801 REM * controleren regelnr.*
9802 REM ***********************
9810 LET ln=ln-1
9820 POKE 23626,INT (ln/256)
9830 POKE 23625,ln-256*(INT (ln/
256))
9840 RANDOMIZE USR 60000
9845 LET ln=ln+1
9850 IF ln<>(PEEK 23625+256*PEEK
 23626) THEN  LET ok=0
9860 IF ln=(PEEK 23625+256*PEEK
23626) THEN  LET ok=1
9870 RETURN
9999 CLEAR 59990: LOAD "copycode
"CODE 60000,100: STOP
```

Voorbeelden:

Het eerste voorbeeld laat zien wat er gebeurt wanneer er een aantal regels wordt gekopieerd naar een plaats waarbij de verhogingsstap 1 is.

```
  1 REM ***********************
  2 REM * dit is een voorbeeld*
  3 REM *waarmee het resultaat*
  4 REM * van "copylines" kan *
  5 REM *worden gedemonstreerd*
  6 REM ***********************
 10 REM
 20 REM
 40 REM
 80 REM
100 REM
```

```
TE COPIEREN BASIC-REGELS:

Copie van de regels 1
tot en met          6

PLAATS VAN BESTEMMING:

Tussenvoegen na regel 10
max. aantal regels    9
Werk. aantal regels   6
```

```
  1 REM ************************
  2 REM * dit is een voorbeeld*
  3 REM *waarmee het resultaat*
  4 REM * van "copylines" kan *
  5 REM *worden gedemonstreerd*
  6 REM ************************
 10 REM
 11 REM ************************
 12 REM * dit is een voorbeeld*
 13 REM *waarmee het resultaat*
 14 REM * van "copylines" kan *
 15 REM *worden gedemonstreerd*
 16 REM ************************
 20 REM
 40 REM
 80 REM
100 REM
```

Het tweede voorbeeld laat zien wat er gebeurt indien dezelfde regels naar een andere plaats worden gekopieerd, waarbij de door het programma gevonden verhogingsstap groter dan 1 is. Past u bij het kopiëren ervoor op dat indien u iets achter de laatste regel van uw programma kopieert, de verhogingsstap erg groot kan zijn. Om een te grote verhogingsstap in dat geval te voorkomen kunt u voordat u gaat kopiëren even een REM-statement aan het einde van uw programma programmeren. Door de keuze van het regelnummer van die REM-statement beïnvloedt u de verhogingsstap.

```
  1 REM ************************
  2 REM * dit is een voorbeeld*
  3 REM *waarmee het resultaat*
  4 REM * van "copylines" kan *
  5 REM *worden gedemonstreerd*
  6 REM ************************
 10 REM
 20 REM
 40 REM
 80 REM
100 REM
```

```
TE COPIEREN BASIC-REGELS:

Copie van de regels 1
tot en met          6

PLAATS VAN BESTEMMING:

Tussenvoegen na regel 40
max. aantal regels    39
Werk. aantal regels   6

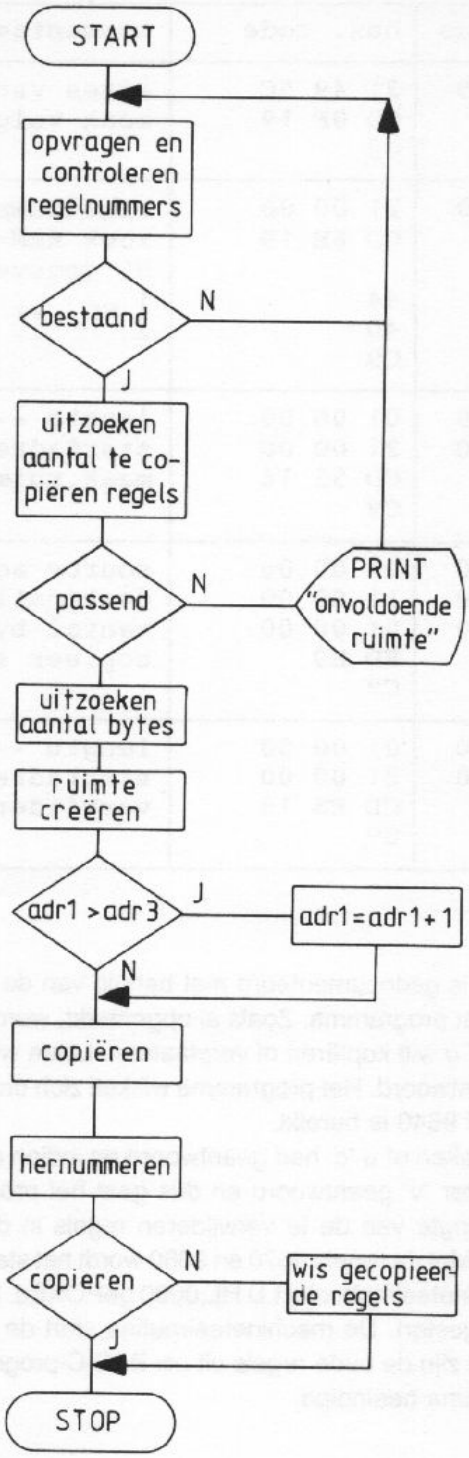
   1 REM ************************
   2 REM * dit is een voorbeeld*
   3 REM *waarmee het resultaat*
   4 REM * van "copylines" kan *
   5 REM *worden gedemonstreerd*
   6 REM ************************
  10 REM
  20 REM
  40 REM
  41 REM ************************
  47 REM * dit is een voorbeeld*
  53 REM *waarmee het resultaat*
  59 REM * van "copylines" kan *
  65 REM *worden gedemonstreerd*
  71 REM ************************
  80 REM
 100 REM
```

10. Het verplaatsen van BASIC-regels

Zoals u bekend zal zijn kan men een programma versnellen door de in dat programma voorkomende subroutines zoveel mogelijk aan het begin van het programma te plaatsen. Vooral de meest gebruikte subroutines moeten het verst naar het begin van een programma worden geplaatst. Tijdens het schrijven van een programma kan het echter voorkomen dat je nog een subroutine bedenkt terwijl je al een heel eind klaar bent met dat programma. Indien er dan aan het begin van het programma voldoende regelnummers zijn overgeslagen, dan kan een subroutine alsnog direct aan het begin worden geplaatst. Is dat echter niet het geval, dan kan het plezierig zijn indien men op eenvoudige wijze kan schuiven met grotere blokken BASIC-regels.
Met dit idee in het achterhoofd heb ik het hierna volgende programma ontwikkeld. Daar echter voor het verplaatsen van BASIC-regels die regels eerst moeten worden gekopieerd, is dit programma eigenlijk alleen een uitbreiding van het programma uit het vorige hoofdstuk geworden. En omdat het hele programma uit het vorige hoofdstuk in dit 'verplaats'-programma is verwerkt, heb ik dan ook aan het begin van dit programma de keuzemogelijkheid opgenomen om te kopiëren of te verplaatsen. Het antwoord op de vraag uit regel 9010: 'kopiëren of verplaatsen', wordt in variabele a$ gezet. Daarna is het programma gelijk aan het programma uit het vorige hoofdstuk, tot regel 9630. Vanaf die regel volgt een uitbreiding die speciaal voor het verplaatsen dient.
Omdat het kopiëren al zeer uitgebreid is behandeld in het vorige hoofdstuk, zal ik daar nu niet meer op ingaan. Waar we wel even naar moeten kijken is wat er voor het verplaatsen nu nog extra moet worden gedaan.
Stelt u zich voor dat u een aantal regels wilt verplaatsen. Dan zult u die regels eerst naar de plaats waar u ze wilt hebben toe moeten kopiëren. Zijn die regels gekopieerd, dan zullen de oude regels nog moeten worden gewist. Wissen van regels zonder meer zou een groot gat tussen de andere regels van het programma achterlaten. We zullen dus de na dat gat volgende regels moeten laten aansluiten op de voorgaande regels. Bovendien moet daarna een aantal systeemvariabelen op de nieuw ontstane lengte van het BASIC-programma worden ingesteld. U begrijpt dat dit een erg omvangrijke klus zou zijn.
Er is echter een ROM-routine die al dit werk, het wissen en het aansluiten, voor ons doet. Deze ROM-routine hebben we al eens gebruikt in het programma 'blokdel'. De routine start op adres 19E8 en verwacht het startadres van de te verwijderen bytes in registerpaar HL en het aantal te verwijderen bytes in registerpaar BC.
Het aanroepen van deze ROM-routine moeten we dus toevoegen aan de machinetaalroutines die we al voor het kopiëren hadden gemaakt. Deze toegevoegde machinetaalroutine start in de machinetaal-listing op adres 60038. U ziet dat de registerparen HL en BC niet met werkelijke waarden worden geladen. Deze waarden zijn immers steeds anders, afhankelijk van de regels die we willen verplaatsen. We zullen deze regels dan ook vanuit het BASIC-programma in de machinetaalinstructies POKEn.

adres	mnem	operands	hex. code	commentaar
60000 60003 60006	LD CALL RET	HL,5C49 190F	21 49 5C CD 0F 19 C9	adres van EPPC --> HL zoek volgend regelnummer
60007 60010 60013 60014 60015	LD CALL LD LD RET	HL,0000 196E B,H C,L	21 00 00 CD 6E 19 44 4D C9	regelnummer --> HL zoek RAM-adres van het in HL gegeven regelnummer } HL --> BC
60016 60019 60022 60025	LD LD CALL RET	BC,0000 HL,0000 1655	01 00 00 21 00 00 CD 55 16 C9	lengte --> BC startadres --> HL maak ruimte in RAM
60026 60029 60032 60035 60037	LD LD LD LDIR RET	HL,0000 DE,0000 BC,0000	21 00 00 11 00 00 01 00 00 ED B0 C9	source adres --> HL destination adres --> DE aantal bytes --> BC copieer source --> dest.
60038 60041 60044 60047	LD LD CALL RET	BC,0000 HL,0000 19E8	01 00 00 21 00 00 CD E8 19 C9	lengte --> BC startadres --> HL verwijder bytes uit RAM

Het BASIC-programma is gedocumenteerd met behulp van de flowchart uit afbeelding 10-1 en de listing van dat programma. Zoals al opgemerkt, wordt aan het begin van het programma gevraagd of u wilt kopiëren of verplaatsen. Laten we even aannemen dat u daarop met 'v' hebt geantwoord. Het programma wikkelt zich dan af zoals dat ook bij kopiëren gaat, totdat regel 9640 is bereikt.
In regel 9640 wordt gekeken of u 'c' had geantwoord en indien dat zo was dan stopt het programma. U had echter 'v' geantwoord en dus gaat het programma door met regel 9650. Daar wordt de lengte van de te verwijderen regels in de machinetaalinstructie LD BC,0000 gePOKEd. Met de regels 9670 en 9680 wordt het startadres van de te verwijderen regels in de machinetaalinstructie LD HL,0000 gePOKEd. En met regel 9690 wordt de machinetaalroutine gestart. De machinetaalroutine start de ROM-routine en in een fractie van een seconde zijn de oude regels uit uw BASIC-programma verwijderd. Hiermee is dan het programma beëindigd.



Afb. 10-1 Flow chart van het programma 'copymove'

```
9001 REM ***********************
9002 REM *Copieren of verplaat-*
9003 REM *sen van BASIC-regels *
9004 REM ***********************
9005 REM * Het opvragen van de *
9006 REM * te copieren regels. *
9007 REM ***********************
9010 INPUT "copieren of verplaat
sen? (c/v)";a$
9015 IF a$="c" OR a$="v" THEN  G
O TO 9020
9017 GO TO 9010
9020 CLS : PRINT "TE COPIEREN BA
SIC-REGELS:"''
9030 INPUT "Eerste regelnummer?
";ln
9040 IF ln>=9000 OR ln=0 THEN  G
O TO 9030
9050 GO SUB 9800
9060 IF ok=0 THEN  GO TO 9030
9070 LET sln1=ln
9080 PRINT "Copie van de regels
";sln1
9090 INPUT "Laatste regelnummer?
 ";ln
9100 IF ln<sln1 OR ln>=9000 OR l
n=0 THEN  GO TO 9090
9110 GO SUB 9800
9120 IF ok=0 THEN  GO TO 9090
9130 LET sln2=ln
9140 PRINT "tot en met
";sln2''
9150 PRINT "PLAATS VAN BESTEMMIN
G:"''
9160 INPUT "Achter welke regel?"
;ln
9170 IF ln=0 OR ln>=9000 THEN  G
O TO 9160
9180 GO SUB 9800
9185 IF ok=0 THEN  GO TO 9160
9190 PRINT "Tussenvoegen na rege
l ";ln
9200 LET ln=ln+1: LET dln1=ln
9210 GO SUB 9800
9220 LET dln2=(PEEK 23625+256*PE
EK 23626)-1
9230 LET pos=dln2-dln1+1
9240 PRINT "max. aantal regels
  ";pos
9250 REM ***********************
9251 REM * uitzoeken aantal te *
9252 REM * copieren regels     *
9253 REM ***********************
9255 LET ln=sln1: LET nrl=1
9260 POKE 23626,INT (ln/256)
```

```
9265 POKE 23625,ln-256*(INT (ln/
256))
9270 RANDOMIZE USR 60000
9275 LET ln=PEEK 23625+256*PEEK
23626
9280 IF sln2<ln THEN  GO TO 9295
9285 LET nrl=nrl+1
9290 GO TO 9260
9295 IF nrl<=pos THEN  PRINT "We
rk. aantal regels   ";nrl
9300>IF nrl>posTHEN PRINT FLASH
1;"ONVOLDOENDE RUIMTE":PAUSE 100
:GO TO 9010
9310 REM **********************
9311 REM *  uitzoeken hoeveel  *
9312 REM * ruimte er nodig is. *
9313 REM **********************
9315 POKE 60009,INT (sln1/256)
9320 POKE 60008,sln1-256*(INT (s
ln1/256))
9330 LET adr1=USR 60007
9335 LET sln2=sln2+1
9340 POKE 60009,INT (sln2/256)
9350 POKE 60008,sln2-256*(INT (s
ln2/256))
9360 LET adr2=USR 60007
9370 LET len=adr2-adr1
9380 REM **********************
9381 REM * ruimte maken (len)  *
9382 REM * tussen de regels    *
9384 REM * dln1 en dln2        *
9385 REM **********************
9390 POKE 60009,INT (dln2/256)
9400 POKE 60008,dln2-256*(INT (d
ln2/256))
9410 LET adr3=USR 60007
9420 POKE 60018,INT (len/256)
9430 POKE 60017,len-256*(INT (le
n/256))
9440 POKE 60021,INT (adr3/256)
9450 POKE 60020,adr3-256*(INT (a
dr3/256))
9460 RANDOMIZE USR 60016
9470 REM **********************
9471 REM *   copieren regels   *
9472 REM **********************
9475 IF adr1>adr3 THEN  LET adr1
=adr1+len
9480 POKE 60028,INT (adr1/256)
9490 POKE 60027,adr1-256*(INT (a
dr1/256))
9500 POKE 60031,INT (adr3/256)
9510 POKE 60030,adr3-256*(INT (a
dr3/256))
9520 POKE 60034,INT (len/256)
```

```
9530 POKE 60033,len-256*(INT (le
n/256))
9540 RANDOMIZE USR 60026
9550 REM ************************
9551 REM *  hernummeren van de *
9552 REM *  gecopieerde regels *
9553 REM ************************
9555 LET incr=INT (pos/nrl)
9560 POKE adr3,INT (dln1/256)
9570 POKE (adr3+1),dln1-256*(INT
 (dln1/256))
9580 LET adr3=adr3+(PEEK (adr3+2
)+256*PEEK (adr3+3))+4
9585 LET dln1=dln1+incr
9590 LET nrl=nrl-1
9600 IF nrl<=0 THEN  GO TO 9630
9620 GO TO 9560
9630>REM ************************
9631 REM * verwijderen orginele*
9632 REM *regels(sln1 t/m sln2)*
9633 REM ************************
9640 IF a$="c" THEN  STOP
9650 POKE 60040,INT (len/256)
9660 POKE 60039,len-256*INT (len
/256)
9670 POKE 60043,INT (adr1/256)
9680 POKE 60042,adr1-256*INT (ad
r1/256)
9690 RANDOMIZE USR 60038
9700 STOP
9800 REM ************************
9801 REM * controleren regelnr.*
9802 REM ************************
9810 LET ln=ln-1
9820 POKE 23626,INT (ln/256)
9830 POKE 23625,ln-256*(INT (ln/
256))
9840 RANDOMIZE USR 60000
9845 LET ln=ln+1
9850 IF ln<>(PEEK 23625+256*PEEK
 23626) THEN  LET ok=0
9860 IF ln=(PEEK 23625+256*PEEK
23626) THEN  LET ok=1
9870 RETURN
9999 CLEAR 59990: LOAD "copmovco
de"CODE 60000,100: STOP
```

Voorbeelden:
Het eerste voorbeeld laat zien wat het resultaat is van het verplaatsen van een aantal regels naar een plaats achter een hoger regelnummer.

```
  1 REM **********************
  2 REM *  testregels voor    *
  3 REM * programma "copymove"*
  4 REM **********************
 10 REM
 20 REM
 30 REM
 40 REM
 50 REM
 60 REM
 70 REM
 80 REM
 90 REM
100 REM


Na het kiezen van de functie: verplaatsen


TE COPIEREN BASIC-REGELS:

Copie van de regels 1
tot en met          4

PLAATS VAN BESTEMMING:

Tussenvoegen na regel 50
max. aantal regels    9
Werk. aantal regels   4


 10 REM
 20 REM
 30 REM
 40 REM
 50 REM
 51 REM **********************
 53 REM *  testregels voor    *
 55 REM * programma "copymove"*
 57 REM **********************
 60 REM
 70 REM
 80 REM
 90 REM
100 REM
```

Het tweede voorbeeld laat het resultaat zien van het verplaatsen naar een lager regelnummer. Wellicht herinnert u zich nog dat regel 9475 van het BASIC-programma speciaal voor dit soort verplaatsing nodig was (adr1 moest worden bijgesteld).

```
 10 REM
 20 REM
 30 REM
 40 REM
 50 REM
 51 REM ************************
 52 REM * testen van verplaat-*
 53 REM * sen naar een lager  *
 54 REM * regelnummer         *
 55 REM ************************
 60 REM
 70 REM
 80 REM
 90 REM
100 REM


TE COPIEREN BASIC-REGELS:

Copie van de regels 51
tot en met          55

PLAATS VAN BESTEMMING:

Tussenvoegen na regel 10
max. aantal regels    9
Werk. aantal regels   5



 10 REM
 11 REM ************************
 12 REM * testen van verplaat-*
 13 REM * sen naar een lager  *
 14 REM * regelnummer         *
 15 REM ************************
 20 REM
 30 REM
 40 REM
 50 REM
 60 REM
 70 REM
 80 REM
 90 REM
100 REM
```

11. Opsomming van de gebruikte ROM-routines

Mocht u nu van plan zijn zelf een aantal programma's te schrijven, waarbij u de eerder in dit boek gebruikte ROM-routines wilt toepassen, dan zal de in dit hoofdstuk gegeven opsomming van die routines u wellicht goed van pas komen. Voor iedere routine zullen de volgende gegevens worden vermeld:

Naam : Een korte aanduiding van de functie van de routine.
Startadres : Het hexadecimale startadres van de ROM-routine. Dit adres gebruikt men om de routine met de Z80-instructie CALL aan te roepen.
Registers : Een opsomming van alle Z80-registers die u voor of na het uitvoeren van de ROM-routine moet laden of uitlezen.
Omschrijving: Een omschrijving van de functie van de ROM-routine.

Naam : 'LOAD' van cassette.
Startadres : 0556
Registers : Carry flipflop, A, DE, IX.
Omschrijving: Leest de carry flipflop om te zien of er een LOAD- danwel VERIFY-opdracht moet worden uitgevoerd.
(Indien er sprake is van LOAD, dan is de carry flipflop gezet.) Kijkt in register A of de header danwel het datablok moet worden gelezen. (Indien de header moet worden gelezen, dan staat in A de waarde '00'.) Laadt de data die van cassette wordt gelezen in het RAM vanaf het in register IX aangegeven adres. Het aantal bytes dat wordt gelezen is aangegeven in registerpaar DE.

Naam : Open een channel.
Startadres : 1601
Registers : A.
Omschrijving: Leest het stream-nummer uit register A. Indien daar een geldig stream-nummer in staat, dan wordt aan de hand van de bij dat stream-nummer behorende stream-data het desbetreffende channel geopend.

Naam : Ruimte maken.
Startadres : 1655
Registers : BC, HL.
Omschrijving: Maakt ruimte in het BASIC-geheugendeel (schuift de BASIC-regels uit elkaar) op de plaats direct voor het met registerpaar HL aangegeven adres. Het aantal bytes dat deze ruimte groot zal zijn wordt aangegeven in registerpaar BC.

Naam : PRINT een BASIC-regel.
Startadres : 1865
Registers : D, HL.
Omschrijving: PRINT de BASIC-regel waarvan het startadres in registerpaar HL staat. Deze routine decodeert ook instructie-codes naar strings van ASCII-karakters.
Om geen cursor-indicator afgedrukt te krijgen dient register D met de ASCII-code voor een spatie te worden geladen (hex. '20', dec. 32).

Naam : Zoek volgende regelnummer.
Startadres : 190F
Registers : HL.
Omschrijving: Registerpaar HL dient te zijn geladen met een adres dat naar systeemvariabele E-PPC of S-TOP wijst. De systeemvariabele moet een regelnummer bevatten. De ROM-routine leest de systeemvariabele en zoekt in het BASIC-geheugen naar de eerstvolgende BASIC-regel. Het nummer van die eerstvolgende regel wordt vervolgens in de eerder uitgelezen systeemvariabele gezet.

Naam : Zoek RAM-adres van regel.
Startadres : 196E
Registers : DE, HL.
Omschrijving: Bij het aanroepen van deze routine moet registerpaar HL met een regelnummer geladen zijn. De ROM-routine zoekt dan het RAM-adres van die regel op. Indien de aangegeven regel bestaat, dan zal het bijbehorende RAM-adres in HL worden geladen en het RAM-adres van de daarvoor liggende regel in registerpaar DE. Bestaat de aangegeven regel niet, dan zal in HL het adres van de eerstvolgende wel bestaande regel worden geladen.
De 'zero-flag' wordt gezet indien het opgegeven regelnummer bestaat.

Naam : Ruimte verwijderen (1).
Startadres : 19E5
Registers : DE, HL.
Omschrijving: Indien er in een BASIC-programma ruimte is ontstaan tussen twee op elkaar volgende BASIC-regels, dan kan met deze ROM-routine die ruimte worden verwijderd. Hiertoe moet registerpaar DE het startadres van die ruimte bevatten, terwijl registerpaar HL het adres van het eerste byte na die ruimte bevat. In feite doet deze routine dus niets anders dan het opschuiven van de BASIC-regels, zodanig dat de regels weer direct op elkaar volgen.

Naam : Ruimte verwijderen (2).
Startadres : 19E8
Registers : BC, HL.
Omschrijving: Deze routine doet precies hetzelfde als de vorige, alleen moet nu registerpaar HL het eerste te verwijderen byte aangeven, terwijl het aantal te verwijderen bytes in registerpaar BC staat.

Appendix A

Literatuurlijst

Indien u door de programma's in dit boek ertoe bent aangezet zelf verder te gaan met het maken van dit soort programma's, hetgeen naar ik hoop het geval is, dan zult u behoefte hebben aan meer lectuur op dit gebied. Hoewel er vele tientallen boeken over machinetaal en ROM-routines voor de Spectrum zijn verschenen, zou het opsommen van al die boeken te ver gaan. Daarom zal ik mij beperken tot het noemen van de boeken die ik zelf het meeste heb gebruikt, in de hoop dat u daarmee in ieder geval op een spoor bent gezet.

'The complete Spectrum ROM disassembly', geschreven door dr. Ian Logan en uitgegeven door Melbourne House.

'Spectrum machine language for the absolute beginner', geschreven door William Tang en uitgegeven door Melbourne House. Van dit boek is inmiddels een Nederlandse vertaling verschenen onder de titel 'Machinetaal voor de ZX Spectrum'. Dit wordt uitgegeven door Kluwer Technische Boeken in Deventer.

'Zakboekje voor de ZX Spectrum', geschreven door Wessel Akkermans en uitgegeven door Kluwer Technische Boeken.

Appendix B

Conversietabel hexadecimaal – decimaal
De cijfers in een hexadecimaal getal moeten afhankelijk van hun plaats binnen dat getal worden vermenigvuldigd met een macht van 16 om de decimale waarde te verkrijgen. Vervolgens moeten de verkregen decimale getallen bij elkaar worden opgeteld om het eindresultaat te krijgen.

Voorbeeld: Converteren van het hexadecimale getal '3A5C'.

Hier staat in feite:

C.16^0	=	12
5.16^1	=	80
A.16^2	=	2560
3.16^3	=	12288
		+
		14940

In de volgende tabel zijn vier kolommen opgenomen, zodat hexadecimale getallen van vier cijfers kunnen worden geconverteerd naar decimale getallen, zonder dat u de vermenigvuldigingen nog hoeft te doen. De meest linkse kolom bevat het hexadecimale cijfer dat met de derde macht van 16 is vermenigvuldigd, de meest rechtse kolom bevat de getallen die met de nulde macht van 16 (=1) zijn vermenigvuldigd.
Het optellen van de verkregen tussenresultaten dient u zelf te doen, op de manier als in voorgaand voorbeeld.

hex		dec		hex		dec		hex		dec		hex		dec
0	=	0		0	=	0		0	=	0		0	=	0
1	=	4096		1	=	256		1	=	16		1	=	1
2	=	8192		2	=	512		2	=	32		2	=	2
3	=	12288		3	=	768		3	=	48		3	=	3
4	=	16384		4	=	1024		4	=	64		4	=	4
5	=	20480		5	=	1280		5	=	80		5	=	5
6	=	24576		6	=	1536		6	=	96		6	=	6
7	=	28672	+	7	=	1792	+	7	=	112	+	7	=	7
8	=	32768		8	=	2048		8	=	128		8	=	8
9	=	36864		9	=	2304		9	=	144		9	=	9
A	=	40960		A	=	2560		A	=	160		A	=	10
B	=	45056		B	=	2816		B	=	176		B	=	11
C	=	49152		C	=	3072		C	=	192		C	=	12
D	=	53248		D	=	3328		D	=	208		D	=	13
E	=	57344		E	=	3584		E	=	224		E	=	14
F	=	61440		F	=	3840		F	=	240		F	=	15

Bij het programmeren in BASIC kan het wel eens handig zijn om over een aantal extra-functies te beschikken die niet standaard op uw computer aanwezig zijn. Deze functies, die men ook wel 'utilities' of 'functieprogramma's' noemt, kunnen het programmeren sterk vereenvoudigen. Over het algemeen moet men zelf deze functies samenstellen uit de mogelijkheden die de computer heeft.
Om deze programma's echter redelijk kort te houden en om ze snel te laten zijn, is het wel noodzakelijk om in die programma's gebruik te maken van de reeds in iedere standaard-ZX Spectrum aanwezige ROM-routines.
De auteur van dit boek, zelf een enthousiast Spectrum-programmeur, heeft bij zijn programmeeractiviteiten veel van deze utilities ontworpen en wil deze niet onthouden aan andere Spectrum-gebruikers.
Van de programma's zijn de originele 'listings' afgedrukt zodat de kans op fouten tot een minimum beperkt is.
Hopelijk zult u net als de auteur bij het gebruik van de programma's versteld staan van de mogelijkheden en de soms onverwachte hoge snelheid van uw ZX Spectrum.

ISBN 90 201 1731 9