John Hardman
Andrew Hewson

ZX Spectrum machinetaalroutines

(ook voor ZX Spectrum +)

Kluwer Technische Boeken B.V. - Deventer - Antwerpen

Omslag: W. Niessink
Vertaling: L. H. Navarro

ISBN 90 201 1797 1
D/1985/0108/177
Oorspronkelijke titel: 40 Best machine code routines for the ZX Spectrum

1e druk 1985

Deel B

DEEL A

1. Inleiding

Het doel van dit boek is zowel de beginner als de ervaren computergebruiker te voorzien van een kant en klare referentiegids met een aantal bruikbare, interessante of aangename machinetaalroutines voor de ZX Spectrum. Met dit doel voor ogen werd dit boek in twee delen gesplitst. Deel A beschrijft de kenmerken van de Spectrum die interessant zijn voor de machinetaalprogrammeur – wat wordt bedoeld met machinetaalroutine, de belangrijkste interne kenmerken van de Spectrum en de structuur van de machinetaal op zich.
Deel B toont de routines zelf. Deze zijn opgemaakt in standaardformaat hetgeen in het begin van dit deel in detail wordt uitgelegd. De routines staan geheel op zichzelf, zodat ze afzonderlijk kunnen worden geladen zonder verwijzing naar andere routines.
Het is niet nodig te begrijpen hoe een routine werkt om deze te kunnen gebruiken, omdat elke routine geladen kan worden door middel van de eenvoudige M/C-Lader die aan het begin van Deel B staat. Dus, als u werkelijk ongeduldig bent om bijvoorbeeld de routine 'Variabelenlijst' te gebruiken, ga dan naar de betreffende pagina, toets in en RUN de M/C-Lader (machinecode lader), en toets de decimale getallen in die afgedrukt zijn in de kolom 'In te toetsen getallen'. Wanneer al deze getallen zijn geladen, vergelijk dan de waarde van de check sum, die is afgedrukt bij de M/C-Lader, met de gegeven waarde in de routine zelf. Als deze hetzelfde zijn, kunt u er zeker van zijn dat de getallen goed zijn ingetoetst (tenzij u twee of meer fouten heeft ingetoetst die elkaar opheffen). De routine kan nu door u worden gebruikt.
Als u nog niet genoeg zelfvertrouwen heeft om een lange routine zoals de 'Variabelenlijst' uit te proberen, maar u bent erop gebrand om zo gauw mogelijk met machinetaal te beginnen, kies dan een kortere routine. Zodat als u verdwaalt of te veel fouten maakt, u er niet zoveel tijd aan heeft besteed. De routine 'Scroll attributen omlaag' is ideaal. Nogmaals, het is alleen een kwestie van intikken en 'RUNnen' van de M/C-Lader die aan het begin van deel B uitgelijst wordt, en het kopiëren van de aangegeven getallen in de kolom 'In te toetsen getallen'. Als u klaar bent zorg er dan voor dat de Check sum correct is.
Als u liever het gebruik van de machinetaalroutines uitstelt, lees dan verder. De rest van dit hoofdstuk introduceert de basisbegrippen en gaat door met het in detail uitleggen van de opzet van dit boek. De beginner wordt geadviseerd deze informatie aandachtig te lezen, maar de meer ervaren gebruiker kan volstaan met het vluchtig doornemen ervan.

De Z80A-microprocessor die de ZX Spectrum bestuurt begrijpt geen BASIC-woorden zoals PRINT, IF, TAB enzovoort. In plaats daarvan luistert het naar zijn eigen taal, machinetaal genoemd. De instructies in de Sinclair-ROM die de Spectrum zijn 'personaliteit' geven, zijn geschreven in deze speciale taal en ze bestaan uit een groot aantal routines voor het intoetsen, het uitlijsten, het vertalen en uitvoeren van het eigen BASIC-dialect dat de Spectrum gebruikt. In werkelijkheid vormen de routines een groep van 'WAT TE DOEN ALS'-routines. Bijvoorbeeld, ze vertellen de Z80A WAT TE DOEN ALS de volgende instructie het woord PRINT is, en dan WAT TE DOEN ALS de volgende instructie een variabelenaam is, en DAN WAT TE DOEN ALS de volgende instructie een komma is, enz. enz.

Machinetaal bestaat uit een serie positieve gehele getallen ieder kleiner dan 256, en het schrijft de werking voor van de Z80A door het aanzetten van acht schakelaars in overeenstemming met het patroon van de binaire waarde van het getal. De binaire waarde van 237 bijvoorbeeld is 11101101, zodat wanneer het getal 237 wordt aangetroffen de acht schakelaars respectievelijk aan, aan, aan, uit, aan, aan, uit, aan worden gezet.
Juist omdat de machine naar de binaire versie van het getal luistert, is het voor de mens niet nodig de instructie in deze vorm te bekijken. Wij zijn gewend decimalen te gebruiken en daarom herkent de M/C-Lader van deel B deze vorm. Echter, zelfs een serie decimale getallen is moeilijk te vertalen en dus worden de decimalen gewoonlijk opnieuw vertaald naar een *assembly-taal* die een beetje geheimzinnig lijkt maar in de praktijk niet zo moeilijk te gebruiken is. Elke routine in deel B wordt zowel in assembly-taal als in decimale getallen uitgedrukt.
Assembly-taal wordt zo genoemd omdat er een speciaal programma, *assembler* genoemd, goed kan worden gebruikt om veel machinetaalinstructies bij elkaar te brengen (assembleren) om een nieuw programma te vormen. Assemblers zijn goed beredeneerde programma's omdat de machinetaal erg uitgebreid is en gewoonlijk zijn ze in machinetaal geschreven. Sommige van die assemblers zijn verkrijgbaar voor de Spectrum en terwijl de routines in dit boek met gebruik van een assembler konden worden geladen, wordt benadrukt dat de M/C-Lader voor dit doel zeer geschikt is.
Eén getal alleen is voldoende om de meest eenvoudige Z80A-instructies te bepalen. De instructie om de inhoud van register d naar register c te kopiëren is bijvoorbeeld decimaal 81. (De betekenis van het woord register wordt verder uitgelegd in hoofdstuk 3. Op dit moment is het voldoende om over c en d te denken als verwant aan BASIC-variabelen). Voor deze instructies is er een één-op-één overeenkomst tussen het decimale getal en de assembly-taalversie, zodat bijvoorbeeld decimaal 81 in assembly-taal geschreven wordt als ld d, c. 'ld' staat voor 'laad'. Veel assembly-taalinstructies bestaan uit eenvoudig vergelijkbare afkortingen en om deze reden worden ze vaak mnemonics genoemd.
Meer gecompliceerde instructies hebben twee, drie of zelfs vier getallen nodig voordat ze helemaal gespecificeerd worden. In dit geval wordt één enkele assemblermnemonic gebruikt om ze allemaal uit te drukken. In tabel 1.1 staan enkele voorbeelden van getallen met hun mnemonics en een korte uitleg.
Regel (a) van de tabel is het voorbeeld van ld d, c waarover we hiervoor hebben gesproken. Regels (b) en (c) laten zien hoe een positief geheel getal kleiner dan

Tabel 1.1: Enkele voorbeelden van machinetaalinstructies voor de Z80A

Ref	*Decimaal*	*Assembly-taal*	*Commentaar*
(a)	81	ld, d, c	Laad d met inhoud van c.
(b)	14 27	ld c, 27	Laad c met 27.
(c)	14 13	ld c, 13	Laad c met 13.
(d)	33 27 52	ld hl, 13339	Laad 13339 in het registerpaar hl. Opm.: 27 + 256 × 52 = 13339; 27 gaat naar l; 52 naar h.
(e)	221 33 27 52	ld ix, 13339	Laad 13339 in het registerpaar ix.

256 in een register mag worden geladen door middel van twee achtereenvolgende getallen – het eerste geeft aan wat moet worden gedaan en het tweede specificeert het te laden getal. Regel (d) laat zien hoe een groot getal geladen mag worden in twee registers, h en l, samen. Nu geven het tweede en derde getal aan welk getal moet worden geladen. Het laatste voorbeeld in regel (e) laat een vier-getallen-code zien voor het laden van een groot geheel getal in het registerpaar ix. Merk op hoe drie van de vier getallen ook in regel (d) voorkomen. In feite specificeert het eerste getal het ix-paar in plaats van het hl-paar.
De structuur van machinetaal wordt in hoofdstuk 3 meer in detail uitgelegd en een complete lijst van de assembler-mnemonics wordt gegeven in appendix A. Op dit moment is het belangrijker dat de volgende vraag beantwoord wordt:

Waarom machinetaal gebruiken?
Het schijnt altijd zo te zijn dat er met elke programmeertaal op elke computer taken zijn waarvan de gebruiker wil dat ze uitgevoerd worden, maar welke niet behoorlijk in de beschikbare taal zijn te schrijven of indien dat wel gebeurt de uitvoering zeer langzaam is. De ZX Spectrum is geen uitzondering in dit opzicht. Neem bijvoorbeeld het probleem van het opslaan van het hele scherm op de top van de RAM en dit terug te kopiëren met de bedoeling een tekenfilm-effect te creëren door het snel wisselen van verschillende beelden. De display file (beeldschermbestand) en de attributen (gegevenskenmerken) vergen in totaal 6912 bytes en dus is het nodig de RAMTOP te verlagen naar 32768−6912=25856 bij de 16K machine om voor genoeg ruimte te zorgen voor de kopie van het scherm buiten het BASIC-gedeelte (65536−6912=58624 bij de 48K machine). Het volgende eenvoudige BASIC-programma slaat het scherm op in het geheugen maar doet er erg lang over, ongeveer 70 seconden:

```
10 FOR i = 0 to 6911
20 POKE 25856+i, PEEK (16384+i)
30 NEXT i
```

De reden waarom het zo lang duurt is dat de Spectrum de meeste tijd gebruikt voor het decoderen van de instructies voordat die worden uitgevoerd. Een bepaalde tijd wordt ook gebruikt voor het omzetten van de getallen van de twee-byte vorm die de Z80A begrijpt, naar de decimale vijf-byte vorm waarin de lusteller wordt bijgehouden en ook voor het uitvoeren van vijf-byte rekenkundige bewerkingen. De te volgen stappen zijn:

1) Tel i op bij 16384.
2) Zet het resultaat om in de twee-byte vorm.
3) Haal de inhoud van het PEEK-adres binnen.
4) Tel i op bij 25856.
5) Zet het resultaat om in de twee-byte vorm.
6) Zet de gehaalde waarde in het POKE-adres weg.
7) Tel 1 op bij i en zet het resultaat weg.
8) Trek i van 6911 af. Als het resultaat positief of nul is, ga dan naar 1.

Iedere keer dat de lus wordt doorlopen moet de Spectrum iedere instructie opnieuw decoderen omdat het geen weet meer heeft van de vorige bewerkingen. Het is makkelijk te zien dat de computer meer dan 99% van de tijd kwijt is aan het voorbereiden van de taak, dan aan de uitvoering zelf. Daarom is het geen verrassing te weten dat een machinetaalroutine voor het opslaan van het scherm min of meer direct werkt. Een voorbeeld van een routine wordt in deel B gegeven.

Hoe machinetaal leren

De machinetaal van de Z80A is erg ingewikkeld en er is een goed naslagwerk nodig om al zijn faciliteiten te begrijpen, maar tevens veel denkwerk en veel praktijk. Er zijn verschillende boeken verkrijgbaar. Het standaardnaslagwerk is *'Programmeren van de Z80'* van Rodney Zaks, uitgegeven door Sybex. Het bevat erg veel informatie over de hardware-organisatie van de microprocessor, evenals een complete lijst van de instructieset. De beginner zal het nogal indrukwekkend vinden omdat het meer dan 600 pagina's telt.

Een wat meer leesbaar verslag is te vinden in *'Programmering van de Z80 en 8080'* van Kathe Spacklen, uitgegeven door Hayden. Dit boek begint op een eenvoudiger niveau, bestrijkt de belangrijkste software-onderwerpen en negeert de hardware bijna in zijn geheel.

Dit boek is niet alleen bedoeld als een inleiding in de machinetaal voor de beginner, maar ook om de meer ervaren gebruiker van dienst te zijn. Het geeft de lezer een sterke prikkel om machinetaal te leren door hem van routines te voorzien die hij in zijn eigen BASIC- of machinetaalprogramma's mee kan nemen, met of zonder aanpassingen.

De meeste routines hangen grotendeels af van de structuur van de Spectrum en zodoende behandelt het volgende hoofdstuk dit onderwerp meer in detail. Het behandelt bijvoorbeeld de vorm van de display file, het programmageheugen en het variabelengebied, legt de opzet van BASIC-regels uit en introduceert de vijfbyte rekenkunde met drijvende komma. Er wordt aangenomen dat de lezer van deel B op de hoogte is van de inhoud van dit hoofdstuk.

Het derde hoofdstuk behandelt enigszins gedetailleerd de Z80-machinetaal en beschrijft veel punten die later als bekend worden aangenomen. Het bevat een lijst van de instructieset met de meest opvallende kenmerken, zonder het gedetailleerde verslag uit het boek van Zaks te herhalen.

2. Interne structuur van de ZX Spectrum

Een computer is een machine die in staat is een serie opdrachten te onthouden en die daarna uit te voeren. Om dit te doen heeft het een geheugen nodig waarin de instructies kunnen worden opgeslagen. De ZX Spectrum heeft twee verschillende typen geheugens. Het eerste type is het ROM-geheugen (read-only-memory) die de vaste instructieset bevat die de fabrikant in de machine heeft ingeprent. Het tweede type is het willekeurig toegankelijke geheugen (random-access-memory). De RAM is het kladblok van de Spectrum. Als de computer met een taak bezig is, kijkt het constant naar wat er in de RAM aanwezig is ('lezen' van het geheugen) en verandert de inhoud van de RAM ('schrijven' naar het geheugen). De Spectrum gebruikt het kladblok niet op goed geluk. Verschillende gedeeltes van de RAM worden gebruikt om verschillende soorten informatie in op te slaan. Een BASIC-programma welke door de gebruiker is ingetoetst wordt in een bepaald gedeelte opgeslagen, terwijl de variabelen die door het programma worden gebruikt, elders worden opgeslagen. De afmeting van het kladblok is beperkt en dus zorgt de machine ervoor dat er precies genoeg ruimte wordt gereserveerd voor de informatie die het bevat. Zo wordt de vrije ruimte altijd op één plaats bijgehouden zodat, als de gebruiker bijvoorbeeld een nieuwe regel aan zijn programma wil toevoegen, de informatie in RAM kan worden opgeschoven door de vrije ruimte te gebruiken om de extra regel te plaatsen.
Het grootste gedeelte van dit hoofdstuk wordt gebruikt om in detail uit te leggen hoe de Spectrum de RAM verdeelt omdat veel van de routines uit deel B werden gemaakt om de RAM te manipuleren. Daarom, moet de lezer, als hij de opzet van de routines wil begrijpen in plaats van deze blindelings te gebruiken, de inhoud van dit hoofdstuk begrijpen. Dit hoofdstuk behandelt de display file, de attributen, de printer-buffer, de systeemvariabelen, het programmagedeelte en het variabelengebied. Het laatste stuk gaat over de routines in ROM, die in deel B worden geraadpleegd.

2.1 Geheugenindeling

Er zijn 16384 geheugenlocaties in RAM, in de niet uitgebreide ZX Spectrum (de uitgebreide versie bevat nog 32768 locaties extra, wat een totaal van 49152 locaties oplevert). Elke locatie kan een enkel getal bevatten tussen 0 t/m 255 en wordt geïdentificeerd door zijn adres. Het adres wordt voorgesteld door een positief geheel getal.
Adressen 0 tot en met 16383 worden aan het vaste geheugen toegekend, de ROM, en dus is het eerste adres in RAM 16384. Tabel 2.1 is de geheugenindeling van de Spectrum en laat zien hoe de RAM, dat op adres 16384 begint, wordt gebruikt. De display file bijvoorbeeld, die de informatie bevat die gewoonlijk op het scherm te zien is, gebruikt de locaties 16384 tot 22527. De attributen, die de kleur, de helderheid enzovoort van het beeldscherm bepalen, volgen er onmiddellijk achter in de locaties 22528 tot 23295.
De eerste vijf startadressen in kolom 1 van tabel 2.1 staan allemaal vast, omdat de display file, de attributen enzovoort, allemaal een vaste hoeveelheid ruimte in beslag nemen. Het vijfde gebied wordt aan de microdrive-gegevens toegekend. Als een microdrive aan de Spectrum is aangesloten bevat dit gebied informatie voor de opmaak van data in de microdrive. Als er geen microdrive is aangesloten, is dit gebied niet nodig. In dit geval wordt het zesde gebied, de kanaalinformatie, on-

middellijk achter het vierde gebied, de systeemvariabelen geplaatst, overeenkomstig de gewoonte van de Spectrum om ruimte te besparen waar het mogelijk is. Vandaar dat het startadres van het kanaal-informatiegebied niet vast is, maar in de RAM op en neer kan 'drijven'.
De Spectrum houdt het startadres van al deze variabelen bij door de actuele waarde van elk adres in het systeemvariabelengebied op te slaan. Het systeemvariabelengebied ligt voor de microdrive-gegevens op de locaties 23552 tot en met 23733 en er is dus geen sprake van verplaatsing van dit gebied binnen de RAM! Het adres van het gebied dat het startadres van al de 'drijvende' gebieden bijhoudt, is afgedrukt in kolom twee van tabel 2.1. Het adres van het gebied van het BASIC-programma bijvoorbeeld, wordt op adres 23635 bijgehouden, binnen het systeemvariabelengebied.

Tabel 2.1: De geheugenindeling (memory map). De stapelwijzer (stack pointer) zit niet in RAM maar in het sp-register van de Z80A-microprocessor.

Startadres of naam systeemvariabele	*Locatie systeemvariabele*	*Geheugeninhoud*
16384	- - - -	Display file.
22528	- - - -	Attributen.
23296	- - - -	Printer-buffer.
23552	- - - -	Systeemvariabelen.
23734	- - - -	Microdrive map.
CHANS	23631	Kanaal-informatie.
PROG	23635	BASIC-programma.
VARS	23627	Variabelen.
E-LINE	23641	Opdracht/regel die wordt opgemaakt (editing).
WORKSP	23649	Ingegeven data.
STKBOT	23651	Stapel rekenorgaan.
STKEND	23653	Vrij.
sp	- - - -	Machine- en GOSUB-stapel.
RAMTOP	23730	Gebruiker machinetaalroutines.
UDG	23675	Door gebruiker gedefinieerde graphics.
P-RAMT	23732	Einde van de RAM.

Verwijzingen naar het adres waar een systeemvariabele zit is erg onhandig en dus heeft iedere variabele een naam gekregen, PROG in het geval van de locatie die het adres van het gebied van het BASIC-programma bevat. Deze namen zijn alleen voor het gemak van de gebruiker, omdat de Spectrum ze niet als zodanig herkent. Dus intoetsen van de regel:

```
PRINT PROG
```

zal de foutmelding geven: '2 Variable not found', wat op zich betekent dat de te PRINTen variabele niet wordt gevonden, tenzij de BASIC-variabele –PROG– toevallig door de gebruiker of een programma was gegenereerd. De waarde van zo'n BASIC-variabele zou niets te maken hebben met de waarde van de systeemvariabele PROG.

2.2 PEEK en POKE

De geheugenindeling is de sleutel om het gebruik van de RAM te begrijpen, maar de sleutels om de RAM te onderzoeken zijn de woorden PEEK en POKE. Deze stellen de gebruiker in staat de inhoud van elke geheugenlocatie te bekijken, respectievelijk te veranderen.

PEEK is een functie van de vorm:

PEEK adres

Het adres kan een positief geheel getal zijn tussen 0 en 65535 of een rekenkundige uitdrukking waarvan de uitkomst zo'n getal oplevert. Het is belangrijk om een rekenkundige uitdrukking tussen haakjes te zetten omdat:

```
PEEK 16384 + 2
```

geïnterpreteerd wordt als tel 2 op bij het resultaat van:

```
PEEK 16384
```

terwijl:

```
PEEK (16384 + 2)
```

geïnterpreteerd wordt als:

```
PEEK (16386)
```

De door de functie PEEK verkregen waarde is de actuele inhoud van het adres in kwestie, welke altijd een positief geheel getal zal zijn tussen 0 t/m 255. Hierboven werd uitgelegd dat de systeemvariabele PROG op adres 23635 staat, maar de waarde ervan die een adres in RAM vertegenwoordigt, is altijd veel groter dan 255 en daarom zijn er twee adressen nodig, 23635 en 23636, om die te bevatten. De waarde van PROG kan zo worden verkregen:

```
PRINT "PROG="; PEEK 23635+256*PEEK 23636
```

Alle adressen worden op dezelfde manier in twee locaties opgeslagen en kunnen zo worden bekeken:

PRINT PEEK *eerste adres* + 256*PEEK *eerstvolgend adres*

Als een Spectrum bijvoorbeeld zonder een aangesloten microdrive wordt gebruikt, dan zal de microdrive-map niet bestaan en de kanaal-informatie zal onmiddellijk achter het systeemvariabelen-gebied volgen. Dus zal de waarde van de systeemvariabele CHANS dezelfde zijn als het startadres van de microdrive-map, als deze bestaat, nl. 23734. CHANS staat op adres 23631 *en* 23632, zo zal het intoetsen van:

```
PRINT PEEK 23631+256*PEEK 23632
```

de waarde 23734 opleveren.

De functie PEEK kan worden gebruikt om elke geheugenlocatie te bekijken, inclusief de vaste instructies in ROM. Het is dus een belangrijk stuk gereedschap. Het 'PEEKen' van een locatie zal niet tot gevolg hebben dat de Spectrum vastloopt of het programma of de variabelen verstoren. Zeer zelden kan het resultaat van een PEEK misleidend zijn omdat de inhoud van de locatie die gePEEKed wordt, tijdens of gelijk na het uitvoeren van de instructie kan veranderen. Bijvoorbeeld, als de inhoud van de locatie die toegewezen wordt aan de linker bovenhoek van het scherm wordt gePEEKed, en het resultaat ervan in de linker bovenhoek wordt geprojecteerd, zal het iedere keer dat de gebruiker dit bekijkt anders zijn.
De POKE-opdracht is in alle opzichten gevaarlijker dan de PEEK-functie, omdat bij gebruik hiervan de gebruiker de werking van de Spectrum doorkruist. Dus is het best mogelijk dat bij gebruik ervan de informatie in RAM wordt verminkt met als gevolg het vastlopen van de machine, of het verschijnen van een foutboodschap. De vorm van de opdracht is:

POKE *adres, getal*

Nogmaals, het adres is een positief geheel getal tussen 0 t/m 65535, of een rekenkundige uitdrukking waarvan de uitkomst zo'n getal oplevert. Het is in dit geval niet essentieel dat de rekenkundige uitdrukking tussen haakjes wordt gezet omdat POKE een opdracht is en geen functie en daarom kan het niet als één geheel worden gerekend. De waarde die gePOKEd wordt moet liggen tussen 0 t/m 255.
De Spectrum zal een POKE-opdracht voor een ROM-adres (tussen 0 en 16383) accepteren en uitvoeren, maar het getal zal nooit zijn bestemming bereiken. Dit feit kan worden bewezen door het volgende programma te RUNnen:

```
10 PRINT PEEK 0
20 POKE 0,92
30 PRINT PEEK 0
```

Regels 10 en 30 zullen elk de waarde 243 PRINTen, wat de inhoud van locatie 0 blijkt te zijn. Regel 20 heeft geen effect.

2.3 De display file

Het normale display (beeldscherm) bestaat uit 24 regels van 32 tekens. We hebben gezien dat de display file de locaties 16384 t/m 22527 in beslag neemt; dat wil zeggen 6144 locaties in totaal. Dus het gebruikte aantal locaties per teken is:

```
6144/(32*8)= 8
```

De beste manier om een globale indruk te krijgen van hoe het display is georganiseerd, is een beeld te PRINTen, dit op cassette wegzetten (SAVE), het scherm schoonmaken en het beeld opnieuw laden (LOAD). Programma P2.1 laat deze toepassing zien en gebruikt daarbij het graphics-teken 5 om een origineel beeld te creëren.
Als het beeld opnieuw van cassette wordt geladen, wordt het duidelijk dat het display in drie delen van elk acht regels is verdeeld. Verder is elke regel in acht regels met *pixels*(beeldelementen) verdeeld. Echter, de Spectrum laadt niet de eerste acht pixelregels die de eerste tekenregel vormen, gevolgd door de acht pixel-regels die de tweede tekenregel vormen, enzovoort. In plaats daarvan laadt

het de eerste acht pixel-regels van de eerste acht tekenregels, gevolgd door de volgende pixelregels van dezelfde acht tekenregels, enzovoort. De top van het display die uit acht tekenregels bestaat en de laatste acht tekenregels vormen respectievelijk het midden en de bodem van het display.

Programma P2 1: Een programma om het scherm op te slaan, schoon te maken en te laden

```
1ØØ FOR i = Ø TO 7Ø3
11Ø PRINT "■";
12Ø NEXT i
13Ø SAVE "Beeld" SCREEN$
14Ø CLS
15Ø INPUT "Spoel cassette terug, druk op 'play' en
    druk hierna een toets in"; z$
16Ø LOAD "Beeld" SCREEN$
```

Een andere manier om te begrijpen hoe het display in elkaar zit, is te bekijken waar de acht bytes zitten die gebruikt worden om de linker bovenhoek van het scherm te vormen. De eerste byte vormt het bovenste achtste deel van het teken en bevindt zich aan het begin van de display file op adres 16384. Na een korte tijd van experimenteren zien we dat:

POKE 16384,0

de bovenste regel schoonmaakt die de top vormt van het eerste teken, terwijl:

POKE 16384,255

de oorzaak ervan is dat alle pixels zichtbaar worden. Getallen POKEn tussen 0 en 255 hebben een spikkeltjeseffect tot gevolg
De tweede van de acht pixel-regels van het eerste teken op het scherm wordt niet gevormd door het getal dat op adres 16385 zit, want deze locatie wordt gebruikt voor de bovenste pixel-regel van het volgende teken. Er zitten 32 tekens in één regel en 8 in één gedeelte zodat de tweede pixel-regel van het eerste teken gevormd wordt door het getal van locatie:

16384+32*8=16640

Deze redenering is ook van toepassing op de resterende zes van de acht pixelregels; dus de vorm van het eerste teken in de linker bovenhoek van het scherm wordt bepaald door de inhoud van de adressen:

16384, 16640, 16896, 17152, 17408, 17664, 17920, 18176

Programma P2.2 stelt de gebruiker in staat verschillende getallen in deze locaties te POKEn.
Iedere locatie binnen de display file bepaalt de toestand van acht pixels op het scherm. Dit wordt gedaan door het getal uit de betreffende locatie in zijn binaire vorm om te zetten en dan de acht pixels in overeenstemming te brengen met het

Programma P2.2: Een programma om een teken in de linker bovenhoek te zetten

```
1Ø  REM Routine om een teken in de linker bovenhoek
    van het scherm te zetten
2Ø  INPUT "Een teken bestaat uit 8 bytes met een
    waarde tussen Ø t/m 255, geef aantal bytes op
    (Ø t/m 7)"; n
3Ø  IF n<Ø OR n>7 OR n<> INT n THEN BEEP .2,24:GO
    TO 2Ø
4Ø  INPUT "Geef waarde van de byte";m
5Ø  IF m<Ø OR m>255 OR m <>INT m THEN BEEP .2,24:GO
    TO 4Ø
6Ø  POKE 16384+8*32*n,m
```

Programma P2.3: Programma om een attribuut te decoderen

```
 1Ø REM Attribuut decoder
 2Ø DATA "Zwart", "Blauw", "Rood", "Magenta",
    "Groen", "Cyaan", "Geel", "Wit", "Bright",
    "Flash"
 3Ø DIM c$(8,7)
 4Ø FOR i=1 TO 8
 5Ø READ c$(i)
 6Ø NEXT i
1ØØ REM Attribuut decoder
11Ø INPUT "Geef een getal tussen Ø en 255. Dit
    programma decodeert de betekenis in de
    attributen-file";n
12Ø IF n<Ø OR n>255 OR n<>INT n THEN BEEP .2,24: GO
    TO 11Ø
2ØØ PRINT "Ink-kleur is ";c$(1+n-8*INT(n/8))
21Ø PRINT "Paper-kleur is ";c$(1+INT
    (n/8)-8*INT(n/64))
22Ø IF INT (n/64)=1 OR INT (n/64)=3 THEN PRINT
    "Teken met BRIGHT effect"
23Ø IF n>127 THEN PRINT "Teken met FLASH effect"
3ØØ PRINT AT 6,Ø; "
    "
31Ø FOR i=22720 TO 22751
32Ø POKE i,n
33Ø NEXT i
5ØØ INPUT "Toets ENTER in voor herhaling";z$
51Ø CLS
52Ø GO TO 11Ø
```

nul/éénpatroon van de acht binaire digits. Bijvoorbeeld 240 in binaire vorm wordt.

11110000

Dus als een locatie het getal 240 bevat zullen vier van de acht pixels worden verlicht en de resterende vier zullen blank blijven.
Samenvattend: de display file bestaat uit 6144 locaties met acht locaties toegewezen aan iedere tekenpositie. Iedere locatie bepaalt de toestand van een horizontale streep van acht pixels. De toegewezen locaties voor een gegeven tekenpositie staan niet achter elkaar; in plaats daarvan is het display ingedeeld in acht delen, en elk deel bestaat uit 256 locaties die de samengestelde bytes van elke positie onderscheiden.

2.4 De attributen

De inhoud van de display file bepaalt alleen wélke pixels worden verlicht. De kleur van de PAPER-, INK-, BRIGHT- en FLASH-toestand wordt door de attributen bepaald. Het attributengebied bezet de locaties 22528 tot 23295 met één locatie voor elk van de 768 tekenposities. In tegenstelling tot de display file, worden de locaties aan de tekens op de goede manier toegekend; dat wil zeggen, beginnend in de linker bovenhoek van links naar rechts en doorgaand van de top tot de bodem.
Iedere locatie zet zowel de INK als PAPER van de corresponderende positie in één van de acht kleuren om die de bovenste rij van het spectrum-toetsenbord laat zien. Het bepaalt ook wanneer de positie BRIGHT, of FLASH is. De vier parameters worden gecodeerd door middel van de volgende berekening:

Attribuutwaarde=128*FLASH + 64*BRIGHT + 8*PAPER + INK

FLASH en BRIGHT krijgen de waarde één als de betreffende toestand van toepassing is en PAPER en INK krijgen de waarde van de betreffende kleur zoals op het toetsenbord te zien is (rood is 2 bijvoorbeeld). Programma P2.3 decodeert de attributen, dat wil zeggen, eenmaal een attribuutwaarde gegeven, zal het de corresponderende PAPER- en INK-kleur afdrukken.

2.5 De printer-buffer

De 256 locaties in RAM achter het attributengebied worden gebruikt om tijdelijk, een incomplete tekenregel te bewaren, die later naar de printer overgebracht moet worden. De buffer is nodig omdat een BASIC-programma een gedeelte van een regel kan afdrukken (print) door de regel met een komma of puntkomma te beëindigen, om aan te geven dat de rest ervan nog moet komen. In bepaalde omstandigheden kan een TAB-opdracht op dezelfde manier werken. Een gedeelte van een regel kan niet direct aan de printer worden doorgegeven, omdat de printer alleen een complete regel kan drukken, het papier omhoog schuift, en zich zo voorbereidt op de volgende regel.
Daarom wordt een deel van een regel tijdelijk in de printer-buffer opgeslagen totdat het programma de rest ervan met L-PRINT verwerkt.
Veel van de routines in deel B maken gebruik van de printer-buffer om data vanuit het BASIC-programma op het toetsenbord naar de routines door te geven. De buffer is handig voor dit doel, omdat hij een vaste locatie heeft, en omdat de ge-

bruiker het waarschijnlijk niet voor een ander doel gaat gebruiken als hij een machinetaalroutine aanroept.

2.6 Het gebied van het BASIC-programma

Als een microdrive op de Spectrum is aangesloten, zal het begin van het gebied van het BASIC-programma worden bepaald door de systeemvariabele PROG op adres 23635 te raadplegen. Bij afwezigheid van een microdrive begint dit gebied op 23755. In dit verslag wordt aangenomen dat er geen microdrive is aangesloten. Het vier regels tellende programma P 2.4 PRINT de inhoud van de 18 locaties aan het begin van het programmagebied zoals het op afbeelding 2.1 te zien is. Deze 18 locaties worden gebruikt om de eerste regel te bevatten dat wil zeggen:

10 REM Peek program

Er kan veel worden geleerd over de coderingsmethode van het programma, door afbeelding 2.1 te bestuderen.

Programma P2.4: Een programma om de inhoud van de eerste achttien locaties van het programmagebied af te drukken

```
1Ø  REM Peek programma
2Ø  FOR i = 23755 TO 23772
3Ø  PRINT i, PEEK i
4Ø  NEXT i
```

23755	0
23756	10
23757	14
23758	0
23759	234
23760	80
23761	101
23762	101
23763	107
23764	32
23765	112
23766	114
23767	111
23768	103
23769	114
23770	97
23771	109
23772	13

Afbeelding 2.1: De vorm waarin de regel: 10 REM Peek program in het programmagebied zit

Het regelnummer 10, wordt in de eerste twee locaties opgeslagen in deze vorm:

Regelnummer = 256* PEEK *eerste adres* + PEEK *tweede* adres

Merk op dat de afspraak van de Z80A om de inhoud van de tweede byte met 256 te vermenigvuldigen en dit bij de eerste byte op te tellen, hier niet toegepast wordt.
De afspraak wordt toegepast op de twee volgende locaties, 23757 en 23758, welke samen de lengte van de rest van de regel bevatten, beginnend vanaf adres 23759. Het opgeslagen getal in dit geval is:

14 + 256*0 = 14

De volgende regel begint dus op locatie:

23759 + 14 = 23773

De locatie 23759 zelf bevat het getal 234, dat de tekencode is van REM. De volgende 12 locaties bevatten de tekencodes van de elf letters en een spatie in de titel:

Peek program

Ten slotte bevat locatie 23772 het getal 13, wat de code is van het ENTER-teken dat het einde van de regel aangeeft. Tabel 2.2 vat de methode samen om programma's in het programmagebied te decoderen.

Tabel 2.2: De gebruikte methode om programmaregels te decoderen

Locaties	*Inhoud*
1 en 2	Regelnummer andersom opgeslagen dan de afspraak bij de Z80A.
3 en 4	Lengte van de regel, buiten de eerste 4 locaties.
5	De opdrachtcode.
Laatste	Het ENTER-teken, getal 13.

23755	0
23756	10
23757	14
23758	0
23759	241
23760	97
23761	61
23762	49
23763	52
23764	52
23765	51
23766	14
23767	0
23768	0
23769	163
23770	5
23771	0
23772	13

Afbeelding 2.2: De vorm waarin de regel: 10 LET a=1443 in het programmagebied zit

Een punt dat buiten de tabel werd gelaten is de omschrijving van de methode die gebruikt wordt om waardes op te slaan die in het programma voorkomen. Deze methode kan worden onderzocht door het vervangen van de regel:

```
10 LET a = 1443
```

in het programma P 2.4. Afbeelding 2.2 laat het resultaat zien van het RUN-nen van het programma:
Locaties 23755 tot 23758 zijn onveranderd. Deze worden gevolgd door de codes van LET, a, =, en de vier digits die samen het getal 1443 vormen. Het volgende punt in locatie 23766 is 14, de tekencode die aangeeft dat de volgende vijf locaties het getal 1443 in numerieke vorm bevatten. De regel wordt weer door het ENTER-teken op locatie 23772 beëindigd.

2.7 Getallenvoorstelling in vijf bytes

Voor elk getal dat in een BASIC-programma voorkomt worden vijf bytes gebruikt.
Dit geldt zoals we gezien hebben niet voor de regelnummers. Gehele getallen tussen −65535 en 65535 worden opgeslagen volgens een met de Z80A overeengekomen manier. Voor deze getallen bevatten de eerste en de laatste locatie elk nul, en de derde en vierde bevatten het getal in deze vorm:

getal = PEEK *derde locatie* + 256*PEEK *vierde locatie*

Het getal 16553 bijvoorbeeld, wordt dus in 5 locaties gevat als:

```
0 0 169 64 0
```

omdat

169+256*64=16553

Niet-gehele getallen worden in de drijvende-komma-vorm (Engels: floating point) gevat met de exponent in de eerste locatie en een mantisse in de volgende vier bijvoorbeeld:

getal = mantisse* 2 ↑ exponent

Programma P2.5: Dit programma herleidt een niet-geheel getal vanuit zijn vijf-componenten-waardes.

```
1Ø  PRINT "Geef de exponent en de vier bytes van de
    mantisse. Alle getallen moeten liggen tussen Ø
    t/m 255"
2Ø  INPUT e,a,b,c,d
3Ø  PRINT , , "Exponent = ";e
4Ø  PRINT "Mantisse = ",a,,b,,c,,d
5Ø  PRINT , , "Het getal = "; (2 * (a<128)-1) *
    2↑(e-16Ø) * (((256 * (a+128 * (a<128))+b) *
    256+c) * 256+d)
```

De eerste locatie van de mantisse wordt ook gebruikt om het teken van het getal te bepalen. Als de locatie een waarde tussen 0 en 127 bevat is het getal positief, anders negatief.
Het programma P2.5 kan worden gebruikt om een niet-geheel getal vanuit zijn vijf-componenten-waardes te herleiden.

2.8 Het variabelengebied

Het variabelengebied begint op het adres dat in de systeemvariabele VARS zit, die op zijn beurt op adres 23627 zit. Wanneer een nieuwe variabele wordt vastgesteld vanuit het programma of vanuit het toetsenbord, dan wordt binnen dit gebied de juiste ruimte ervoor gecreëerd. Alle variabelenamen moeten met een letter beginnen en er wordt geen onderscheid gemaakt tussen hoofdletters en kleine letters. Deze beperkingen stellen de Spectrum in staat de tekencode van de eerste letter van elke variabele te manipuleren zodat hij de zes toegestane variabelesoorten kan onderscheiden door het bereik te inspecteren waarin deze codes liggen. Alle numerieke variabelen met een enkele tekennaam bijvoorbeeld, hebben codes tussen 97 en 122, waarbij a de code 97 heeft, b 98, c 99 enzovoort. Op dezelfde manier hebben numerieke arrays codes in het bereik 129 tot 153, waarbij a de code 129 heeft, b 130, c 131, enzovoort. Het codebereik wordt opgesomd in tabel 2.3. De lengte van elke variabelesoort is ook te zien in tabel 2.3.

Tabel 2.3: Variabelen, het bereik van de tekencodes en de lengte van de variabelen

Variabelesoort	*Bereik tekencode*	*lengte in variabelen-gebied*
Numeriek (enkele tekennaam)	97 tot 122	6
Numeriek (veelvoudige tekennaam)	161 tot 186	5 + lengte naam
Numeriek-array	129 tot 154	4+2*aantal dimensies+ 5*totaal aantal elementen.
Controlevariabele voor een FOR-NEXT-lus	225 tot 250	18
String	65 tot 90	3+ lengte string
Teken-array	193 tot 218	4+ aantal dimensies+ totaal aantal elementen

2.9 ROM-routines

Sommige van de routines in deel B gebruiken de routines in ROM als volgt:

rst 16
PRINT de inhoud van de accumulator

call 3976
Het teken in de accumulator invoegen bij het adres dat in het hl-registerpaar staat.

call 4210
Verwijder één teken bij het adres dat in hl zit.

call 6326
Als de accumulator het getal 14 bevat, zet de zero-vlag op, en verhoog het hl-register met vijf.

call 6510
Kom terug met in hl het adres in RAM, van de regel waarvan het nummer in de hl-registers werd meegegeven.

3. Z80A-machinetaal

Dit hoofdstuk begint met het uitleggen van woorden als *bit, byte, adres,* en *register.* die verder in dit hoofdstuk als bekend worden aangenomen. Het aantal en de verscheidenheid van de Z80A-registers wordt dan bekeken waarbij de nadruk in het bijzonder valt op een klein aantal instructies die als voorbeeld dienen. Er wordt tenslotte een opsomming van dc instructieset gegeven.
Het moeilijkste aspect voor de beginner in de programmering in machinetaal is misschien wel het aantal nieuwe woorden en begrippen die hij in zich op moet nemen. Laten we daarom, voordat we ons aan het hoofdstuk zelf wagen, één instructie als voorbeeld nemen van wat er gaat komen. Bekijk de volgende samengestelde instructie die we in veel van de routines van deel B zullen vinden:

```
ld hl, (23627)
```

Deze instructie moet gelezen worden als *laad* het *hl-registerpaar* met de *bytes* van *adres* 23627 en 23628. Elk van de woorden in cursief wordt later in meer detail in dit hoofdstuk uitgelegd.
Deze instructie wordt in de vorm van drie decimale getallen – 42, 75, 92 – omgezet. Het eerste getal betekent:

```
ld hl, (     )
```

Dat wil zeggen, laad het registerpaar hl met de inhoud van twee opeenvolgende adressen. De betreffende adressen worden door de tweede en derde byte aangegeven door middel van de berekening;

```
lager adres = eerste getal + 256* tweede getal
hoger adres = lager adres + 1
```

of, in dit geval:

```
lager adres = 75 + 256* 92 = 23627
hoger adres = 23627 + 1 = 23628
```

Het woord laad is alleen maar een andere manier om – kopieer – te zeggen en h en l kunnen we als twee speciale locaties binnen de Z80A zien, die worden gebruikt om getallen te bevatten. De hele instructie betekent dus: kopieer de inhoud van 23627 in register l, en de inhoud van 23628 in register h. Merk op dat het *l*ager adres de bron van l is, en het *h*oger adres die van *h*.

3.1 Bits

Een *bit* is de fundamentele eenheid van het computergeheugen omdat het alleen in twee toestanden kan bestaan. We kunnen ons die twee toestanden voorstellen als AAN of UIT; WAAR of VALS; JA of NEE; BOVEN of BENEDEN; MAN of VROUW of welk paar dan ook van logisch tegengestelde toestanden. De mechaniek waarmee een computergeheugen werkt is voor ons niet werkelijk belangrijk, maar in de Spectrum wordt de toestand van een bit aangegeven door een

microscopische vaste (solid-state) schakelaar, al naar gelang dat nodig is AAN of UIT te zetten.
De gebruikelijke notatie is om zo'n toestand voor te stellen als NUL-toestand, en de andere als ÉÉN-toestand. Men zegt dat een bit wordt 'ge-set' als het in de ÉÉN-toestand is, en anders 'ge-reset'. Deze notatie stelt ons in staat om van een bit-patroon in termen van zijn binaire equivalent te praten, en door elk binair getal in zijn decimale vorm om te zetten, kunnen we elk bit-patroon een uniek geheel positief decimaal getal toekennen.
Bekijk bijvoorbeeld 8 bits waarvan de vier meest rechtse zijn ge-set en de meest linkse ge-reset. Zo'n bit-patroon wordt in tabel 3.1 gegeven.

Tabel 3.1: Een groep van 8 bits waarvan de vier meest linkse ge-reset zijn, en de vier meest rechtse ge-set

Schakelaar	Uit	Uit	Uit	Uit	Aan	Aan	Aan	Aan
Toestand	Ge-reset	Ge-reset	Ge-reset	Ge-reset	Ge-set	Ge-set	Ge-set	Ge-set
Binair patroon	0	0	0	0	1	1	1	1
Bit-nummer	7	6	5	4	3	2	1	0

Het binaire patroon kan decimaal worden omgezet als we ons herinneren dat, van een binair getal, de meest rechtse kolom de eenheden aangeeft, de volgende naar links is de twee-kolom, opnieuw de volgende naar links is de vier-kolom enzovoort, elke keer naar links het dubbele nemen. Het decimale equivalent van 00001111 is dus:

$$0*128 + 0*64 + 0*32 + 0*16 + 1*8 + 1*4 + 1*2 + 1*1 = 15$$

omdat er een 1 in de één-, twee-, vier-, en acht-kolommen zit, een 0 in de rest.
Het is zeer onhandig over bits te praten als 'de meest rechtse' of als 'de tweede van links' en dus werd de afspraak geaccepteerd om de bits vanaf rechts te noemen, beginnend bij nul. Het is niet zomaar toevallig dat, wanneer deze afspraak wordt gebruikt, het bit-nummer ook de macht is tot welke 2 moet worden verheven om de kolomwaarde te krijgen. Dat wil zeggen:

$$2^{\text{bitnummer}} = \text{kolomwaarde}$$

Bit 3, bijvoorbeeld, komt voor in de acht-kolom en $2^3 = 8$.

3.2 Bytes

De Z80A-microprocessor die in het hart van de Spectrum zit, werkt iedere keer met acht bits tegelijk. (Het woord 'werkt' omvat alle verschillende taken die in de instructieset werden ingebouwd, zoals optellen, aftrekken, rotatie, logische AND (EN), enzovoort. De vorm van deze instructies wordt later in dit hoofdstuk uitgelegd.) Dus ondanks dat een bit de fundamentele eenheid is van een computergeheugen, worden ze gewoonlijk samen in groepen van acht gemanipuleerd. Een groep van acht bits wordt *byte* genoemd (als bait uitspreken).
Iedere byte in RAM kan worden gebruikt om een enkel positief geheel getal tus-

sen 0 t/m 255 te bevatten, door het set-ten of reset-ten van de acht bits in de byte, overeenkomstig het binaire equivalent van het getal. De byte in tabel 3.1, bijvoorbeeld, bevat het geval 15.
Er zijn 16384 bytes in het ROM-geheugen van de Spectrum en het is de inhoud van deze bytes samen met de electronische organisatie wat de computer zijn 'karakter' geeft. De bytes werden vastgelegd toen de computer werd gemaakt en kunnen later niet worden veranderd. Dat is dus de reden waarom het ROM-geheugen – alleen leesbaar geheugen – wordt genoemd; de inhoud ervan kan worden gelezen, maar het kan niet worden overschreven. De niet-uitgebreide Spectrum bevat nog 16384 bytes van het willekeurig toegankelijke geheugen. Het woord willekeurig is eigenlijk een verkeerde benaming. Het betekent niet dat het geheugen op goed geluk wordt gebruikt; het betekent eerder dat elke byte op elk moment, direct kan worden bereikt. Dit in tegenstelling tot een sequentieel toegankelijk geheugen zoals bij een cassetterecorder waarbij het noodzakelijk is langs het hele geheugen te gaan totdat het gezochte gedeelte wordt bereikt.
Voor de niet-ingewijden lijkt 16384 geen gemakkelijk getal voor het te gebruiken aantal bytes. In feite is het een erg geschikt aantal, omdat $2^{14} = 16384$ (dwz. 16384 is gelijk aan 2, veertien maal met zichzelf vermenigvuldigd). In de computerwereld zijn machten van twee 'ronde getallen', net zoals de machten van tien -honderdtallen, -duizendtallen, -miljoenen, 'ronde getallen' zijn. Een specifiek 'rond getal' is 1024 dat 2 tot de macht 10 is. 1024 ligt dicht genoeg bij het getal duizend om het gebruik van de letter K te verantwoorden om het weer te geven. (K wordt in het metrieke systeem gebruikt als kilogram -kg, kilometer -km, enz.) 1024 wordt dus als 1K geschreven en 16384, wat gelijk is aan 16*1024, wordt als 16K geschreven.

3.3 Adressen

Een computer moet in staat zijn elke locatie in zijn geheugen te identificeren zodat het van en naar de juiste locatie kan kopiëren. Vandaar dat iedere locatie een uniek *adres* krijgt. Een adres is een positief geheel getal, groter dan of gelijk aan nul.
Veel van de Z80A-instructies hebben de vorm 'kopieer de inhoud van het volgende adres naar dit-of-dat register of registerpaar'. De instructie:

```
ld hl, (23627)
```

die aan het begin van dit hoofdstuk werd beschreven, heeft deze vorm. Het adres achter de instructie wordt in twee bytes opgeslagen en dus kan het aantal adressen dat de microprocessor kan bereiken, in twee bytes worden gevat. Dit aantal is hetzelfde als het aantal verschillende bit-patronen dat door de 16 bits kan worden aangenomen die de twee-adres-bytes vormen: $2^{16} = 65536$. Een twee-adres-byte wordt geïnterpreteerd in de vorm:

```
adres = eerste byte + 256*tweede byte.
```

Soms worden de twee bytes respectievelijk de lage en de hoge byte genoemd. De twee-byte-vorm van 16384 bijvoorbeeld, (het begin van de RAM in de Spectrum), is: lage byte = 0; hoge byte = 64 omdat:

```
0 + 256*64 = 16384.
```

3.4 De Z80A-registers

Een computer verandert niet rechtstreeks de inhoud van het geheugen wanneer het bezig is een programma uit te voeren, maar hij kopieert de inhoud van een locatie in het geheugen naar een *register* en werkt met de inhoud van dit register. Registers hebben, in machinetaal, een soortgelijke functie als de variabelen in BASIC, in die zin dat ze gebruikt worden om er getallen in op te slaan en omdat ze kunnen worden gebruikt om beslissingen te controleren. Ze verschillen van de BASIC-variabelen omdat ze beperkt in aantal zijn en ze binnen de microprocessor zelf bestaan en niet in RAM. Ze bevatten slechts één byte, of twee in het geval van een registerpaar.
De Z80A is een krachtige microprocessor omdat het verscheidene registers heeft en dus verscheidene getallen tegelijk kan bevatten; dankzij dat vermindert het de tijdrovende overdrachten tussen de processor en het geheugen. De meeste registers hebben één of meerdere speciale kenmerken.

Het accumulatorregister-a

De accumulator is het belangrijkste register, omdat de meeste rekenkundige instructies zoals optellen bijvoorbeeld, en de logische instructies zoals de logische OR (OF), op de inhoud van dit register werken. In feite verdient het deze naam omdat het resultaat van verschillende opeenvolgende instructies zich in het a-register verzamelen.
Sommige van de instructies die te maken hebben met de accumulator gebruiken een tweede register of een geheugenadres als bron van gegevens. Bijvoorbeeld, add a, b, vertelt de processor de inhoud van register b bij register a op te tellen, en het resultaat in a op te slaan.

Het vlaggenregister-f

De meeste registers zijn dubbel, in die zin dat sommige instructies op twee registers tegelijk werken. Het *f*-register wordt in dit opzicht aan het *a*-register gekoppeld, alhoewel het verband nogal vaag is, omdat het beperkt is tot de push-, pop- en exchange-instructies.
Het f-register verschilt nogal van de andere omdat de acht bits in het register als zogenaamde vlaggen (flags) worden gebruikt om de volgorde van de programma-uitvoering te registreren en te controleren. Iedere vlag wordt gebruikt om aan te geven dat één van de twee tegengestelde, logische gebeurtenissen heeft plaatsgevonden; de nul-vlag bijvoorbeeld, geeft aan wanneer het resultaat van de laatste optelling, aftrekking enzovoort, nul was. Slechts vier van de acht vlaggen zijn van belang voor de meeste gebruikers. Hun kenmerken worden in tabel 3.2 opgesomd.
De *tekenvlag* is de simpelste. Volgens afspraak wordt, bij gebruik van een byte om een getal met teken weer te geven, bit 7 ge-set om het teken te bevatten; het wordt dus ge-set als het getal negatief is en anders ge-reset. De tekenvlag geeft het teken weer van het laatste resultaat.
De *nulvlag* wordt ge-set als het resultaat van de laatste bewerking nul is. Het wordt ook bij vergelijkingen gebruikt, die in feite aftrekinstructies zijn waarvan het resultaat opzij wordt gezet.
De *carry-vlag* wordt ge-set om de overflow (overloop) te registreren die plaatsvindt wanneer het resultaat van een optelling te groot is om in het register te worden opgeslagen en als er een 'lening' (borrow) bij een aftrekking plaatsvindt. Er zijn ook enkele roteerinstructies waarbij de bits van register a naar links of naar rechts worden geroteerd en bit 7 of 0 van of naar de carry-vlag wordt geroteerd.

Tabel 3.2 De vier vlaggen die de meeste bewerkingen van de Z80A controleren

Vlag	*Mnemonic*	*Mnemonic ge-reset*	*Gebruik*
Teken	M	P	Ge-set als laatste resultaat negatief is.
Nul	Z	NZ	Ge-set als laatste resultaat nul is of een evenbeeld wordt gevonden.
Carry	C	NC	Ge-set als laatste resultaat te groot was om het in één byte op te slaan (of twee bytes bij een registerpaar).
Parity/ Overflow	PE	PO	Parity: ge-set als laatste resultaat de even-pariteit had. Overflow: ge-set als een bewerking bit 7 verandert als resultaat van een overflow van andere bits.

De *Parity/Overflow-vlag* is in werkelijkheid twee vlaggen in één. Het werkt als een overflow-vlag bij een rekenkundige instructie, gebruikt om aan te geven of bit 7 was beïnvloed door een carry of een 'lening' (borrow) die door bit zes was gegenereerd. Het wordt daarom gebruikt om te controleren of het tekenbit was verminkt. Logische instructies gebruiken dezelfde vlag om de pariteit van het resultaat weer te geven. (De pariteit van een binair getal is het aantal bits dat op één wordt ge-set. Als dit getal even is, zegt men dat de pariteit even is; als het oneven is, dan is de pariteit oneven.) De vlag wordt ge-set als de pariteit van een resultaat even is.
Het effect van bepaalde instructies hangt af van de actuele stand van specifieke vlaggen. Bijvoorbeeld, de instructie:

```
jr z, d
```

heeft als gevolg dat de Z80A over de volgende d instructies springt als de nulvlag ge-set is. Als de nulvlag niet is ge-set voert de processor de volgende instructie in de gewone volgorde uit. Het vlaggenregister is dus belangrijk omdat het de processor in staat stelt beslissingen te nemen en ook naar een ander gedeelte van het programma te springen.

De telregisters — b en c

Het register b, en in zekere mate register c waaraan het is gekoppeld, is beschikbaar voor een aantal doeleinden, maar zijn belangrijkste rol is dat van een teller. We hebben al gezien hoe het verloop van een programma door het gebruik van de nulvlag in de instructie jr z, d kan worden gecontroleerd. Een andere instructie:

```
djnz d
```

gebruikt ook de nulvlag om de opbouw van lussen in machinetaal mogelijk te maken door b als teller te gebruiken op dezelfde manier als de FOR-NEXT-lussen in BASIC.
Als de instructie wordt gevonden, vermindert de Z80A de inhoud van het b-register, dat wil zeggen het wordt met één verlaagd. Als het resultaat nul is, wordt de volgende instructie in de reeks uitgevoerd. Als het resultaat niet nul is springt de

routine d instructies. Als de programmeur een negatieve waarde voor d gebruikt, dan is het een sprong terug naar een eerder gedeelte in het programma en als we aannemen dat er geen andere vertakkingen zijn dan komt de processor dezelfde instructie nog eens tegen. Dus door register b oorspronkelijk met een geschikte waarde te laden, en de verplaatsing d, op de juiste plaats te zetten, kan een gedeelte van het programma een gegeven aantal keer worden uitgevoerd.
Het b-register bevat slechts één byte en daarom kan het alleen worden gevuld met getallen tussen 0 en 255 maximaal. Dus door gebruik van dit mechanisme kan een gedeelte van een programma maximaal 255 keer worden doorlopen.
Op deze wijze zijn er verder geen mogelijkheden om een lus meer dan 255 keer te doorlopen, maar er zijn een aantal krachtige instructies die alle 16 bits van het registerpaar bc als een teller tot 65535 gebruiken. Een voorbeeld ervan is de instructie:

```
cpdr
```

Als de Z80A deze instructie tegenkomt dan zal hij:
1) bc met één verlagen;
2) de inhoud van hl verlagen (hl is een ander registerpaar), zie onderaan;
3) de inhoud van de accumulator, a, vergelijken met de inhoud van de locatie waarvan het adres in hl te vinden is.

De processor herhaalt deze acties totdat een gelijkenis wordt gevonden tussen a en de inhoud van het geheugen of totdat bc=0. Dus deze instructie kan worden gebruikt om het geheugen te doorzoeken naar een adres met een bepaald getal.

De adresregisters – de en hl
De registers d en e hebben geen individuele functie en worden meestal als tijdelijk, snel toegankelijk geheugen gebruikt. Ze mogen ook samen worden gebruikt om het adres van een locatie te bevatten dat op dat moment van belang is.
De belangrijkste functie van de h- en l-registers is samen het adres van een locatie in het geheugen op te slaan, en we hebben al gezien hoe bepaalde krachtige instructies voor dit doel gebruik maken van hl. h staat voor hoog-byte en l staat voor laag-byte en het adres wordt als volgt opgeslagen:

```
adres = 256*h + l
```

wat een maximum van 65536 unieke adressen oplevert (dwz. 0 t/m 65535).

De indexregisters – ix en iy
De registers ix en iy zijn 16-bit registers en kunnen alleen als zodanig worden gebruikt, in tegenstelling tot de bc-, de- en hl-registers die we al kennen; deze kunnen 2 aan 2 als 16-bit registers worden gebruikt, of als enkele 8-bit registers.
De ix- en iy-registers worden op dezelfde manier als het hl-registerpaar gebruikt ondanks dat de instructies die ze besturen één byte meer nodig hebben in vergelijking met de gelijkwaardige hl-instructies. Bijvoorbeeld:

```
add hl, bc
```

is een één-byte instructie die de Z80A opdracht geeft de inhoud van hl en bc op te tellen en het resultaat in hl op te slaan. Dezelfde instructie met ix:

```
add ix, bc
```

is een twee-byte instructie. ix en iy hebben nog een eigenschap die niet aanwezig is bij hl en dat is dat ze gebruikt kunnen worden met een verplaatsing d. Dit betekent dat een instructie die aan (ix + d) refereert, niet de locatie in het geheugen gebruikt waarvan het adres in ix zit, maar d wordt bij de waarde van ix opgeteld om een nieuw adres te krijgen en de instructie gebruikt dan de betreffende geheugenlocatie. Het is juist deze eigenschap waarvan de naam indexregisters wordt afgeleid.

De stapelwijzer — sp

De stapel is een gebied aan of vlakbij de top van de RAM die gebruikt wordt voor tijdelijke opslag van de inhoud van registerparen. Het is ontworpen om binnen de RAM 'omlaag' te groeien als het wordt gevuld, en om daar te slinken als het leeg wordt gemaakt. De bodem van de stapel is vast en het ligt in de Spectrum, onmiddellijk onder de locatie die door de systeemvariabele RAMTOP wordt aangegeven. De top van de stapel ligt onder de bodem van de stapel omdat het omlaag groeit en omhoog slinkt. Het adres van de actuele locatie van de top wordt in het sp-register opgeslagen.
Overdracht van gegevens van en naar de stapel gebeuren met behulp van de instructies push en pop. Bijvoorbeeld:

```
push hl
```

heeft als gevolg dat de processor:

1) sp verlaagt;
2) de inhoud van h naar de door sp (Engels: stack pointer) aangegeven locatie kopieert;
3) sp verlaagt;
4) de inhoud van l naar de door sp aangegeven locatie kopieert.

De pop-instructie doet precies het tegenovergestelde. Op deze manier is altijd het meest recente paar van waardes dat naar de stapel ge-push-ed werd, het eerste dat er vandaan ge-pop-ped wordt. Dit voorziet in een simpele maar handige methode om de inhoud van registers tijdelijk op te slaan, misschien tijdens het aandoen van een subroutine. Mits de registerparen in de omgekeerde volgorde ge-pop-ped worden van die waarin ze oorspronkelijk werden ge-push-ed, zullen er geen problemen zijn.

De programmateller — pc

De programmateller pc (Engels: program counter) is een zeer belangrijk 16-bit register, want het bevat het geheugenadres van de volgende uit te voeren instructie.
Het normale verloop van gebeurtenissen wanneer een instructie wordt uitgevoerd is als volgt:

1) Kopieer de inhoud van de door pc aangegeven locatie in een speciaal register binnen de microprocessor.
2) Als de instructie in meerdere bytes wordt opgeslagen, verhoog dan pc en kopieer de inhoud van de volgende locatie in een tweede speciaal register.
3) Verhoog pc zodanig dat het naar de volgende uit te voeren instructie wijst.
4) Voer de instructie uit die net gekopieerd werd.

Een sprong-instructie als djnz d, of jr z, d, verandert het normale verloop van gebeurtenissen door pc tijdens stap 4 te veranderen. Merk op dat deze verandering gebeurt nadat pc is verhoogd. Dus de waarde van de verplaatsing, d, zou altijd moeten worden berekend ten opzichte van de positie van de instructie die volgt op de instructie waarin de verplaatsing staat.

De alternatieve registers – af', bc', de', hl'

De Z80A bezit duplicaten van elk van de registers a, b, c, d, e, h, en l. De duplicaten worden onderscheiden door het gebruik van een apostrof ('); bijvoorbeeld, a' is het duplicaat van register a. De instructies werken niet rechtstreeks op deze duplicaten, maar de wisselinstructies zijn beschikbaar om twee of meer registers buiten gebruik te stellen en om de duplicaten ervan op hun plaatsen terug te brengen.

De wisselinstructies worden zeer snel uitgevoerd, sneller dan bijvoorbeeld push- en pop-instructies. De inhoud wordt niet fysisch van het ene register naar het andere gekopieerd. Eerder nog, er worden een aantal interne schakelaars veranderd zodat het alternatieve register door de volgende instructies wordt gebruikt en het oorspronkelijke register ongebruikt blijft.

3.5 Over de instructieset

Er zijn meer dan 600 elementen in de Z80A-instructieset zoals uit appendix A blijkt. Aangezien er slechts 256 verschillende rangschikkingen van 8 bits mogelijk zijn (omdat $2^8 = 256$), kan minder dan de helft van de instructies in één byte worden opgeslagen. De rest van de instructies worden in twee of zelfs in drie bytes opgeslagen. De eerste byte van een twee-byte instructie is of 203, 221, 237, of 253 (CB, DD, ED of FD in hexadecimaal). De eerste twee bytes van een drie-byte instructie zijn of 221, 203, of 253, 203 (DD, CB of FD, CB in hexadecimaal). Sommige instructies worden door een één-byte verplaatsings, d, gevolgd, of door een één-byte nummer, n, of door een twee-byte nummer of adres, nn, waarmee deze te maken heeft. Op deze manier kan een enkele instructie in totaal hoogstens vier bytes in beslag nemen. Bijvoorbeeld, de instructie:

```
jr nz, d
```

die we al kennen, heeft één byte nodig om de instructie zelf op te slaan (32 decimaal, 20 hexadecimaal) en een tweede byte voor de verplaatsing, d.

In dit hoofdstuk worden alle instructies vermeld met behulp van hun assembly-taal-mnemonic of opcode (operatie code). De mnemonics vormen een verkorte manier om iedere instructie te beschrijven en zijn alleen voor de mens bruikbaar. De Spectrum zal geen mnemonic herkennen tenzij door middel van een assembler-programma. De volgende afspraken worden in acht genomen:

1) Verwijzingen naar één register gebeuren aan de hand van hun eigen letter, bijvoorbeeld b. Registerparen worden alfabetisch genoemd, bijvoorbeeld bc.
2) Een verplaatsing d, wordt als positief beschouwd als het tussen 0 en 127 ligt, en negatief tussen 128 en 255. Grotere of kleinere getallen worden niet toegestaan. De negatieve waarde wordt berekend door 0 van 256 af te trekken. Bijvoorbeeld, de onvoorwaardelijke spronginstructie:

   ```
   jr d
   ```

   veroorzaakt een sprong van 8 bytes voorwaarts als d = 8 en een sprong

achterwaarts als d = 248 (= 256−8). Onthoud dat bij de berekening van een verplaatsing een sprong wordt gemaakt vanaf het adres van de eerste byte achter de instructie.

3) Een één-byte getal, n, ligt tussen 0 t/m 255.
4) Een twee-byte getal of een adres, wordt door nn aangegeven en ligt in het bereik 0 t/m 65535. De waarde ervan wordt berekend door bij de eerste n, 256 maal de tweede n op te tellen.
5) nn tussen haakjes, (nn), betekent 'de inhoud van de locatie op adres nn', terwijl nn zonder haakjes betekent 'het getal nn'. Dus:

```
ld hl, (23627)
```

betekent: laad het hl-registerpaar met de inhoud van de locatie 23627 en 23628, terwijl:

```
ld hl, 23627
```

betekent: laad hl met het getal 23627. Zodoende betekent (hl) 'de inhoud van de locatie waarvan het adres in hl zit', terwijl hl zonder haakjes betekent: 'het getal in hl'.

6) De bestemming van het resultaat van een instructie wordt altijd als eerste aangegeven. Bijvoorbeeld:

```
add a, b
```

betekent: 'Tel de inhoud van b bij a op en zet het resultaat in a'.

3.6 Verklarende woordenlijst van machinetaalinstructies

Dit gedeelte biedt u een samenvatting van het grootste deel van de Z80A-instructieset. Enkele van de meer gespecialiseerde instructies die met interrupts te maken hebben, zijn weggelaten.

Geen bewerking nop

Dit is de simpelste instructie, en zoals de naam zegt, de processor doet niets (Engels: no operation) als hij het tegenkomt. Het kan erg handig zijn als we bezig zijn de fouten uit een routine te verwijderen, omdat het tijdelijk door een toepasselijke instructie kan worden vervangen, zonder de werking van de rest van de routine te veranderen. Het kan ook worden gebruikt voor het vullen van gaten ontstaan bij het veranderen van bestaande programma's of om een vertraging in de uitvoering te veroorzaken met name als het in een juiste lus wordt ingebracht. De decimale code ervan is 0.

Laden ld

Laadinstructies (Engels: load) worden gebruikt om één of twee bytes tussen twee registers of tussen registers en het geheugen te verplaatsen. Er zijn meer dan honderd laadinstructies, en dat is meer dan elke andere soort. Ze worden in acht groepen onderverdeeld:

1) 8-bit register naar register.
   De inhoud van elk van de registers a, b, c, d, e, h of l kan naar elk van de anderen worden gekopieerd.

2) 8-bit geheugen naar register.
(hl), (ix + d) of (iy + d) kunnen naar elk van de registers a, b, c, d, e, h of l worden gekopieerd. (bc), (de) of (nn) kunnen naar a worden gekopieerd.

3) 8-bit register naar geheugen.
a, b, c, d, e, h of l kunnen worden gekopieerd naar (hl), (ix + d) of (iy + d). a kan naar (bc), (de) of (nn) worden gekopieerd.

4) 8-bit register naar onmiddellijk (immediate) geheugen.
'Onmiddellijk' is de naam voor een getal, vanaf het programma zelf ingelezen, eerder dan vanuit een register of vanuit een ander adres in het geheugen. Een getal n, kan in a, b, c, d, e, h, l, (hl), (ix + d) of (iy + d) worden geladen.

5) 16-bit register naar register.
De inhoud van hl, ix of iy kan naar sp worden gekopieerd.

6) 16-bit geheugen naar register.
(nn) kan naar bc, de, hl, ix, iy of sp worden gekopieerd.

7) 16-bit register naar geheugen.
bc, de, hl, ix, iy of sp kan naar (nn) worden gekopieerd.

8) 16-bit register onmiddellijk.
nn kan naar bc, de, hl, ix, iy of sp worden gekopieerd.

Push en pop push, pop

Een push-instructie kopieert de inhoud van een zogenaamd 16-bit register naar de stapel en verlaagt de teller met twee. Een pop-instructie doet het tegenovergestelde zodat beide instructies kunnen worden gebruikt om de waarde van registers te bewaren en deze later in het programma opnieuw te laden. Op elk van de registerparen af, bc, de, hl, ix en iy kan een push- of pop-instructie worden uitgevoerd.

Verwisselen ex

Verwisselingen (Engels: exchange) kunnen tussen hl en de, hl en (sp), ix en (sp), iy en (sp), af en af', en tussen bcdehl en bcdehl' worden gemaakt (een enkele instructie switcht alle zes 8-bit registers om).

8-Bit optellen en aftrekken add, sub

a, b, c, d, e, h, l, (hl), n, (ix + d) en (iy + d) kunnen worden opgeteld of afgetrokken (Engels: add en subtract) bij of van het register a, met of zonder de carry-vlag. Instructies die betrekking hebben op de carry-vlag eindigen met een c.

8-Bit and, or en xor and, or, xor

a, b, c, d, e, h, l, (hl), n, (ix + d) en (iy + d) kunnen worden gecombineerd met het a-register door middel van één van de drie logische operatoren. And 'set' in het resultaat iedere bit die in beide bronnen (source) was ge-set; Or 'set' iedere bit die in een of beide bronnen was ge-set; en xor 'set' iedere bit die in de ene of de andere bron was ge-set maar niet die welke in beide was ge-set.

Vergelijken cp

Compare is net als subtract met het verschil dat alleen de vlaggen worden beïn-

vloed en niet de inhoud van a. a, b, c, d, e, h, l, (hl), n, (ix + d) en (iy + d) kunnen met de accumulator worden vergeleken (Engels: compare).

8-Bit verhogen en verlagen inc, dec

a, b, c, d, e, h, l, (hl), (ix + d) en (iy + d) kunnen worden verhoogd of verlaagd (Engels: increment en decrement).

16-Bit verhogen en verlagen inc, dec

bc, de, hl, ix, iy en sp kunnen worden verhoogd of verlaagd.

16-Bit optellen en aftrekken add, sub

bc, de, hl, ix kunnen met of zonder carry bij hl worden opgeteld, of alleen met carry van hl worden afgetrokken. bc, de, sp, ix kunnen zonder carry bij ix worden opgeteld. bc, de, sp en iy kunnen zonder carry bij iy worden opgeteld.

Jump, call en return

Het vlaggenregister, f, heeft een carry-vlag, c; een parity-vlag p, die ge-set wordt als het resultaat een even parity heeft; een tekenvlag, die ge-set wordt als een resultaat negatief is; een overflow-vlag, v, die bij overflow wordt ge-set; en een zero-vlag, z, die bij een nul-resultaat wordt ge-set. Deze vlaggen kunnen worden gebruikt om jumps, subroutine-calls en subroutine-returns te controleren.

1) Jump jp of jr

   De volgende sprongen (Engels: jump) naar adres nn zijn mogelijk: absolute sprong (jp); sprong bij nul of niet nul (jp z) en (jp nz); sprong bij carry of geen carry (jp c en jp nc); sprong bij positief of bij negatief (jp p en jp m); sprong bij p/v = 1 of p/v = 0 (jp pe en jp po).

   De volgende relatieve sprongen naar een adres d, relatief aan de actuele positie, zijn mogelijk, waarbij d geïnterpreteerd wordt als liggend tussen −128 tot 127: absolute relatieve sprong (jr); relatieve sprong bij nul of niet nul (jr z en jr nz); relatieve sprong bij carry of geen carry (jr c en jr nc).

   Sprongen kunnen ook naar de inhoud van registers hl, ix of iy worden gemaakt (jp (hl), jp (ix), jp (iy)). De djnz-instructie verlaagt het b-register en springt naar d als b niet nul is.

2) Call call

   Deze instructie heeft een gelijkwaardige functie als de BASIC-opdracht GOSUB. Als er aan de voorwaarde voldaan wordt, gaat het programma naar de instructie op adres nn. De volgende calls mogen worden gebruikt: absolute call (call); call bij nul of bij niet nul (call z en call nz); call bij carry of bij geen carry (call c en call nc); call bij positief of bij negatief (call p en call m); call bij p/v = 1 of call bij p/v = 0 (call pe of call po).

3) Return ret

   Deze instructie heeft een gelijkwaardige functie als de BASIC-opdracht RETURN. Er zijn voorwaarden beschikbaar om aan alle call-voorwaarden te voldoen; returns kunnen ook vanaf de interrupt en de non-maskable interrupt worden gemaakt (reti en retn).

Bit-instructies

De acht bits in elk register worden van rechts naar links, van 0 tot 7 genummerd. Elk van de volgende bewerkingen kan op de registers a, b, c, d, e, h, l, en bij (hl), (ix + d) en (iy + d) worden toegepast:

1) Bit test bit
De bit-testinstructie 'set' de nulvlag op het tegenovergestelde van de stand van de geteste bit. Elke bit kan worden getest.
2) Bit set set
Elke bit kan worden ge-set.
3) Bit reset res
Elke bit kan worden ge-reset.
4) Rotate left (links roteren) rl
Bit 7 wordt naar de carry gekopieerd, de carry naar bit 0 en de rest van de bits wordt een plaats naar links gekopieerd.
5) Rotate right (rechts roteren) rr
Bit 0 wordt naar de carry gekopieerd, de carry naar bit 7 en de rest van de bits wordt een plaats naar rechts gekopieerd.
6) Rotate left circular (cirkelvormig links roteren) rlc
Bit 7 wordt naar de carry en naar bit 0 gekopieerd. De rest van de bits wordt een plaats naar links gekopieerd.
7) Rotate right circular (cirkelvormig rechts roteren) rrc
Bit 0 wordt naar de carry en naar bit 7 gekopieerd. De rest van de bits wordt een plaats naar rechts gekopieerd.
8) Shift left arithmetic (rekenkundige verschuiving links) sla
Alle bits worden één plaats naar links gekopieerd, bit 7 naar de carry, en bit nul wordt ge-reset.
9) Shift right arithmetic (rekenkundige verschuiving rechts) sra
Alle bits worden één plaats naar rechts gekopieerd, bit 0 naar de carry en bit 7 naar zichzelf.
10) Shift right logical (logische verschuiving rechts) srl
Zoals shift right arithmetic, maar bit 7 wordt ge-reset.

Roteer links decimaal rld
Bit 0 tot 3 van A worden naar bit 0 tot 3 van (hl) gekopieerd; bit 0 tot 3 van (hl) worden naar bit 4 tot 7 van (hl) gekopieerd; bit 4 tot 7 van (hl) worden naar bit 0 tot 3 van a gekopieerd.

Roteer rechts decimaal rrd
Bit 0 tot 3 van a worden naar bit 4 tot 7 van (hl) gekopieerd; bit 4 tot 7 van (hl) worden naar bit 0 tot 3 van (hl) gekopieerd; bit 0 tot 3 van (hl) worden naar bit 0 tot 3 van a gekopieerd.

Bewerkingen met de accumulator
1) Complement a (complementeer a) cpl
Elke ge-set bit wordt ge-reset, en andersom.
2) Negate a (trek de accumulator af van nul) neg
Complementeer a en tel er 1 bij op.
3) Complement carry (complementeer de carry-vlag) ccf
De carry-vlag wordt ge-reset als het ge-set was, en andersom.
4) Set Carry scf
De carry-vlag wordt ge-set.
5) Decimal adjust daa
Corrigeer a na een bcd-optelling of aftrekking.

Restart rst

Sla de programmateller op in de stapel en spring naar locatie 8*n, waarbij n in de volgende byte te vinden is.

Blokmanipulaties

Deze samengestelde instructies zijn ontworpen om data in het geheugen te verplaatsen of om naar data in het geheugen te zoeken.

1) Load and increment (laad en verhoog) ldi
   Verplaats byte van (hl) naar (de); verhoog hl en de en verlaag bc.
2) Load, increment and repeat (herhaald blok laden met verhogen) ldir
   Verplaats een byte van (hl) naar (de); verhoog hl en de en verlaag bc. Herhaal tot bc = 0.
3) Load and decrement (laad en verlaag) ldd
   Verplaats een byte van (hl) naar (de) en verlaag hl, de en bc.
4) Load, decrement and repeat (herhaald blok laden met verlagen) lddr
   Verplaats een byte van (hl) naar (de) en verlaag hl, de en bc. Herhaal tot bc = 0.
5) Compare and increment (vergelijk en verhoog) cpi
   Vergelijk a en (hl). Verhoog hl en verlaag bc.
6) Compare, increment and repeat (herhaald blok vergelijken met verhogen) cpir
   Vergelijk a en (hl). Verhoog hl en verlaag bc. Herhaal tot a = (hl) of bc = 0.
7) Compare and decrement (vergelijk en verlaag) cpd
   Vergelijk a en (hl). Verlaag hl en bc.
8) Compare, decrement and repeat (herhaald blok vergelijken met verlagen) cpdr
   Vergelijk a en (hl). Verlaag hl en bc. Herhaal tot a = (hl) of bc = 0.

DEEL B

4. Inleiding

De 40 machinetaalroutines uit deel B worden, om het gebruik daarvan te vergemakkelijken, in een standaardformaat uitgeprint. Deze inleiding legt het formaat uit en introduceert een BASIC-programma dat gebruikt kan worden om de routines in het geheugen te laden.

Lengte:
Dit is de lengte van de routine in bytes.

Aantal variabelen:
De uitvoering van sommige routines kan worden gecontroleerd door de waarde van één of meerdere variabelen aan te passen die via de printer-buffer aan de routine werden meegegeven.

Check sum:
Elke routine wordt als een reeks positieve gehele getallen aangeboden, die in opeenvolgende geheugenlocaties gePOKEd moet worden. De check sum (dwz. de som van alle getallen die de routine vormen) wordt zodanig aangeboden, dat de lezer er zeker van kan zijn dat hij de routine correct geladen heeft.

Werking:
Er wordt een korte uitleg gegeven van de door de routine uit te voeren taak.

Variabelen:
De naam, lengte en locatie van elke variabele in de printer-buffer worden gedefinieerd. Een variabele die één byte lang is, moet een positief geheel getal zijn tussen 0 t/m 255 en wordt vanuit BASIC of via het toetsenbord doorgegeven door middel van:

POKE *locatie, waarde.*

Een twee-byte variabele wordt via twee opdrachten doorgegeven:

POKE *locatie, waarde* 256*INT(waarde/256).
POKE *locatie+1, INT(waarde*/256).

De gebruikte locaties zitten in de printer-buffer.

Call:
De routines worden aangeroepen via de USR-functie die deel uit moet maken van een opdracht. Als de machinetaalroutine geen waarde aan BASIC teruggeeft, wordt de RAND-opdracht aangeraden zoals in:

RAND USR adres

Als de waarde in het bc-registerpaar terug moet komen, wordt of:

LET A=USR adres

of

PRINT USR adres

aangeraden, afhankelijk van of de teruggegeven waarde in een BASIC-variabele moet worden opgeslagen of op het scherm moet worden gePRINT.

Foutafhandeling:
De door de routine gemaakte controles voor onlogische of tegenstrijdige waardes enzovoort, worden uitgelegd.

Commentaar:
Eenvoudige varianten van de hoofdroutines worden uitgelegd.

Machinetaal-listing:
De routine wordt aangeboden in assembly-taal met in de derde kolom, 'In te toetsen getallen' genoemd, de absolute vorm ervan. Om de routine in het geheugen te laden, worden de getallen uit de derde kolom als een reeks in het geheugen gePOKEd. Het zijn allemaal decimale getallen.

Hoe het werkt:
De bewerkingsvorm van de routine wordt aan de hand van de machinetaal-listing uitgelegd.

4.1 Machinetaallader

Bijna alle machinetaalroutines in dit boek zijn verplaatsbaar (Engels: relocatable) wat betekent dat ze correct zullen werken onafhankelijk van waar ze in de RAM zitten. Als een routine niet verplaatsbaar is, wordt in de paragraaf 'commentaar' uitgelegd hoe het veranderd kan worden als het op een andere plaats opgeslagen moet worden dan oorspronkelijk bedoeld was.
We hebben in deel A, hoofdstuk 2, gezien dat de Spectrum verschillende gedeeltes van de RAM voor verschillende doelen gebruikt en dat het gebied tussen de systeemvariabelen RAMTOP en UDG bedoeld is voor het opslaan van machinetaalroutines.
Het programma 4.7 kan worden gebruikt om een machinetaalroutine te laden, te veranderen of te verplaatsen. Daarmee kan de gebruiker het RAMTOP-adres veranderen om meer ruimte voor een routine te krijgen, een routine vanuit het toetsenbord in te toetsen, voorwaarts en achterwaarts door de routine heen te stappen om een fout te corrigeren en om een deel in de routine in te voegen of een gedeelte eruit te halen.
Als het programma draait, PRINT hij het laagste adres van waaruit de routine kan worden opgeslagen; dat wil zeggen, één byte meer dan RAMTOP, en ook de beschikbare ruimte tussen dat adres en het einde van de RAM.
In de 16K machine is dat adres 32600 tenzij de gebruiker de systeemvariabele RAMTOP heeft aangepast. Evenzo is bij een 48K machine het laagste adres gewoonlijk 65368.
De 168 bytes op het einde van de RAM worden gewoonlijk voor de 'user defined graphics' (udg) gereserveerd maar het programma stelt de gebruiker in staat dit gebied te overschrijven als hij dat wil. Als alternatief kan hij een nieuw lager

adres kiezen dat het programma dan in het RAMTOP-adres zet door middel van de opdracht CLEAR. Het programma zal geen adres lager dan 27000 accepteren omdat de routine dan binnen de benodigde ruimte voor het programma zelf zou vallen. Het programma vraagt naar een startadres voor de routine. De gebruiker kan dus ruimte voor verschillende routines reserveren en deze dan los van elkaar laden.
Eenmaal de gebruiker de gelegenheid te hebben gegeven zijn keuze te veranderen als hij niet tevreden is, projecteert het programma het hoofdmenu. Afbeelding 4.1 laat de vorm van het display zien wanneer de routine 'Beeldinversie' op locatie 32000 geladen is. De eerste kolom is het adres, de tweede de inhoud van het adres en de derde de check sum. De routine 'Beeldinversie' is 18 bytes lang en zijn check sum is 1613. Het bezet dus locaties 32000 tot 32017 en de check sum voor locatie 32017, dat wil zeggen, de som van de locaties 32000 tot 32017 is 1613.
Wanneer het hoofdmenu wordt gepresenteerd, wordt de aandacht van de gebruiker naar een locatie geleid omdat de decimaal van FLASH voorzien is. Het heet de actuele (Engels: current) locatie en in beginsel is dat het gekozen startadres van de routine. De gebruiker toetst een heel getal in tussen 0 t/m 255, dat het programma in de actuele locatie POKEd en dan wordt het volgende adres de actuele locatie. Zodoende kan een complete routine op zijn plaats worden gePOKEd, het display bijgewerkt en als het nodig is bij elke stap ge-scrolled.
De gebruiker kan kiezen een getal intoetsen of een selectie maken uit één van de in tabel 4.1 opgesomde opties. Deze faciliteiten creëren de mogelijkheid om correcties te maken.

Tabel 4.1 Deze opties zijn beschikbaar bij het werken met machinetaal

Code	*Optie*
b	De actuele locatie één adres achterwaarts verplaatsen.
b getal	De actuele locatie het ingegeven aantal adressen achterwaarts verplaatsen.
f	De actuele locatie één adres voorwaarts verplaatsen.
f getal	De actuele locatie het ingegeven aantal adressen voorwaarts verplaatsen.
i getal	Een ingegeven aantal bytes op 0 vanaf de actuele locatie tussenvoegen.
d getal	Een ingegeven aantal bytes vanaf de actuele locatie verwijderen.
t	Einde programma.

Programma P4.1: Machinetaallader (l is kleine letter 'l')

```
1ØØ  GO SUB 81ØØ
2ØØ  REM***** Bereken beschikbaar geheugen
21Ø  LET min=1+PEEK 2373Ø+256*PEEK 23731
22Ø  LET p=PEEK 23732+256*PEEK 23733
23Ø  LET t=p-min+1
4ØØ  REM*****Haal startadres
41Ø  PRINT "Laagste start = ";min,, "Maximaal
     beschikbare geheugenruimte = ";t
```

```
42Ø  INPUT "Wilt u het laagste startadres
     veranderen (J of N)? ";z$
43Ø  IF z$= "J" OR z$= "j" THEN GO TO 7ØØØ
44Ø  INPUT "Geef startadres van waar de
     machinetaal geladen moet worden";a
45Ø  IF a<min OR a>p THEN BEEP .2,24: GO TO 44Ø
5ØØ  GO SUB 81ØØ
51Ø  LET t=t-a+min
52Ø  PRINT "U kunt tot ";t;" bytes gebruiken",,,
53Ø  LET u = PEEK 23675+256*PEEK 23676
54Ø  IF a<u AND u<p THEN PRINT "Als u meer
     gebruikt dan"; u-a;" bytes, dan overschrijft
     u het user-defined-graphics gebied."
55Ø  IF a>=u THEN PRINT "U overschrijft zo het
     user-defined-graphics gebied."
56Ø  INPUT "Is dat goed (J of N)? ";z$
57Ø  IF z$="N" OR z$="n" THEN GO TO 7ØØØ
58Ø  IF z$<>"J" OR z$<>"j" THEN BEEP.2,24:GO TO 56Ø
7ØØ  REM Ga verder en laad
71Ø  LET l=a
75Ø  GO SUB 82ØØ
76Ø  INPUT "Geef getal b, f, i, d, of t ";z$
77Ø  IF z$=""THEN BEEP .2,24: GO TO 76Ø
78Ø  LET a$=CHR$ (CODE z$(1)-32*(z$(1)>"£"))
79Ø  GO TO 8ØØ+2ØØ * (a$="B")+3ØØ * (a$="F")+4ØØ *
     (a$="I")+5ØØ * (a$="D")+6ØØ * (a$="T")
8ØØ  LET x=VAL zS
81Ø  IF l>p THEN BEEP .2,24:GO TO 75Ø
82Ø  IF x<Ø OR x>255 OR x<> INT x THEN BEEP
     .2,24:GO TO 76Ø
83Ø  POKE l,x
84Ø  LET l=l+1
85Ø  GO TO 74Ø
1ØØØ REM *****Beweeg vooruit
1Ø1Ø LET l=l-1
1Ø2Ø IF LEN z$>1 THEN LET l=l+1-VAL z$(2 TO)
1Ø3Ø IF l<a THEN LET l-a
1Ø4Ø GO TO 74Ø
11ØØ REM *****Beweeg achteruit
1111 LET l=l+1
1120 IF LEN z$>1 THEN LET l=l-1+VAL z$(2 TO )
1130 IF l>p THEN LET l=p
1140 GO TO 74Ø
12ØØ REM *****Invoegen
1210 IF LEN z$=1 THEN LET n=1:GO TO 1225
1220 LET n=VAL z$(2 TO ):IF n<1 OR n>p-l OR n<>INT
     n THEN BEEP .2,24:GO TO 74Ø
```

adres kiezen dat het programma dan in het RAMTOP-adres zet door middel van de opdracht CLEAR. Het programma zal geen adres lager dan 27000 accepteren omdat de routine dan binnen de benodigde ruimte voor het programma zelf zou vallen. Het programma vraagt naar een startadres voor de routine. De gebruiker kan dus ruimte voor verschillende routines reserveren en deze dan los van elkaar laden.
Eenmaal de gebruiker de gelegenheid te hebben gegeven zijn keuze te veranderen als hij niet tevreden is, projecteert het programma het hoofdmenu. Afbeelding 4.1 laat de vorm van het display zien wanneer de routine 'Beeldinversie' op locatie 32000 geladen is. De eerste kolom is het adres, de tweede de inhoud van het adres en de derde de check sum. De routine 'Beeldinversie' is 18 bytes lang en zijn check sum is 1613. Het bezet dus locaties 32000 tot 32017 en de check sum voor locatie 32017, dat wil zeggen, de som van de locaties 32000 tot 32017 is 1613.
Wanneer het hoofdmenu wordt gepresenteerd, wordt de aandacht van de gebruiker naar een locatie geleid omdat de decimaal van FLASH voorzien is. Het heet de actuele (Engels: current) locatie en in beginsel is dat het gekozen startadres van de routine. De gebruiker toetst een heel getal in tussen 0 t/m 255, dat het programma in de actuele locatie POKEd en dan wordt het volgende adres de actuele locatie. Zodoende kan een complete routine op zijn plaats worden ge-POKEd, het display bijgewerkt en als het nodig is bij elke stap ge-scrolled.
De gebruiker kan kiezen een getal intoetsen of een selectie maken uit één van de in tabel 4.1 opgesomde opties. Deze faciliteiten creëren de mogelijkheid om correcties te maken.

Tabel 4.1 Deze opties zijn beschikbaar bij het werken met machinetaal

Code	*Optie*
b	De actuele locatie één adres achterwaarts verplaatsen.
b getal	De actuele locatie het ingegeven aantal adressen achterwaarts verplaatsen.
f	De actuele locatie één adres voorwaarts verplaatsen.
f getal	De actuele locatie het ingegeven aantal adressen voorwaarts verplaatsen.
i getal	Een ingegeven aantal bytes op 0 vanaf de actuele locatie tussenvoegen.
d getal	Een ingegeven aantal bytes vanaf de actuele locatie verwijderen.
t	Einde programma.

Programma P4.1: Machinetaallader (l is kleine letter 'l')

```
1ØØ  GO SUB 81ØØ
2ØØ  REM***** Bereken beschikbaar geheugen
21Ø  LET min=1+PEEK 2373Ø+256*PEEK 23731
22Ø  LET p=PEEK 23732+256*PEEK 23733
23Ø  LET t=p-min+1
4ØØ  REM*****Haal startadres
41Ø  PRINT "Laagste start = ";min,,, "Maximaal
     beschikbare geheugenruimte = ";t
```

Machinetaallader

Adres	Decimaal	Check sum
32000	33	33
32001	0	33
32002	64	97
32003	1	98
32004	0	98
32005	24	122
32006	22	144
32007	255	399
32008	122	521
32009	150	671
32010	119	790
32011	35	825
32012	11	836
32013	120	956
32014	177	1133
32015	32	1165
32016	247	1412
32017	201	1613

Afbeelding 4.1: Het door de machinetaallader geproduceerde scherm wanneer de Beeld-inversieroutine op locatie 32000 werd geladen

5. Scroll-routines

5.1 Scroll attributen naar links

Lengte: 23
Aantal variabelen: 1
Check sum: 1574

Werking

Deze routine rolt (Engels: scroll) de attributen van alle tekens van het scherm één positie naar links.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
new attr	1	23296	Het inkomende attribuut in de meest rechtse kolom.

Call

RAND USR adres

Foutafhandeling

Geen

Commentaar

Deze routine is te gebruiken om de nadruk te leggen op een tekst of graphics-gedeelte. Om alleen de bovenste 22 regels te scroll-en, zou de 24* in 22 moeten worden veranderd.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 22528	33 0 88
	ld a, (23296)	58 0 91
	ld c, 24	14 24*
next line	ld b, 31	6 31
next char	inc hl	35
	ld e, (hl)	94
	dec hl	43
	ld (hl),e	115
	inc hl	35
	djnz, next char	16 249
	ld (hl),a	119
	inc hl	35
	dec c	13
	jr nz, next line	32 242
	ret	201

Hoe het werkt

Registerpaar hl wordt met het adres van het attributengebied geladen. De accumulator wordt met de waarde van het in de meest rechtse kolom inkomende attri-

buut geladen. Het register c wordt met het aantal te scrollen regels geladen, zodat het als een teller kan dienen. Het register b wordt met het aantal tekens per regel, min één, geladen, om als teller te worden gebruikt.
hl wordt verhoogd zodat het naar het volgende attribuut wijst en deze laatste komt in register e. hl wordt verlaagd en wordt dan met de waarde uit e gePOKEd. hl wordt opnieuw verhoogd zodat het naar het volgende attribuut wijst. Register b wordt verlaagd, en als het nog niet nul is, dan wordt naar 'next char' teruggesprongen. hl wijst nu naar de meest rechtse kolom en dit wordt met de waarde uit de accumulator gePOKEd. hl wordt verhoogd zodat het naar de start van de nieuwe regel wijst. De regelteller, register c, wordt verlaagd. Als de resulterende waarde niet nul is, loopt de routine naar 'next line' terug.
De routine komt dan in BASIC terug.

5.2 Scroll attributen naar rechts

Lengte: 23
Aantal variabelen: 1
Check sum: 1847

Werking

Deze routine scrollt de attributen van alle tekens van het scherm één positie naar rechts.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
new attr	1	23296	Het inkomende attribuut in de linker kolom.

Call

RAND USR adres

Foutafhandeling

Geen

Commentaar

Deze routine is te gebruiken om de nadruk te leggen op een tekst of graphics-gedeelte. Om alleen de bovenste 22 regels te scrollen, zou de 24* in 22 moeten worden veranderd.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 23295	33 255 90
	ld a, (23296)	58 0 91
	ld c, 24	14 24*
next line	ld b, 31	6 31
next char	dec hl	43
	ld e, (hl)	94
	inc hl	35
	ld (hl),e	115
	dec hl	43
	djnz, next char	16 249
	ld (hl),a	119

dec hl	43
dec c	13
jr nz, next line	32 242
ret	201

Hoe het werkt
Het hl-registerpaar wordt met het adres van de laatste byte uit het attributengebied geladen. De accumulator wordt met de waarde van het in de meest linkse kolom inkomende attribuut geladen. In register c wordt het aantal te scrollen regels geladen, zodat het als teller kan dienen. In register b wordt het aantal tekens, min één, geladen, om als teller te worden gebruikt.
hl wordt verlaagd zodat het naar het volgende attribuut wijst en deze laatste komt in register e. hl wordt verhoogd en wordt dan met de waarde uit e gePOKEd. hl wordt opnieuw verlaagd zodat het naar het volgende attribuut wijst. De teller in b wordt verlaagd, en als het nog niet nul is, loopt de routine naar 'next char' terug.
hl wijst nu naar de meest linkse kolom en dit wordt met de waarde uit de accumulator gePOKEd. hl wordt verlaagd zodat het naar het einde van de nieuwe regel wijst. De regelteller wordt verlaagd en als dit niet nul is, loopt de routine naar 'next line' terug.
De routine komt dan in BASIC terug.

5.3 Scroll attributen omhoog
Lengte: 21
Aantal variabelen: 1
Check sum: 1591

Werking
Deze routine scrollt de attributen van alle tekens van het scherm één positie omhoog.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
new attr	1	23296	Het inkomende attribuut in de onderste regel.

Call
RAND USR adres

Foutafhandeling
Geen

Commentaar
Deze routine is te gebruiken om de nadruk te leggen op een tekst of graphicsgedeelte. Om alleen de bovenste 22 regels te scrollen zou de 224* in 160 moeten worden veranderd.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 22560	33 32 88
	ld de, 22528	17 0 88
	ld bc, 736	1 224* 2

```
                ldir                237 176
                ld a, (23296)       58 0 91
                ld b, 32            6 32
next char       ld (de),a           18
                inc de              19
                djnz next char      16 252
                ret                 201
```

Hoe het werkt
hl wordt met het adres van de tweede regel van de attributen geladen; de, met dat van de eerste regel en bc met het aantal te verplaatsen bytes.
De bc-bytes startend op het hl-adres worden op de gekopieerd, door middel van de 'ldir'-instructie. Het resultaat is een verwijzing naar het einde van de attributen. De accumulator wordt geladen met de code van het inkomende attribuut op de onderste regel. Register b wordt dan met het aantal tekens per regel geladen om als teller te worden gebruikt.
de wordt gePOKEd met de waarde uit de accumulator, en wordt dan verhoogd zodat het naar de volgende byte wijst. De teller wordt verlaagd, en als het niet nul is geworden, loopt de routine naar 'next char' terug. De routine komt dan in BASIC terug.

5.4 Scroll attributen omlaag

Lengte: 21
Aantal variabelen: 1
Check sum: 2057

Werking
Deze routine scrollt de attributen van alle tekens van het scherm één positie omlaag.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
new attr	1	23296	Het inkomende attribuut in de bovenste kolom.

Call
RAND USR adres

Foutafhandeling
Geen

Commentaar
Deze routine is te gebruiken om de nadruk te leggen op een tekst- of graphicsgedeelte. Om alleen de bovenste 22 regels te scrollen, moeten de volgende veranderingen worden aangebracht:

233*	naar 159
255**	naar 191
224***	naar 160

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 23263	33 223* 90
	ld de, 23295	17 255** 90
	ld bc, 736	1 224*** 2
	lddr	237 184
	ld a, (23296)	58 0 91
	ld b, 32	6 32
next char	ld (de),a	18
	dec de	27
	djnz next char	16 252
	ret	201

Hoe het werkt

Het hl-registerpaar wordt met het adres van het laatste attribuut van de 23ste regel geladen. de wordt met het adres van het laatste attribuut van de 24ste regel geladen. In bc komt het aantal te verplaatse bytes. De 'lddr'-instructie verplaatst dan de bc-bytes (de laatste ervan wordt door hl aangegeven) zodat het bij de eindigt. Deze bevat dan het adres van het laatste attribuut op de eerste regel.
De accumulator wordt dan geladen met de code van de inkomende attribuut op de eerste regel. Register b wordt geladen met het aantal bytes van de eerste regel; dit wordt als teller gebruikt. de wordt gePOKEd met de waarde uit de accumulator, en de wordt verlaagd zodat het naar de volgende byte wijst. De teller wordt verlaagd en als het niet nul is wordt er naar 'next char' teruggesprongen.
De routine komt dan in BASIC terug.

5.5 Scroll één teken naar links

Lengte: 21
Aantal variabelen: 1
Check Sum: 1745

Werking

Deze routine scrollt de inhoud van de display file één teken naar links.

Call

RAND USR adres

Foutafhandeling

Geen

Commentaar

Deze routine is bruikbaar wanneer het scherm als een 'raam' wordt gebruikt, om een klein gedeelte van een groter display-gebied te laten zien. Het 'raam' wordt verplaatst door middel van scroll-routines.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 16384	33 0 64
	ld d,l	85
	ld a,192	62 192

next line	ld b,31	6 31
next byte	inc hl	35
	ld e, (hl)	94
	dec hl	43
	ld (hl),e	115
	inc hl	35
	djnz next byte	16 249
	ld (hl),d	114
	inc hl	35
	dec a	61
	jr nz, next line	32 242
	ret	201

Hoe het werkt
Het hl-registerpaar wordt geladen met het adres van de display file, en register d wordt op nul gezet. De accumulator wordt met het aantal regels van het scherm geladen. In register b komt het aantal tekens per regel min één, omdat dit het aantal te kopiëren bytes is.
hl wordt verhoogd zodat het naar de volgende byte wijst, en deze laatste komt in register e. hl wordt verlaagd en wordt met de waarde uit e gePOKEd. hl wordt verhoogd om de volgende byte te adresseren, en de teller in b wordt verlaagd. Als dat niet nul is loopt de routine naar 'next byte' terug.
Als register b nul bevat, is de laatste byte gekopieerd en hl wijst naar de meest rechtse byte. Dit is dan met nul gePOKEd, en hl verhoogd om naar de volgende regel te wijzen. De regelteller in de accumulator wordt verlaagd en als het niet nul is wordt er naar 'next line' gesprongen.
De routine komt dan in BASIC terug.

5.6 Scroll één teken naar rechts

Lengte: 22
Aantal variabelen: 0
Check sum: 1976

Werking
Deze routine scrollt de inhoud van de display file één teken naar rechts.

Call
RAND USR adres

Foutafhandeling
Geen

Commentaar
Deze routine is bruikbaar wanneer het scherm als een 'raam' wordt gebruikt, om een klein gedeelte van een groter display-gebied te laten zien. Het 'raam' wordt verplaatst door middel van scroll-routines.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 22527	33 255 87
	ld d, 0	22 0
	ld a, 192	62 192

next line	ld b, 31	6 31
next byte	dec hl	43
	ld e, (hl)	94
	inc hl	35
	ld (hl),e	115
	dec hl	43
	djnz next byte	16 249
	ld (hl),d	114
	dec hl	43
	dec a	61
	jr nz, next line	32 242
	ret	201

Hoe het werkt
Het hl-registerpaar wordt geladen met hèt adres van de laatste byte in de display file, en register d wordt op nul gezet. De accumulator wordt met het aantal regels van het scherm geladen. In register b komt het aantal tekens per regel min één, om als teller te worden gebruikt.
hl wordt verlaagd zodat het naar de volgende byte wijst, en register e wordt met deze waarde geladen. hl wordt verhoogd en met de waarde uit e gePOKEd. hl wordt verlaagd om de volgende byte te adresseren, en de teller in b wordt verlaagd. Als dat niet nul is loopt de routine naar 'next byte' terug.
Als register b nul bevat, is de laatste byte gekopieerd en hl wijst naar de meest linkse byte. Dit wordt dan met nul gePOKEd, en hl wordt verlaagd om naar de volgende regel te wijzen. De teller in de accumulator wordt verlaagd en als het niet nul is wordt er naar 'next line' gesprongen.
De routine komt dan in BASIC terug.

5.7 Scroll één teken omhoog
Lengte: 68
Aantal variabelen: 0
Check sum: 6328

Werking
Deze routine scrollt de inhoud van de display file acht pixels omhoog.

Call
RAND USR adres

Foutafhandeling
Geen

Commentaar
Geen

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 16384	33 0 64
	ld de, 16416	17 32 64
save	push hl	229
	push de	213

	ld c,23	14 23
next line	ld b,32	6 32
copy byte	ld a, (de)	26
	ld (hl),a	119
	ld a,c	121
	and 7	230 7
	cp 1	254 1
	jr nz, next byte	32 2
	sub a	151
	ld (de),a	18
next byte	inc hl	35
	inc de	19
	djnz copy byte	16 241
	dec c	13
	jr z, restore	40 19
	ld a,c	121
	and 7	230 7
	cp 0	254 0
	jr z, next block	40 22
	cp 7	254 7
	jr nz, next linc	32 225
	push de	213
	ld de, 1792	17 0 7
	add hl,de	25
	pop de	209
	jr next line	24 217
restore	pop de	209
	pop hl	225
	inc d	20
	inc h	36
	ld a,h	124
	cp 72	254 72
	jr nz, save	32 204
	ret	201
next block	push hl	229
	ld hl, 1792	33 0 7
	add hl,de	25
	ex de,hl	235
	pop hl	225
	jr next line	24 198

Hoe het werkt

hl wordt geladen met het adres van de display file en de wordt geladen met het adres van de byte acht regels lager. hl en de worden op de stapel opgeslagen. Register c wordt geladen met het aantal te PRINTen regels op het scherm, min één. Register b wordt geladen met het aantal bytes per regel van het scherm, om als teller te worden gebruikt.

De accumulator wordt geladen met de door de geadresseerde byte en dit wordt in hl gePOKEd. De accumulator wordt geladen met de inhoud van c en als dit 1, 9 of 17 bevat, dan wordt dc met nul gePOKEd. hl en de worden verhoogd om naar de volgende bytes te wijzen, de teller in b wordt verlaagd en als dit niet nul is, wordt er naar 'copy byte' gesprongen.

De regelteller in c wordt dan verlaagd. Als dit nul is, wordt er naar 'restore' gesprongen. Als c de waarde 8 of 16 bevat, dan wordt er naar 'next block' gesprongen. Als c niet de waardes 7 of 15 bevat, dan loopt de routine naar 'next line'. De waarde 1792 wordt bij hl opgeteld, zodat het naar het volgende blok van het scherm wijst. De routine springt dan naar 'next line'.
Bij 'restore' worden de en hl van de stapel opgehaald en het getal 256 wordt bij beiden opgeteld. Dus, de en hl wijzen nu naar een regel lager dan de positie die ze in de voorgaande lus hadden. Als hl 18432 bevat, gaat de routine naar BASIC terug, anders wordt er naar 'save' gesprongen. Bij 'next block' wordt het getal 1792 bij de opgeteld, zodat de naar het volgende blok van het scherm wijst. De routine gaat dan naar 'next line'.

5.8 Scroll één teken omlaag

Lengte: 73
Aantal variabelen: 0
Check sum: 7987

Werking
Deze routine scrollt de inhoud van de display file acht pixels omlaag.

Call
RAND USR adres.

Foutafhandeling
Geen

Commentaar
Geen

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 22527	33 255 87
	ld de, 22495	17 223 87
save	push hl	229
	push de	213
	ld c, 23	14 23
next line	ld b,32	6 32
copy byte	ld a, (de)	26
	ld (hl),a	119
	ld a,c	121
	and 7	230 7
	cp 1	254 1
	jr nz, next byte	32 2
	sub a	151
	ld (de),a	18
next byte	dec hl	43
	dec de	27
	djnz copy byte	16 241
	dec c	13
	jr z, restore	40 21
	ld a,c	121

```
                and 7               230 7
                cp 0                254 0
                jr z, next block    40 24
                cp 7                254 7
                jr nz, next line    32 225
                push de             213
                ld de, 1792         17 0 7
                and a               167
                sbc hl,de           237 82
                pop de              209
                jr next line        24 215
restore         pop de              209
                pop hl              225
                dec d               21
                dec h               37
                ld a,h              124
                cp 79               254 79
                ret z               200
                jr save             24 201
next block      push hl             229
                ld hl, 1792         33 0 7
                ex de, hl           235
                and a               167
                sbc hl,de           237 82
                ex de, hl           235
                pop hl              225
                jr next line        24 193
```

Hoe het werkt

Het hl-registerpaar wordt geladen met het adres van de laatste byte in de display file, en register de wordt geladen met het adres van de byte acht regels hoger. hl en de worden op de stapel opgeslagen. Register c wordt dan geladen met het aantal te PRINTen regels op het scherm, min één. Register b wordt geladen met het aantal bytes per regel van het scherm, om als teller te worden gebruikt.

De accumulator wordt geladen met de door de geadresseerde byte en dit wordt in hl gePOKEd. De accumulator wordt geladen met de inhoud van c en als dit 1, 9 of 17 bevat, wordt de, met nul gePOKEd. hl en de worden dan verlaagd om naar de volgende bytes van het display te wijzen. De teller in b wordt verlaagd en als het niet nul is, wordt er naar 'copy byte' gesprongen.

De regelteller in c wordt verlaagd, en als het niet nul bevat wordt er naar 'restore' gesprongen. Als c de waarde 8 of 16 bevat, wordt er naar 'next block' gesprongen. Als c niet de waardes 7 of 15 bevat, loopt de routine naar 'next line'. De waarde 1792 wordt dan van hl afgetrokken, zodat hl naar het volgende blok van het scherm wijst. De routine springt dan naar 'next line'.

Bij 'restore' worden de en hl van de stapel opgehaald en het getal 256 wordt van beide afgetrokken. Dus, de en hl wijzen nu naar een regel hoger dan de positie die ze in de voorgaande lus hadden. Als hl 20479 bevat gaat de routine naar BASIC terug, anders wordt er naar 'save' gesprongen. Bij 'next block' wordt het getal 1792 van de afgetrokken, zodat de naar het volgende blok van het scherm wijst. De routine gaat dan naar 'next line'.

5.9 Scroll één pixel naar links

Lengte: 17
Aantal variabelen: 0
Check sum: 1828

Werking
Deze routine scrollt de inhoud van de display file één pixel naar links.

Call
RAND USR adres

Foutafhandeling
Geen

Commentaar
Deze routine geeft een soepeler beweging dan 'Scroll één teken naar links' maar er zijn acht calls voor nodig om het display met één vol teken te verplaatsen.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 22527	33 255 87
	ld c, 192	14 192
next line	ld b, 32	6 32
	or a	183
next byte	rl (hl)	203 22
	dec hl	43
	djnz next byte	16 251
	dec c	13
	jr nz, next line	32 245
	ret	201

Hoe het werkt
Het hl-register wordt geladen met het adres van de laatste byte in de display file en register c wordt geladen met het aantal regels in de display file en dient als teller. Register b wordt geladen met het aantal bytes van één regel om als teller te worden gebruikt. De carry-vlag wordt dan op nul gezet.
De door hl geadresseerde byte wordt dan één bit naar links geroteerd, zodat de carry-vlag naar de meest rechtse bit wordt gekopieerd, en de meest linkse bit wordt naar de carry-vlag gekopieerd. Register hl wordt dan verlaagd zodat het naar de volgende byte wijst en de teller in het b register wordt verlaagd. Als dit niet nul bevat, loopt de routine naar 'next line' terug. De regelteller wordt verlaagd, en als dit niet gelijk is aan nul, springt de routine naar 'next line' terug.
De routine springt dan in BASIC terug.

5.10 Scroll één pixel naar rechts

Lengte: 17
Aantal variabelen: 0
Check sum: 1550

Werking
Deze routine scrollt de inhoud van de display file één pixel naar rechts.

Call
RAND USR adres

Foutafhandeling
Geen

Commentaar
Deze routine geeft een soepeler beweging dan 'Scroll één teken naar rechts', maar er zijn acht calls voor nodig om het display met één vol teken te verplaatsen.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 16384	33 0 64
	ld c, 192	14 192
next line	ld b, 32	6 32
	or a	183
next byte	rr (hl)	203 30
	inc hl	35
	djnz next byte	16 251
	dec c	13
	jr nz, next line	32 245
	ret	201

Hoe het werkt
Het registerpaar hl wordt geladen met het adres van de display file, en register c wordt geladen met het aantal regels van het display en dient als een teller. Register b wordt geladen met het aantal bytes van één regel om als teller te worden gebruikt. De carry-vlag wordt dan op nul gezet. De door hl geadresseerde byte wordt dan één bit naar rechts geroteerd, zodat de carry-vlag naar de meest linkse bit wordt gekopieerd, en de meest rechtse bit wordt naar de carry-vlag gekopieerd. Register hl wordt dan verhoogd zodat het naar de volgende byte wijst en de teller in het b-register wordt verlaagd. Als dit niet nul bevat, loopt de routine naar 'next byte' terug. De regelteller wordt verlaagd, en als dit niet gelijk aan nul is, springt de routine naar 'next line' terug.
De routine komt dan in BASIC terug.

5.11 Scroll één pixel omhoog
Lengte: 91
Aantal variabelen: 0
Check sum: 9228

Werking
Deze routine scrollt de inhoud van de display file één pixel omhoog.

Call
RAND USR adres.

Foutafhandeling
Geen

Commentaar
Geen

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 16384	33 0 64
	ld de, 16640	17 0 65
	ld c, 192	14 192
next line	ld b, 32	6 32
next byte	ld a, (de)	26
	ld (hl),a	119
	ld a,c	121
	cp 2	254 2
	jr nz, next byte	32 2
	sub a	151
	ld (de),a	18
next byte	inc de	19
	inc hl	35
	djnz copy byte	16 243
	push de	213
	ld de, 224	17 224 0
	add hl,de	25
	ex (sp),hl	227
	add hl,de	25
	ex de,hl	235
	pop hl	225
	dec c	13
	ld a,c	121
	and 7	230 7
	cp 0	254 0
	jr nz, subtract	32 10
	push de	213
	ld de, 2016	17 224 7
	and a	167
	sbc hl,de	237 82
	pop de	209
	jr next block	24 14
subtract	cp 1	254 1
	jr nz, next block	32 10
	push hl	229
	ex de,hl	235
	ld de, 2016	17 224 7
	and a	167
	sbc hl,de	237 82
	ex de,hl	235
	pop hl	225
next block	ld a,c	121
	and 63	230 63
	cp 0	254 0
	jr nz, add	32 6
	ld a,7	62 7
	add h	132
	ld h,a	103
	jr next line	24 187

```
add     cp 1              254 1
        jr nz, next line  32 183
        ld a,7            62 7
        add d             130
        ld d,a            87
        ld a,c            121
        cp 1              254 1
        jr nz, next line  32 174
        ret               201
```

Hoe het werkt
Het registerpaar hl wordt geladen met het adres van de display file, en het registerpaar de wordt geladen met het adres van de eerste byte van de tweede regel van het display. Register c wordt geladen met het aantal regels van het display. Register b wordt geladen met het aantal bytes per regel, om als teller te worden gebruikt.
De accumulator wordt geladen met de door de geadresseerde byte. Dit wordt dan in het hl-adres gePOKEd. De accumulator wordt geladen met de inhoud van het c-register. Als dit de waarde twee bevat, wijst de naar de onderste regel van het scherm, en dus wordt dit met nul gePOKEd. de en hl worden dan verhoogd zodat ze naar de volgende bytes wijzen. De teller in het b-register wordt verlaagd, en als het niet nul is loopt de routine naar 'copy byte'.
Er wordt 224 zowel bij hl als bij de opgeteld, zodat ze naar de volgende regel van het scherm wijzen. De regelteller, in register c, wordt verlaagd. Als de waarde van c geen veelvoud van 8 is, wordt er naar 'subtract' gesprongen. Er wordt 2016 van hl afgetrokken, en er wordt naar 'next block' gesprongen. Dit om hl naar de volgende set van acht regels te laten wijzen.
Bij 'subtract' wordt als de waarde (c-1) geen veelvoud van acht is, naar 'next block' gesprongen; anders wordt 2016 van de afgetrokken zodat de naar de volgende set van acht regels wijst. Bij 'next block' wordt, als c een veelvoud is van 64, 1792 bij hl opgeteld, en er wordt naar 'next line' gesprongen zodat hl naar het volgende blok van 64 regels wijst. Bij 'add', als (c-1) een veelvoud is van 64, wordt 1792 bij de opgeteld zodat de naar het volgende blok van 64 regels wijst. Als c niet de waarde 1 bevat, gaat de routine naar 'next line', anders komt de routine in BASIC terug.

5.12 Scroll één pixel omlaag
Lengte: 90
Aantal variabelen: 0
Check sum: 9862

Werking
Deze routine scrollt de inhoud van de display file één pixel omlaag.

Call
RAND USR adres

Foutafhandeling
Geen

Commentaar
Geen

byte uit de display-file wordt de logische OR-functie uitgevoerd. De resulterende waarde wordt dan naar het display teruggekopieerd.
hl en de worden naar de volgende positie verplaatst, en de teller wordt verlaagd. Als de teller niet nul is, loopt de routine terug om het proces op de volgende byte te herhalen.

6.2 Beeldinversie

Lengte: 18
Aantal variabelen: 0
Check sum: 1613

Werking

Inverteert alles uit de display file – waar een punt uit was, wordt het aangezet, en waar een punt aan was, wordt het uitgezet.

Call

RAND USR adres

Foutafhandeling

Geen

Commentaar

Deze routine kan in spelletjes worden gebruikt om een explosie-effect te creëren. Dit effect wordt verbeterd als deze routine meerdere malen wordt aangeroepen, met een zekere vorm van geluid erbij.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 16384	33 0 64
	ld bc, 6144	1 0 24
	ld d, 255	22 255
next byte	ld a,d	122
	sub (hl)	150
	ld (hl),a	119
	inc hl	35
	dec bc	11
	ld a,b	120
	or c	177
	jr nz, next byte	32 247
	ret	201

Hoe het werkt

Het hl-registerpaar wordt geladen met het adres van de display file, en bc met de lengte ervan. Register d wordt op 255 gezet. Iedere keer dat de routine naar 'next byte' teruggaat wordt de accumulator vanaf d geladen. Deze methode wordt liever gebruikt dan de 'ld a,255'-instructie, omdat 'ld a,d' ongeveer de helft van de tijd neemt van die van de instructie 'ld a,255'. De waarde van de in hl opgeslagen byte wordt van de accumulator afgetrokken, en het resultaat wordt in hl teruggekopieerd, dus het wordt geïnverteerd.
hl wordt verhoogd zodat het naar de volgende byte wijst, en de teller bc wordt verlaagd. Als de teller niet nul is, gaat de routine naar 'next byte' terug. Als de teller nul is, komt de routine in BASIC terug.

6.3 Inverteer teken verticaal

Lengte: 20
Aantal variabelen: 1
Check sum: 1757

Werking

Deze routine keert een teken verticaal om; bijvoorbeeld een up-arrow (pijl omhoog) zou een down-arrow (pijl omlaag) worden.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
chr. start	2	23296	Adres van tekendata in RAM.

Call

RAND USR adres

Foutafhandeling

Geen

Commentaar

Deze routine is bruikbaar voor spelletjes als 'Minefield' en 'Puckman' omdat symbolen de richting kunnen veranderen zonder meer dan één teken te gebruiken.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23296)	42 0 91
	ld d,h	84
	ld e,l	93
	ld b,8	6 8
next byte	ld a, (hl)	126
	inc hl	35
	push af	245
	djnz next byte	16 251
	ld b,8	6 8
replace	pop af	241
	ld (de),a	18
	inc de	19
	djnz replace	16 251
	ret	201

Hoe het werkt

Register hl wordt geladen met het adres van de tekendata in RAM. Dit wordt naar de gekopieerd. Register b wordt op 8 gezet, om als teller te dienen.
Voor iedere byte wordt de accumulator met de aanwezige waarde geladen, hl wordt verlaagd en de accumulator wordt op de stapel opgeslagen. De teller wordt verlaagd en als dit niet nul is, loopt de routine terug om dit proces voor de volgende byte te herhalen. Register b wordt opnieuw met 8 geladen om nogmaals als teller te dienen.
Voor iedere byte wordt de accumulator van de stapel opgehaald en in het adres uit de gePOKEd. de wordt dan verhoogd om naar de volgende byte te wijzen en de

teller wordt verlaagd. Als dit niet nul is, gaat de routine naar 'replace' terug. Daarna wordt er een return naar BASIC uitgevoerd.

6.4 Inverteer teken horizontaal

Lengte: 19
Aantal variabelen: 1
Check sum: 1621

Werking

Deze routine keert een teken horizontaal om. Bijvoorbeeld een left-arrow (pijl naar links) wordt een right-arrow (pijl naar rechts) en andersom.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
chr. start	2	23296	Adres tekendata in RAM.

Call

RAND USR adres

Foutafhandeling

Geen

Commentaar

Geen

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23296)	42 0 91
	ld a, 8	62 8
next byte	ld b, 8	6 8
next pixel	rr (hl)	203 30
	rl c	203 17
	djnz next pixel	16 250
	ld (hl),c.	113
	inc hl	35
	dec a	61
	jr nz, next byte	32 243
	ret	201

Hoe het werkt

Het hl-registerpaar wordt geladen met het adres van de tekendata in RAM, en de accumulator wordt geladen met het aantal te inverteren bytes. Register b wordt geladen met het aantal bits in elke byte, om als teller te gebruiken.
De byte op het door hl aangegeven adres wordt naar rechts geroteerd zodat de meest rechtse bit naar de carry-vlag gekopieerd wordt. Register c wordt naar links geroteerd zodat de carry-vlag naar de meest rechtse bit wordt gekopieerd. De teller in b wordt verlaagd. Als de teller niet nul is, wordt er naar 'next pixel' gesprongen. De geïnverteerde byte, welke in register c zit, wordt gePOKEd op het adres waar het oorspronkelijk vandaan kwam.
hl wordt verhoogd zodat het naar de volgende byte wijst, en de accumulator

wordt verlaagd. Als de accumulator niet nul is wordt er naar 'next byte' gesprongen.
Er wordt een return naar BASIC uitgevoerd.

6.5 Roteer teken rechtsom

Lengte: 42
Aantal variabelen: 1
Check sum: 3876

Werking

Deze routine keert een teken 90 graden om; bijvoorbeeld een pijl omhoog wordt een pijl naar rechts.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
chr. start	2	23296	Adres van tekendata in RAM.

Call

RAND USR adres.

Foutafhandeling

Geen

Commentaar

Deze routine is bruikbaar in spelletjes en serieuze applicaties; graphics bijvoorbeeld kunnen zodoende van labels worden voorzien.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23296)	42 0 91
	ld e, 128	30 128
next bit	push hl	229
	ld c, 0	14 0
	ld b,1	6 1
next byte	ld a, e	123
	and (hl)	166
	cp 0	254 0
	jr z, not set	40 3
	ld a,c	121
	add a,b	128
	ld c,a	79
not set	sla b	203 32
	inc hl	35
	jr nc, next byte	48 242
	pop hl	225
	push bc	197
	srl e	203 59
	jr nc, next bit	48 231
	ld de,7	17 7 0
	add hl,de	25
	ld b,8	6 8

```
replace     pop de          209
            ld (hl),e       115
            dec hl          43
            djnz replace    16 251
            ret             201
```

Hoe het werkt

Ieder teken bestaat uit een 8 × 8 groep pixels, waarvan elk aan- (=1) of uit- (=0) gezet kan worden. Bekijk ieder bit B-2 van byte B-1 in afbeelding 6.1. De opgeslagen data op locatie (B2, B1) in de matrix zal zijn:

$$\begin{pmatrix} N1 & N3 \\ N2 & N4 \end{pmatrix}$$

waarbij:

N1 = de byte waartussen de pixel (B2,B1) gevoegd wordt na de rotatie.
N2 = de bit in N1, waar het tussengevoegd zal worden.
N3 = de actuele waarde van die bit.
N4 = de waarde van bit N2.

1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1
1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1
2	4	2	4	2	4	2	4	2	4	2	4	2	4	2	4
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1
3	8	3	8	3	8	3	8	3	8	3	8	3	8	3	8
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1
4	16	4	16	4	16	4	16	4	16	4	16	4	16	4	16
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1
5	32	5	32	5	32	5	32	5	32	5	32	5	32	5	32
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1
6	64	6	64	6	64	6	64	6	64	6	64	6	64	6	64
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1
7	128	7	128	7	128	7	128	7	128	7	128	7	128	7	128

1 2 3 4 5 6 7 8 — Byte (B_1)

7 6 5 4 3 2 1 0 — Bit (B_2)

Afbeelding 6.1: Sleutel tot de tekenrotatie-routine

Iedere byte van het geroteerde teken wordt één voor één opgebouwd, door alle waardes van alle bits N2 bij elkaar op te tellen, hetgeen in de nieuwe byte komt.
Register hl wordt geladen met het adres van de eerste byte van het teken in RAM.
Register e wordt geladen met de waarde van de byte waarvan bit 7 aan staat en bits 0-6 uit, dwz. 128. hl wordt op de stapel opgeslagen. Het register c waarbij data opgeteld gaat worden met als resultaat de nieuwe waarde van de op te bouwen byte, wordt met nul geladen. Register b wordt geladen met de waarde van de byte waarvan bit 0 aan staat en bits 1 – 7 uit, dwz. 1.
De accumulator wordt geladen met de inhoud van het e-register (N3). Dit wordt geAND met de byte waarvan het adres in hl is opgeslagen. Als het resultaat nul is, wordt er naar 'not set' gesprongen, omdat de door e en hl geadresseerde pixel uit staat. Als dit aan staat wordt de accumulator met de huidige inhoud van byte (n1) geladen. Het register b (n4) wordt bij de accumulator opgeteld en dit wordt in c geladen. Register b wordt dan aangepast om naar de volgende bit van (n1) te wijzen. hl wordt verhoogd om naar de volgende byte (b1) te wijzen. Als de byte n1 niet compleet is, loopt de routine naar 'next byte' terug.
hl wordt van de stapel opgehaald, om opnieuw naar de eerste byte van het teken te wijzen. bc wordt op de stapel opgeslagen. De waarde ervan op dat moment is de waarde van de laatste byte die in c moet worden opgebouwd. Register e wordt aangepast om naar de volgende bit van elke byte te wijzen. Als de rotatie niet compleet is wordt er naar 'next bit' gesprongen.
Register de wordt met 7 geladen, en dit wordt bij hl opgeteld zodat hl naar de laatste byte van de data wijst. b wordt geladen met het aantal bytes dat van de stapel moet worden opgehaald. Voor iedere byte, wordt de nieuwe waarde in e gekopieerd, en dit wordt in hl gePOKEd. hl wordt verlaagd om naar de volgende byte te wijzen, en de teller in b wordt verlaagd. Als de teller niet nul is, wordt er naar 'replace' gesprongen.
De routine komt dan in BASIC terug.

6.6 Attributen veranderen

Lengte: 21
Aantal variabelen: 2
Check sum: 1952

Werking

Deze routine verandert de attributen van alle tekens van het scherm op een bepaalde manier, bijvoorbeeld de inkt-kleur zou kunnen worden veranderd, het hele scherm zou het flash-effect kunnen krijgen, enzovoort.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
data saved	1	23296	De niet te veranderen bits van attribuut.
new data	1	23297	Nieuwe in attribuut in te voegen bits.

Call

RAND USR adres

Foutafhandeling

Geen

Commentaar
Individuele bits van de attributen, van elk teken, kunnen worden veranderd door middel van de machinaalinstructies 'and' en 'or'.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 22528	33 0 88
	ld bc, 768	1 0 3
	ld de, (23296)	237 91 0 91
next byte	ld a, (hl)	126
	and e	163
	or d	178
	ld (hl), a	119
	inc hl	35
	dec bc	11
	ld a,b	120
	or c	177
	jr nz, next byte	32 246
	ret	201

Hoe het werkt
hl wordt geladen met het adres van het attributengebied, en het register bc met het aantal tekens van het display. Register d wordt geladen met de waarde 'new data', en het register e wordt geladen met 'data saved'.
De accumulator wordt geladen met de door hl geadresseerde byte, en de bits worden in overeenstemming met de waardes van de registers d en e aangepast. Het resultaat wordt in hl teruggePOKEd. hl wordt verhoogd om naar de volgende byte te wijzen, en de teller in bc wordt verlaagd. Als bc niet nul is, loopt de routine naar 'next byte'.
De routine komt dan in BASIC terug.

6.7 Attributen verwisselen
Lengte: 22
Aantal variabelen: 2
Check sum: 1825

Werking
Deze routine zoekt het attribuut-gebied door naar een bepaalde waarde, en vervangt iedere goed bevonden waarde door een andere.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
old value	1	23296	Waarde van de te vervangen byte.
new value	1	23297	Nieuwe waarde van vervangen byte.

Call
RAND USR adres

Foutafhandeling
Geen

Commentaar
Deze routine is te gebruiken om de nadruk te leggen op gebieden van tekst- of graphics-tekens.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 22528	33 0 88
	ld bc, 768	1 0 3
	ld de, (23296)	237 91 0 91
next byte	ld a, (hl)	126
	cp e	187
	jr nz, no change	32 1
	ld (hl), d	114
no change	inc hl	35
	dec bc	11
	ld a,b	120
	or c	177
	jr nz, next byte	32 245
	ret	201

Hoe het werkt
hl wordt geladen met het adres van het attributengebied, en het register bc met het aantal tekens van het display. Het register e wordt geladen met de 'old value' en het d-register wordt geladen met de 'new value'.
De accumulator wordt geladen met de door hl geadresseerde byte. Als de accumulator de waarde van het e-register bevat, wordt de door hl geadresseerde byte met de inhoud van het d-register gePOKEd. hl wordt dan verhoogd om naar de volgende byte te wijzen, en de teller in bc wordt verlaagd. Als bc niet nul is, springt de routine naar 'next byte'.
De routine komt dan in BASIC terug.

6.8 Gedeelte vullen
Lengte: 263
Aantal variabelen: 2
Check sum: 26647

Werking
Deze routine geeft een gedeelte van het scherm een tint; dit wordt door een lijn van pixels aan de rand van het scherm omgeven.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
x-co-ord	1	23296	x-coördinaat van de startpositie.
y-co-ord	1	23297	y-coördinaat van de startpositie.

Call
RAND USR adres

Foutafhandeling
Als de y-coördinaat groter is dan 175, of POINT (x,y) = 1, dan komt de routine onmiddellijk in BASIC terug.

Commentaar
Deze routine is niet verplaatsbaar en het startadres is 31955. Gebruik, om deze routine naar een ander adres te kopiëren, de methode die bij de 'Hernummer'-routine wordt gegeven. Als 31955 als het startadres van deze routine wordt gebruikt en 32218 als startadres van de routine 'Hernummer', dan mogen ze tegelijkertijd in de RAM aanwezig zijn. Bij het kleuren van erg onregelmatige gedeelten is een groot stuk RAM nodig. Als dit niet beschikbaar is, kan de routine vastlopen.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23296)	42 0 91
	ld a,h	124
	cp 176	254 176
	ret nc	208
	call subroutine	205 143* 125*
	and (hl)	166
	cp 0	254 0
	ret nz	192
	ld bc, 65535	1 255 255
	push bc	197
right	ld hl, (23296)	42 0 91
	call subroutine	205 143* 125*
	and (hl)	166
	cp 0	254 0
	jr nz, left	32 9
	ld hl, (23296)	42 0 91
	inc l	44
	ld (23296),hl	34 0 91
	jr nz, right	32 236
left	ld de, 0	17 0 0
	ld hl, (23296)	42 0 91
	dec l	45
	ld (23296),hl	34 0 91
plot	ld hl, (23296)	42 0 91
	push hl	229
	call subroutine	205 143* 125*
	or (hl)	182
	ld (hl),a	119
	pop hl	225
	ld a,h	124
	cp 175	254 175
	jr z, down	40 44
	ld a,e	123
	cp 0	254 0
	jr nz, reset	32 16
	inc h	36
	call subroutine	205 143* 125*

	and (hl)	166
	cp 0	254 0
	jr nz, reset	32 7
	ld hl, (23296)	42 0 91
	inc h	36
	push hl	229
	ld e,l	30 1
reset	ld hl, (23296)	42 0 91
	ld a,e	123
	cp 1	254 1
	jr nz, down	32 15
	inc h	36
	call subroutine	205 143* 125*
	and (hl)	166
	cp 0	254 0
	jr z, down	40 6
	ld e, 0	30 0
	jr down	24 2
long jump	jr right	24 167
down	ld hl, (23296)	42 0 91
	ld a,h	124
	cp 0	254 0
	jr z, next pixel	40 40
	ld a,d	122
	cp 0	254 0
	jr nz, restore	32 16
	dec h	37
	call subroutine	205 143* 125*
	and (hl)	166
	cp 0	254 0
	jr nz, restore	32 7
	ld hl, (23296)	42 0 91
	dec h	37
	push hl	229
	ld d,l	22 1
restore	ld a,d	122
	cp 1	254 1
	jr nz, next pixel	32 14
	ld hl, (23296)	42 0 91
	dec h	37
	call subroutine	205 143* 125*
	and (hl)	166
	cp 0	254 0
	jr z, next pixel	40 2
	ld d, 0	22 0
next pixel	ld hl, (23296)	42 0 91
	ld a,l	125
	cp 0	254 0
	jr z, retrieve	40 12
	dec l	45
	ld (23296),hl	34 0 91
	call subroutine	205 143* 125*
	and (hl)	166

	cp 0	254 0
	jr z, plot	40 129
retrieve	pop hl	225
	ld (23296),hl	34 0 91
	ld a, 255	62 255
	cp h	188
	jr nz, long jump	32 177
	cp 1	189
	jr nz, long jump	32 174
	ret	201
subroutine	push bc	197
	push de	213
	ld a, 175	62 175
	sub h	148
	ld h,a	103
	push hl	229
	and 7	230 7
	add a, 64	198 64
	ld c,a	79
	ld a,h	124
	rr a	203 31
	rr a	203 31
	rr a	203 31
	and 31	230 31
	ld b, a	71
	and 24	230 24
	ld d,a	87
	ld a,h	124
	and 192	230 192
	ld e,a	95
	ld h,c	97
	ld a,l	125
	rr a	203 31
	rr a	203 31
	rr a	203 31
	and 31	230 31
	ld l,a	111
	ld a,e	123
	add a,b	128
	sub d	146
	ld e,a	95
	ld d, 0	22 0
	push hl	229
	push de	213
	pop hl	225
	add hl,hl	41
	add hl,hl	41
	add hl,hl	41
	add hl,hl	41
	add hl,hl	41
	pop de	209
	add hl,de	25
	pop de	209
	ld a,e	123

```
                and 7           230 7
                ld b, a         71
                ld a, 8         62 8
                sub b           144
                ld b,a          71
                ld a,l          62 1
rotate          add a,a         135
                djnz rotate     16 253
                rr a            203 31
                pop de          209
                pop bc          193
                ret             201
```

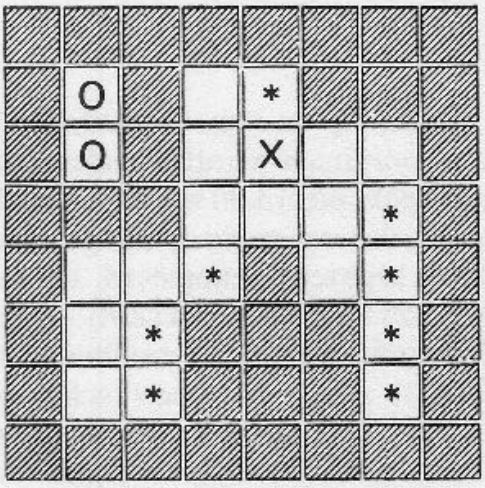
Hoe het werkt

Deze routine trekt horizontale lijnen van aaneenvolgende pixels, 'RUNS' genoemd, binnen gebieden die door verlichte pixels worden omgeven. Iedere 'RUN' wordt onthouden door de coördinaten van de meest rechtse pixel van de 'RUN' op de stapel op te slaan. De routine begint bij de aangegeven coördinaten, vult iedere 'RUN' in, en noteert daarbij de positie van de daarboven of eronder niet gevulde 'RUNS'. Bij het afwerken van een 'RUN', wordt de laatste coördinatenset opgehaald en de betreffende 'RUN' wordt ingevuld. Het proces wordt herhaald tot er geen lege 'RUNS' meer zijn.

Afbeelding 6.2 illustreert de techniek. De vierkantjes stellen verlichte pixels voor, de x-tekens de startpositie binnen het te kleuren gebied, en de *-tekens de meest rechtse pixels van de 'RUNS'.

Deze routine kleurt de horizontale lijn in die de startpositie bevat en slaat op de stapel de startposities op van de 'RUNS' in de regels direct daarboven en eronder. Daarna kleurt het de lijn daarboven en dan die daaronder, en in het laatste geval noteert het dat er nog twee 'RUNS' meer starten op de volgende regel daaronder, enzovoort. Elke positie binnen het te kleuren gebied mag als startpositie worden geselecteerd, maar merk op dat de twee pixels die met 0 zijn aangegeven, onaangeroerd werden gelaten, omdat ze los zitten van het te kleuren gebied.

Het register h wordt geladen met de aangegeven y-coördinaat, en register l met de x-coördinaat. Als de waarde van de y-coördinaat groter is dan 175, komt de routi-



*Afbeelding 6.2: Een illustratie van de gebruikte techniek om een gebied te vullen. Grijze vierkantjes zijn al verlicht en geven het gebied aan dat 'gekleurd' moet worden. x is de startpositie, * zijn de starts van de 'RUNs' en 0 blijft onveranderd*

ne in BASIC terug. De aangeroepen 'subroutine' komt terug met het adres in het geheugen, van de bit (x,y). Als deze bit 'aan' staat, komt de routine in BASIC terug.
Het getal 65535 wordt in de stapel gePUSHed om de eerste opgeslagen waarde te markeren. Later in de routine, als er een getal van de stapel wordt opgehaald, wordt het als een stel coördinaten gebruikt. Als dit getal echter 65535 is, wordt er een return naar BASIC gemaakt, omdat de routine klaar is.
Het h-register wordt geladen met de y-coördinaat en het l-register met de x-coördinaat. De aangeroepen 'subroutine' komt terug met het adres van de bit (x,y). Als deze bit 'aan' staat, wordt er naar 'left' gesprongen. Anders wordt de x-coördinaat verhoogd, en naar 'right' gesprongen als x niet gelijk is aan 256.
Bij 'left' wordt de op nul gezet. Registers d en e worden als vlaggen gebruikt, d voor onder, e voor boven. De x-coördinaat wordt verlaagd. De subroutine wordt aangeroepen en het punt (x,y) wordt ge-plot. Als de y-coördinaat 175 is, springt de routine naar 'down'. Als de 'up flag' de waarde één heeft, wordt er naar 'reset' gesprongen. Als de bit (x, y+1) 'uit' is, worden de waardes van x en van y+1 op de stapel opgeslagen en de 'up flag' wordt op één gezet.
Bij 'reset', als de 'up flag' op nul wordt gezet, wordt naar 'down' gesprongen. Als de bit (x, y+1) 'aan' staat, wordt de 'up flag' op nul gezet. Bij 'down', als de y-coördinaat nul is, wordt er een sprong naar 'next pixel' gemaakt. Als de 'down flag' op één staat, wordt er naar 'restore' gesprongen. Als de bit (x, y−1) 'uit' staat, worden de waardes van x en y−1 op de stapel opgeslagen, en de 'down flag' wordt op één gezet.
Bij 'restore', als de 'down flag' op nul wordt gezet, wordt naar 'next pixel' gesprongen. Als de bit (x, y−1) 'aan' staat, wordt de 'down flag' op nul gezet. Bij 'next pixel', als de x-coördinaat nul is, springt de routine naar 'retrieve'. De x-coördinaat wordt verlaagd, en als de nieuwe bit (x, y) 'uit' staat, wordt er naar 'plot' gesprongen. Bij 'retrieve' worden een x- en een y-coördinaat van de stapel opgehaald. Als x en y beide gelijk zijn aan 255, komt de routine in BASIC terug, omdat het gebied helemaal gevuld is. Anders loopt de routine naar 'right' terug.
De subroutine moet het adres van de bit (x,y) in het geheugen berekenen. In BASIC zou dit adres zijn:

16384+INT(Z/8)+256 X (Z-8 X INT(Z/8))
+32 X (64 X INT (Z/64)+INT(Z/8)−8 X INT (Z/64))

waarbij Z = 175-Y

Registerpaar bc en de worden op de stapel opgeslagen. De accumulator wordt met 175 geladen en de y-coördinaat wordt hiervan afgetrokken. Het resultaat wordt in het h-register teruggekopieerd. hl wordt dan op de stapel opgeslagen. De vijf linker bits van de accumulator worden op nul gezet, en er wordt 64 bij opgeteld. Het resultaat wordt in register c gekopiëerd. Dit met 256 vermenigvuldigd geeft als resultaat 16384 + 256 × (Z−8 × INT(Z/8)). De accumulator wordt geladen met Z, en dit wordt door 8 gedeeld; het resultaat wordt in register b opgeslagen. Dit resultaat is INT(Z/8). Als de drie meest rechtse bits op nul worden gezet, krijgen we de waarde 8×INT (Z/64); deze waarde wordt in register d geladen.
De accumulator wordt met Z geladen, en de zes meest rechtse bits worden op nul gezet, wat als resultaat geeft: 64 × INT(Z/64). Dit wordt in register e geladen. De waarde van register c wordt in h gekopieerd. De accumulator wordt geladen met de x-coördinaat, dit wordt door 8 gedeeld en het resultaat wordt in l gekopieerd.

De accumulator wordt dan geladen met de waarde uit e, en de inhoud van b wordt erbij opgeteld. De waarde uit d wordt afgetrokken van het in register de geladen resultaat. Register hl wordt op de stapel opgeslagen, en dan met de waarde uit de geladen. Dit wordt met 32 vermenigvuldigd, register de wordt van de stapel opgehaald en bij hl opgeteld. Dus, hl bevat nu het adres van de bit (x,y).
De accumulator wordt geladen met de oorspronkelijke waarde van x. Door de vijf linker bits op nul te zetten, wordt de waarde $x-8 \times INT(x/8)$ verkregen. Het register b wordt dan geladen met de waarde acht minus de waarde van de accumulator, om als teller te worden gebruikt. De accumulator wordt op één gezet, en dit wordt met twee keer b-1 vermenigvuldigd.
Op dit moment zou een enkele bit in de accumulator moeten worden gezet, welke overeenkomt met de bit (x,y), door hl geadresseerd. de en bc worden dan van de stapel opgehaald en de subroutine keert naar de hoofdroutine terug.

6.9 Vormtabellen

Lengte: 196
Aantal variabelen: 2
Check sum: 20278

Werking

Deze routine maakt een vorm in elke afmeting op het scherm.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
X start	1	23296	X-coördinaat eerste pixel.
Y start	1	23297	Y-coördinaat eerste pixel.

Call

RAND USR adres

Foutafhandeling

Als A$ niet bestaat, lengte nul heeft, of geen informatie over de vorm bevat, komt de routine onmiddellijk in BASIC terug. Dit gebeurt ook als Y start groter is dan 175.

Commentaar

Dit is een nuttige methode om afbeeldingen in het geheugen op te slaan en die met grote snelheid op het scherm te projecteren.

De methode om deze routine te gebruiken is:
(I) LET A$ = 'vorminformatie'
(II) POKE 23296, X-coördinaat van eerste pixel.
(III) POKE 23297, Y-coördinaat van eerste pixel.
(IV) RAND USR adres

De vorminformatie is een string van tekens, welke de volgende betekenis heeft:
'0' plot punt
'5' verlaag X-coördinaat.
'6' verlaag Y-coördinaat.
'7' verhoog Y-coördinaat.
'8' verhoog X-coördinaat.

Alle andere tekens worden genegeerd. Deze routine heeft een 'wrap-around' faciliteit, dat wil zeggen als de X-coördinaat links, buiten het scherm valt, verschijnt het rechts, enzovoort.
Om de routine te veranderen zodat er een andere string dan A$ kan worden gebruikt, verander dan 65* in de code van het hoofdletterteken van de string-naam.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23627)	42 75 92
next variable	ld a, (hl)	126
	cp 128	254 128
	ret z	200
	bit 7,a	203 127
	jr nz, for next	32 23
	cp 96	254 96
	jr nc, number	48 11
	cp 65	254 65*
	jr z, found	40 35
string	inc hl	35
	ld e, (hl)	94
	inc hl	35
	ld d, (hl)	86
add	add hl,de	25
	jr increase	24 5
number	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
increase	inc hl	35
	jr next variable	24 225
for next	cp 224	254 224
	jr c, next bit	56 5
	ld de, 18	17 18 0
	jr add	24 236
next bit	bit 5,a	203 111
	jr z, string	40 228
next byte	inc hl	35
	bit 7,(hl)	203 126
	jr z, next byte	40 251
	jr number	24 228
found	inc hl	35
	ld c, (hl)	78
	inc hl	35
	ld b, (hl)	70
	inc hl	35
	ex de,hl	235
	ld a, (23297)	58 1 91
	cp 176	254 176
	ret nc	208

again	ld hl, (23296)	42 0 91
	ld a,b	120
	or c	177
	ret z	200
	dec bc	11
	ld a, (de)	26
	inc de	19
	cp 48	254 48
	jr nz, not plot	32 78
	push bc	197
	push de	213
	ld a, 175	62 175
	sub h	148
	ld h,a	103
	push hl	229
	and 7	230 7
	add a,64	198 64
	ld c,a	79
	ld a,h	124
	rr a	203 31
	rr a	203 31
	rr a	203 31
	and 31	230 31
	ld b,a	71
	and 24	230 24
	ld d,a	87
	ld a,h	124
	and 192	230 192
	ld e,a	95
	ld h,c	97
	ld a,l	125
	rr a	203 31
	rr a	203 31
	rr a	203 31
	and 31	230 31
	ld l,a	111
	ld a,e	123
	add a,b	128
	sub d	146
	ld e,a	95
	ld d, 0	22 0
	push hl	229
	push de	213
	pop hl	225
	add hl,hl	41
	add hl,hl	41
	add hl,hl	41
	add hl,hl	41
	add hl,hl	41
	pop de	209
	add hl,de	25
	pop de	209
	ld a,e	123
	and 7	230 7

```
                ld b,a          71
                ld a, 8         62 8
                sub b           144
                ld b,a          71
                ld a,1          62 1
rotate          add a,a         135
                djnz rotate     16 253
                rr  a           203 31
                pop de          209
                pop bc          193
                or (hl)         182
                ld (hl),a       119
here            jr again        24 165
not plot        cp 53           254 53
                jr nz, down     32 1
                dec l           45
down            cp 54           254 54
                jr nz, up       32 8
                dec h           37
                ld a,h          124
                cp 255          254 255
                jr nz, save     32 19
                ld h, 175       38 175
up              cp 55           254 55
                jr nz, right    32 8
                inc h           36
                ld a,h          124
                cp 176          254 176
                jr nz, save     32 7
                ld h, 0         38 0
right           cp 56           254 56
                jr nz, save     32 1
                inc l           44
save            ld (23296),hl   34 0 91
                jr here         24 215
```

Hoe het werkt

Het adres van de string A$ wordt gevonden via een aanpassing van het eerste gedeelte van de 'Instr$'-routine. De lengte van de string wordt in bc geladen, en het adres van het eerste teken van A$ wordt in de geladen. De accumulator wordt geladen met Y start, en als dit groter is dan 175, keert de routine naar BASIC terug. Register h wordt geladen met de Y-coördinaat, en l wordt geladen met de X-coördinaat. Als de waarde van het bc-registerpaar nul is geworden, keert de routine in BASIC terug, omdat het einde van de string was bereikt. bc wordt verlaagd, om aan te geven dat er nog een teken was bewerkt. Het volgende teken wordt in de accumulator geladen, en de wordt verhoogd om naar de volgende byte te wijzen. Als de accumulator niet de waarde 48 bevat, wordt er naar 'not plot' gesprongen. Het punt (X,Y) wordt ge-plot met gebruikmaking van de 'subroutine' uit de 'Gedeelte vullen'-routine. De routine springt dan terug naar 'again'.

Bij 'not plot', als de accumulator de waarde 53 bevat, wordt de X-coördinaat verlaagd. Bij 'down', als de accumulator de waarde 54 niet bevat, wordt er naar 'up' gesprongen. De Y-coördinaat wordt verlaagd, en als dit de waarde −1 bevat, wordt de Y-coördinaat op 175 gezet.
Bij 'up', als de accumulator niet de waarde 55 bevat, wordt naar 'right' gesprongen. De Y-coördinaat wordt verhoogd, en als dit de waarde 176 heeft, wordt het op 0 gezet. Bij 'right', als de accumulator het getal 56 bevat, wordt de X-coördinaat verhoogd. Bij 'save' worden de X- en de Y-coördinaten in het geheugen gePOKEd en de routine springt naar 'here'.

6.10 Beeld vergroten en kopiëren

Lengte: 335
Aantal variabelen: 8
Check sum: 33663

Werking

Deze routine kopieert een gedeelte van het scherm naar een andere plaats op het scherm en vergroot de kopie in het x- of y-vlak.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
upper y-co-ord	1	23296	y-coördinaat eerste rij.
lower y-co-ord	1	23297	y-coördinaat laatste rij.
right x-co-ord	1	23298	x-coördinaat meest rechtse kolom
left x-co-ord	1	23299	x-coördinaat meest linkse kolom.
horizontal scale	1	23300	Vergroting x-vlak.
vertical scale	1	23301	Vergroting y-vlak.
new left co-ord	1	23302	Meest linkse x-coördinaat nieuw gebied waarheen gekopieerd wordt.
new lower co-ord	1	23303	Onderste y-coördinaat nieuw gebied waarheen gekopieerd wordt.

Call

RAND USR adres

Foutafhandeling

Deze routine komt onmiddellijk in BASIC terug als één van de volgende condities waar is:

(I) horizontal scale = 0.
(II) vertical scale = 0.
(III) upper y-co-ord groter dan 175
(IV) nieuwe lower y co-ord groter dan 175
(V) lower y-co-ord groter dan upper y-co-ord
(VI) left x-co-ord groter dan right x-co-ord

Echter, om de routine kort te houden, is er geen controle die verzekert dat het gekopieerde gedeelte wel op het scherm past. Als dit niet het geval is, kan de routine vastlopen. De routine heeft ook een groot stuk RAM nodig, en als dit niet beschikbaar is, kan het vastlopen.

Commentaar

Deze routine is niet verplaatsbaar, als gevolg van het bestaan van een 'Plot/Point'-routine. Deze zit op adres 65033, en kan dus alleen op 48K RAM machines draaien. De routine kan in het geheugen worden verplaatst door middel van de in 'Hernummer' gegeven procedure. Echter, als grote delen van het scherm moeten worden gekopieerd, is er een groot stuk RAM voor nodig, en het startadres moet dus zo hoog mogelijk zitten.
Als het te kopiëren gedeelte van het scherm hetzelfde formaat krijgt als het oorspronkelijke, moeten de schalen op één worden gezet; laad, om het te verdubbelen, de schalen met twee, om het te verdrievoudigen, met drie, enzovoort.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld ix, 23296	221 33 0 91
	ld a, 175	62 175
	cp (ix + 0)	221 190 0
	ret c	216
	cp (ix + 7)	221 190 7
	ret c	216
	sub a	151
	cp (ix + 4)	221 190 4
	ret z	200
	cp (ix + 5)	221 190 5
	ret z	200
	ld hl, (23296)	42 0 91
	ld b, 1	69
	ld a,l	125
	sub h	148
	ret c	216
	ld (23298),a	50 0 91
	ld e,a	95
	ld hl, (23298)	42 2 91
	ld c,l	77
	ld a,l	125
	sub h	148
	ret c	216
	ld (23298),a	50 2 91
	push bc	197
	ld l,a	111
	ld h, 0	38 0
	inc hl	35
	push hl	229
	pop bc	193
	inc e	28
add	dec e	29
	jr z, remainder	40 3
	add hl,bc	9

	jr add	24 250
remainder	ld a,l	125
	and 15	230 15
	ld b,a	71
	pop hl	225
	ld c,l	77
	jr nz, save	32 2
full	ld b,16	6 16
save	push hl	229
	call subroutine	205 13* 255*
	and (hl)	166
	jr z, off	40 2
	ld a, 1	62 1
off	pop hl	225
	rr a	203 31
	rl e	203 19
	rl d	203 18
	ld a,l	125
	cp (ix + 3)	221 190 3
	jr z, next row	40 6
	dec l	45
next bit	djnz save	16 231
	push de	213
	jr full	24 226
next row	ld l,c	105
	ld a,h	124
	cp (ix + 1)	221 190 1
	jr z, copy	40 3
	dec h	37
	jr next bit	24 241
copy	push de	213
	ld b, 0	6 0
	ld h,b	96
	ld l,b	104
reset	ld (23306),hl	34 10 91
	ld a,b	120
	or a	183
	jr nz, retrieve	32 3
	pop de	209
	ld b, 16	6 16
retrieve	sub a	151
	dec b	5
	rr d	203 26
	rr e	203 27
	rl a	203 23
	push de	213
	push bc	197
	push af	245
	ld h,l	38 1
loop	ld l, l	46 1
preserve	ld (23304),hl	34 8 91
	ld a, (23307)	58 11 91

	ld hl, 0	33 0 0
	ld de, (23301)	237 91 5 91
	ld d,1	85
multiply	or a	183
	jr z, calculate	40 6
	add hl,de	25
	dec a	61
	jr multiply	24 249
long jump	jr reset	24 208
calculate	ld a, (23303)	58 7 91
	add a,l	133
	ld hl, (23304)	42 8 91
	add a,l	133
	dec a	61
	push af	245
	ld a, (23306)	58 10 91
	ld hl, 0	33 0 0
	ld de, (23300)	237 91 4 91
	ld d, l	85
repeat	or a	183
	jr z, continue	40 4
	add hl,de	25
	dec a	61
	jr repeat	24 249
continue	ld a, (23302)	58 6 91
	add l	133
	ld hl, (23305)	42 9 91
	add l	133
	dec a	61
	ld l,a	111
	pop af	241
	ld h,a	103
	pop af	241
	push af	245
	or a	183
	jr nz, plot	32 7
	call subroutine	205 13* 255*
	cpl	47
	and (hl)	166
	jr Poke	24 4
Plot	call subroutine	205 13* 255*
	or (hl)	182
Poke	ld (hl),a	119
	ld hl, (23304)	42 8 91
	inc l	44
	ld a, (23301)	58 5 91
	inc a	60
	cp l	189
	jr nz, preserve	32 165
	inc h	36
	ld a, (23300)	58 4 91
	inc a	60
	cp h	188

	jr nz, loop	32 155
	pop af	241
	pop bc	193
	pop de	209
	ld hl, (23306)	42 10 91
	inc l	44
	ld a, (23298)	58 2 91
	inc a	60
	cp l	189
	jr nz, long jump	32 164
	ld l, 0	46 0
	inc h	36
	ld a, (23296)	58 0 91
	inc a	60
	cp h	188
	jr nz, long jump	32 154
	ret	201
subroutine	push bc	197
	push de	213
	ld a, 175	62 175
	sub h	148
	ld h,a	103
	push hl	229
	and 7	230 7
	add a, 64	198 64
	ld c,a	79
	ld a,h	124
	rr a	203 31
	rr a	203 31
	rr a	203 31
	and 31	230 31
	ld b,a	71
	and 24	230 24
	ld d,a	87
	ld a,h	124
	and 192	230 192
	ld e,a	95
	ld h, c	97
	ld a,l	125
	rr a	203 31
	rr a	203 31
	rr a	203 31
	and 31	230 31
	ld l,a	111
	ld a,e	123
	add a,b	128
	sub d	146
	ld e,a	95
	ld d, 0	22 0
	push hl	229
	push de	213
	pop hl	225
	add hl,hl	41
	add hl,hl	41

7. Routines om programma's te manipuleren

7.1 Verwijder programmablok

Lengte: 42
Aantal variabelen: 2
Check sum : 5977

Werking

Deze routine verwijdert blokken BASIC-programma, tussen twee door de gebruiker aangegeven regels.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
start line no	2	23296	Eerste te verwijderen regel.
end line no	2	23298	Laatste te verwijderen regel.

Call

RAND USR adres

Foutafhandeling

Als een van de volgende fouten voorkomt, stopt de routine zonder iets van het BASIC-programma te verwijderen:

(I) laatste regelnummer is kleiner dan eerste regelnummer;
(II) er staat geen BASIC-programma tussen de aangegeven twee regels;
(III) één of beide van de aangegeven regels is nul;

Commentaar

Deze routine is tamelijk langzaam om grote blokken programmaregels te verwijderen; desondanks is hij in gebruik veel sneller dan het verwijderen van alle regels met de hand. Geef geen regelnummers in die groter zijn dan 9999.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23296)	42 0 91
	ld de, (23298)	237 91 2 91
	ld a,h	124
	or l	181
	ret z	200
	ld a,d	122
	or e	179
	ret z	200
	push de	213
	call 6510	205 110 25
	ex (sp), hl	227
	inc hl	35
	call 6510	205 110 25

Commentaar
Deze routine is erg snel, maar natuurlijk, hoe langer het BASIC-programma, hoe langer hij erover doet om te draaien.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld, bc, (23296)	237 75 0 91
	ld a,31	62 31
	cp b	184
	ret nc	208
	cp c	185
	ret nc	208
	ld hl, (23635)	42 83 92
next chr:	inc hl	35
	inc hl	35
	inc hl	35
check:	ld de, (23627)	237 91 75 92
	and a	167
	sbc hl,de	237 82
	ret nc	208
	add hl,de	25
	inc hl	35
	ld a, (hl)	126
	inc hl	35
	cp 13	254 13
	jr z, next chr	40 237
	cp 14	254 14
	jr nz, compare	32 3
	inc hl	35
	jr next chr	24 230
compare:	dec hl	43
	cp c	185
	jr nz, check	32 229
	ld (hl), b	112
	jr check	24 226

Hoe het werkt
Registers b en c worden respectievelijk met het nieuwe en het oude teken geladen. Als een van de tekencodes kleiner is dan 32, keert de routine in BASIC terug.
Registerpaar hl wordt geladen met het startadres van het BASIC-programma. hl wordt dan verhoogd en vergeleken met het adres van het variabelengebied. Als hl niet kleiner is dan het adres van de variabelen, keert de routine naar BASIC terug.
hl wordt verhoogd zodat het naar het volgende teken wijst. De code van dit teken wordt in de accumulator geladen, en hl wordt opnieuw verhoogd. Als de waarde van de accumulator 13 of 14 is, (ENTER of NUMBER), springt de routine naar 'next chr' terug en hl wordt verhoogd zodat het naar het volgende teken wijst. Als de accumulator geen 13 of 14 bevat, wordt de opgeslagen waarde vergeleken met 'chr old'. Als er een gelijkenis wordt gevonden, wordt deze door 'chr new' vervangen.
De routine springt dan terug om het einde van het programma te controleren.

	add hl,de	25
	jr next line	24 214
search	inc hl	35
	ld a, (hl)	126
	cp 13	254 13
	jr nz, not enter	32 8
enter found	pop hl	225
	add hl,bc	9
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	jr check	24 231
not enter	cp 14	254 14
	jr nz, not number	32 7
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	jr search	24 231
not number	cp 33	254 33
	jr c, search	56 227
	cp 34	254 34
	jr nz, not quote	32 8
find quote	inc hl	35
	ld a, (hl)	126
	cp 34	254 34
	jr nz, find quote	32 250
	jr search	24 215
not quote	cp 58	254 58
	jr nz, search	32 211
	ld d,h	84
	ld e, l	93
find enter	inc hl	35
	ld a, (hl)	126
	cp 13	254 13
	jr z, enter found	40 209
	cp 33	254 33
	jr c, find enter	56 246
	cp 234	254 234
	jr nz, not quote	32 236
	ld h,d	98
	ld l,e	107
delete chr	push bc	197
	call 4120	205 24 16
	pop bc	193
	dec bc	11
	ld a, (hl)	126
	cp 13	254 13
	jr nz, delete chr	32 245
	pop hl	225

```
inc hl          35
inc hl          35
ld (hl),c       113
inc hl          35
ld (hl),b       112
dec hl          43
dec hl          43
dec hl          43
jr check        24  160
```

Hoe het werkt

Het registerpaar hl wordt geladen met het startadres van het BASIC-gedeelte, en er wordt gesprongen naar de routine die controleert of hij aan het einde van het programma is. Als het einde wordt bereikt, wordt er een return naar BASIC uitgevoerd.

De routine springt naar 'next line'. Deze routine slaat het adres uit hl op de stapel op, voor later gebruik, en laadt bc dan met de lengte van de gevonden regel. De routine 'next chr' verhoogt het adres in hl en laadt de accumulator met het op dat adres opgeslagen teken. Als de code ervan kleiner is dan 33, wat inhoudt dat het een spatie of een controleteken is, springt de routine terug om dit gedeelte te herhalen. Als het gevonden teken niet het REM-token is, wordt er naar 'search' gesprongen.

Als een REM is gevonden, wordt register bc met vier verhoogd, zodat het als een teller kan worden gebruikt, en hl wordt van de stapel opgehaald. Dan worden bc tekens vanaf adres hl verwijderd door middel van de ROM-routine op adres 4120. De routine gaat dan opnieuw door met de 'check'-routine.

Als er een sprong naar 'search' wordt gemaakt, wordt hl verhoogd zodat het naar het volgende teken wijst en dit wordt in de accumulator geladen. Als dit een ENTER-teken is, wordt hl vanuit de stapel herladen, verhoogd zodat het naar het begin van de nieuwe regel wijst, en er wordt een sprong naar 'check' gemaakt.

Als de accumulator het NUMBER-teken bevat (14), wordt hl verhoogd zodat het naar het eerste teken achter het opgeslagen getal wijst, en het zoekproces wordt hervat.

Er wordt dan gecontroleerd op tekens waarvan de codes kleiner zijn dan 33, en als er één gevonden wordt, wordt er naar 'search' teruggesprongen. Als er een aanhalingstekencode (34) wordt gevonden, gaat de routine door tot hij een tweede aanhalingsteken vindt, en dan gaat het zoeken door. Als het gevonden teken niet een dubbele punt is, wat op meerdere opdrachten in één regel wijst, wordt het zoeken herhaald. hl wordt dan naar de gekopieerd om het adres van de dubbele punt te bewaren, en dan wordt hl verhoogd om naar het volgende teken te wijzen. Als dit een ENTER-teken is, wordt er naar 'enter found' gesprongen, anders, als dit een controleteken of een spatie is, gaat de routine naar 'find enter' terug.

Als het teken geen REM-token is, wordt er naar 'not quote' gesprongen. Als er een REM-token wordt gevonden, wordt hl geladen met het adres van de laatst gevonden dubbele punt, en dan worden alle tekens vanaf hl tot het volgende ENTER-teken verwijderd. De pointers voor de regel worden aangepast, hl op het startadres van de regel gezet, en er wordt naar 'check' teruggesprongen.

7.4 REM creëren

Lengte: 85
Aantal variabelen: 3
Check sum: 9526

	inc hl	35
	pop bc	193
next chr	push bc	197
	ld a,b	120
	or c	177
	jr z, insert REM	40 11
	ld a, (23300)	58 4 91
	call 3976	205 136 15
	inc hl	35
	pop bc	193
	dec bc	11
	jr next chr	24 240
insert REM	pop bc	193
	ld a, 234	62 234
	call 3976	205 136 15
	inc hl	35
	pop bc	193
	inc bc	3
	inc bc	3
	ld a,b	120
	push bc	197
	call 3976	205 136 15
	pop bc	193
	inc hl	35
	ld a,c	121
	call 3976	205 136 15
	inc hl	35
	pop bc	193
	ld a,c	121
	push bc	197
	call 3976	205 136 15
	pop bc	193
	inc hl	35
	ld a,b	120
	jp 3976	195 136 15

Hoe het werkt

Registerpaar hl wordt geladen met het aangegeven regelnummer. Dit wordt met nul vergeleken, en als er een gelijkenis wordt gevonden, keert de routine in BASIC terug. Ook als hl een getal bevat dat groter is dan 9999 (het hoogste regelnummer), wordt er een return naar BASIC uitgevoerd.

Er wordt een ROM-routine aangeroepen die in hl het adres teruggeeft van de regel die daarvoor in hl was opgeslagen. Als de nulvlag aan staat, bevindt zich daar een regel, en dus komt de routine in BASIC terug.

Als de nulvlag niet aan staat, wordt er naar 'create' gesprongen. bc wordt geladen met het aantal in te voegen tekens achter de REM-opdracht, en dit getal wordt in de stapel opgeslagen. De accumulator wordt dan met 13 geladen, wat de code is van het ENTER-teken. De ROM-routine op adres 3976 wordt dan aangeroepen om het ENTER-teken in te voegen. Het register bc wordt van de stapel opgehaald. Na bc opnieuw in de stapel op te slaan, wordt bc getest om te kijken of er nog meer tekens zijn die ingevoegd moeten worden. Zo niet, dan wordt er naar 'insert REM' gesprongen. Als er nog een ander teken moet worden ingevoegd, wordt de accumulator met die waarde geladen, en de ROM-routine op adres 3976

wordt gebruikt om het in te voegen. De teller in bc wordt verlaagd en de routine gaat terug om te testen of bc nul is. Als de routine eenmaal 'insert REM' bereikt, wordt een REM-token ingevoegd via dezelfde ROM-routine. bc wordt dan geladen met de lengte van de nieuwe regel, en de pointers voor die regel worden aangemaakt. Het regelnummer wordt dan van de stapel opgehaald en dit wordt tenslotte ingevoegd voordat de routine in BASIC terugkomt.

7.5 Programma samenpakken

Lengte: 71
Aantal variabelen: 0
Check sum: 7158

Werking

Deze routine verwijdert alle onnodige controletekens en spaties uit een BASIC-programma, waardoor hij de beschikbare RAM groter maakt.

Call

RAND USR adres

Foutafhandeling

Als er geen BASIC-programma in het geheugen aanwezig is, keert de routine onmiddellijk in BASIC terug.

Commentaar

Deze routine gaat ervan uit dat alle REM-opdrachten al uit het BASIC-programma zijn weggehaald. Echter, als dit niet het geval is, zal de computer niet 'vast lopen'. De door de routine gebruikte tijd om het BASIC-programma af te ronden is evenredig met de lengte ervan in het geheugen.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23635)	42 83 92
next line	inc hl	35
	inc hl	35
check	ld de, (23627)	237 91 75 92
	and a	167
	sbc hl,de	237 82
	ret nc	208
	add hl,de	25
length	push hl	229
	ld c, (hl)	78
	inc hl	35
	ld b, (hl)	70
next byte	inc hl	35
load	ld a, (hl)	126
	cp 13	254 13
	jr nz, number	32 8
restore	pop hl	225
	ld (hl),c	113
	inc hl	35
	ld (hl),b	112

```
                add hl,bc       9
                inc hl          35
                jr next line    24 227
number          cp 14           254 14
                jr nz, quote    32 7
                inc hl          35
                inc hl          35
                inc hl          35
                inc hl          35
                inc hl          35
                jr next byte    24 231
quote           cp 34           254 34
                jr nz, control  32 12
find quote      inc hl          35
                ld a, (hl)      126
                cp 34           254 34
                jr z, next byte 40 221
                cp 13           254 13
                jr z, restore   40 223
                jr find quote   24 244
control         cp 33           254 33
                jr nc, next byte 48 211
                push bc         197
                call 4120       205 24 16
                pop bc          193
                dec bc          11
                jr load         24 204
```

Hoe het werkt

Registerpaar hl wordt geladen met het adres van het BASIC-programma. hl wordt dan twee keer verhoogd zodat het naar de eerste byte wijst die de lengte van de volgende regel aangeeft. Registerpaar de wordt geladen met het adres van het variabelengebied. Als hl niet kleiner is dan de, keert de routine naar BASIC terug omdat het einde van het programmagebied is bereikt.

Het adres uit hl wordt in de stapel opgeslagen, bc wordt geladen met de lengte van de huidige regel en hl wordt verhoogd zodat het naar de volgende byte in de regel wijst. De byte op hl wordt dan in de accumulator geladen. Als de accumulator niet het getal 13 bevat, wordt er naar 'number' gesprongen.

Om 'restore' te bereiken, moet het einde van de huidige regel zijn gevonden. Het adres van de regel-'pointers' wordt vanuit de stapel in hl geladen, en de huidige lengte ingevoegd. De regellengte wordt bij hl opgeteld, hl wordt verhoogd, en de routine gaat naar 'next line' terug.

Als de routine 'number' bereikt, wordt de accumulator gecontroleerd om te kijken of hij het NUMBER-teken (14) bevat. Als dit het geval is, wordt hl met vijf verhoogd, zodat het volgende getal niet wordt veranderd, en er wordt naar 'next byte' gesprongen.

Als de accumulator niet de code van een aanhalingsteken bevat, gaat de routine naar 'control'. Als er een aanhalingsteken is gevonden, gaat de routine door totdat het einde van de regel wordt bereikt, of tot er een ander aanhalingsteken wordt gevonden. In het eerste geval wordt er naar 'restore' gesprongen, in het laatste naar 'next byte'.

Bij 'control' wordt het teken gecontroleerd om te kijken of het een code kleiner dan 33 betreft. Zo niet, dan gaat de routine naar 'next byte'.
Als er een spatie of een controlcteken is gevonden, wordt de ROM-routine op adres 4120 aangeroepen om die te verwijderen. De regellengte, die in bc zit, wordt verlaagd en er wordt naar 'load' gesprongen.

7.6 Machinetaal laden in DATA-statements

Lengte: 179
Aantal variabelen: 2
Check sum: 19181

Werking
Deze routine maakt een DATA-opdracht op regel één van een BASIC-programma, en vult het dan met data dat uit het geheugen is gePEEKed.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
data start	2	23296	Adres van machinecode.
data length	2	23298	Aantal te kopiëren bytes.

Call
RAND USR adres

Foutafhandeling
Als het aantal te kopiëren bytes nul is, of als regel één reeds bestaat, keert de routine onmiddellijk in BASIC terug. De routine controleert niet of er genoeg vrije ruimte is voor de regel, dus dit moet handmatig gebeuren.
De routine vereist tien bytes per databyte, plus vijf voor regelnummers, pointers, enzovoort. Echter, de gebruikte ROM-routine gebruikt ook een groot stuk werkgeheugen, dus houdt u hier rekening mee. Als er niet genoeg vrije ruimte beschikbaar is, zullen de pointers niet correct worden gezet, en de BASIC-listing zal worden verminkt.

Commentaar
De door de routine gebruikte tijd is evenredig met de lengte van het te kopiëren geheugen.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld de, (23296)	237 91 0 91
	ld bc, (23298)	237 75 2 91
	ld a,b	120
	or c	177
	ret z	200
	ld hl, (23635)	42 83 92
	ld a, (hl)	126
	cp 0	254 0
	jr nz, continue	32 6
	inc hl	35
	ld a, (hl)	126

	cp 1	254 1
	ret z	200
	dec hl	43
continue	push hl	229
	push bc	197
	push de	213
	sub a	151
	call 3976	205 136 15
	ex de,hl	235
	ld a, l	62 1
	call 3976	205 136 15
	ex de,hl	235
	call 3976	205 136 15
	ex de,hl	235
	call 3976	205 136 15
	ex de,hl	235
	ld a, 228	62 228
	call 3976	205 136 15
	ex de,hl	235
next byte	pop de	209
	ld a, (de)	26
	push de	213
	ld c,47	14 47
hundreds	inc c	12
	ld b, 100	6 100
	sub b	144
	jr nc, hundreds	48 250
	add b	128
	ld b,a	71
	ld a,c	121
	push bc	197
	call 3976	205 136 15
	ex de,hl	235
	pop bc	193
	ld a,b	120
	ld c,47	14 47
tens	inc c	12
	ld b,10	6 10
	sub b	144
	jr nc, tens	48 250
	add b	128
	ld b,a	71
	ld a,c	121
	push bc	197
	call 3976	205 136 15
	pop bc	193
	ex de,hl	235
	ld a,b	120
	add 48	198 48
	call 3976	205 136 15
	ex de,hl	235
	ld a,14	62 14
	ld b, 6	6 6

next zero	push bc	197
	call 3976	205 136 15
	pop bc	193
	ex de,hl	235
	sub a	151
	djnz next zero	16 247
	pop de	209
	push hl	229
	dec hl	43
	dec hl	43
	dec hl	43
	ld a, (de)	26
	ld (hl),a	119
	pop hl	225
	inc de	19
	pop bc	193
	dec bc	11
	ld a,b	120
	or c	177
	jr z, enter	40 10
	push bc	197
	push de	213
	ld a, 44	62 44
	call 3976	205 136 15
	ex de, hl	235
	jr next byte	24 173
enter	ld a,13	62 13
	call 3976	205 136 15
	pop hl	225
	ld bc, 0	1 0 0
	inc hl	35
	inc hl	35
	ld d,h	84
	ld e,l	93
	inc hl	35
pointers	inc hl	35
	inc bc	3
	ld a, (hl)	126
	cp 14	254 14
	jr nz, end?	32 12
	inc bc	3
	inc bc	3
	inc bc	3
	inc bc	3
	inc bc	3
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	jr pointers	24 237
end?	cp 13	254 13
	jr nz pointers	32 233
	ld a,c	121

ld (de),a	18
inc de	19
ld a,b	120
ld (de),a	18
ret	201

Hoe het werkt

Het registerpaar de wordt geladen met het adres van de te kopiëren bytes, en het registerpaar bc wordt geladen met het aantal te kopiëren bytes. Als bc nul bevat, keert de routine onmiddellijk in BASIC terug.

Register hl wordt geladen met het adres van het BASIC-programma. De accumulator wordt geladen met de op adres hl opgeslagen byte. Dit is de hoge byte van het regelnummer. Als dit niet nul bevat, bestaat regel één niet en de routine springt dus naar 'continue'. Als de hoge byte nul bevat, wordt de accumulator met de lage byte geladen. Als dit op één staat, bestaat regel één reeds, en dus keert de routine naar BASIC terug.

Het adres van de hoge byte van het regelnummer wordt in de stapel opgeslagen. Het aantal te kopiëren bytes wordt opgeslagen, gevolgd door het adres van de data.

De accumulator wordt dan met nul geladen – de hoge byte van het nieuwe regelnummer. Door de ROM-routine op adres 3976 aan te roepen, wordt het teken dat in de accumulator is opgeslagen op adres hl ingevoegd. hl krijgt de waarde die het voor de bewerking had. De accumulator wordt met één geladen, en dit wordt drie maal ingevoegd. De eerste is de lage byte van het regelnummer, de volgende twee de regel-pointer. De accumulator wordt dan geladen met de code van het 'DATA'-token en deze wordt ingevoegd.

Het adres van de volgende databyte wordt van de stapel opgehaald en in de geladen. De accumulator wordt geladen met deze byte, en de wordt opnieuw in de stapel opgeslagen. Het register c wordt geladen met één minder dan de code van '0'. Register c wordt verhoogd, en b wordt met 100 geladen. B wordt van de accumulator afgetrokken, en als het resultaat niet negatief is, loopt de routine naar 'hundreds' terug.

Register b wordt opnieuw bij de accumulator opgeteld, zodat deze een positieve waarde bevat. Deze waarde wordt dan in het b-register geladen. De accumulator wordt geladen met de inhoud van c, en bc wordt in de stapel opgeslagen. De ROM-routine op adres 3976 voegt het in de accumulator opgeslagen teken in, op het door hl aangegeven adres. Het registerpaar bc wordt van de stapel opgeslagen en de accumulator wordt geladen met de inhoud van het b-register. Dit proces wordt dan herhaald voor b=10. De accumulator wordt dan met 48 verhoogd, en het resulterende teken wordt ingevoegd.

De vorige routine heeft de decimale waarde van de gevonden databyte in een DATA-opdracht ingevoegd. De binaire voorstelling moet nu worden ingevoegd. Dit wordt gemarkeerd door het NUMBER-token, chr 14, dat als eerste wordt ingevoerd, gevolgd door vijf bytes. De waarde van de te kopiëren byte wordt gePOKEd om de derde nul te vervangen. de wordt verhoogd om naar de volgende data-byte te wijzen. Het aantal te kopiëren bytes wordt vanuit de stapel in bc gekopieerd, en dit wordt verlaagd. Als het resultaat nul is, wordt er naar 'ENTER' gesprongen, anders worden register bc en de opnieuw opgeslagen (re-stacked), er wordt een komma in de DATA-opdracht gezet, en de routine gaat naar 'next byte'.

Bij 'enter' wordt een ENTER-token ingevoegd om het einde van de DATA-op-

dracht aan te geven. hl wordt geladen met het adres van de start van de regel, en bc wordt op nul gezet. hl wordt verhoogd zodat het naar de lage byte van de regelpointer wijst, en dit nieuwe adres wordt in de gekopieerd. hl wordt verhoogd om naar de hoge byte van de regel te wijzen en de accumulator wordt geladen met het op adres hl opgeslagen teken.
Als de accumulator de waarde 14 bevat, is er een getal gevonden en zowel hl als bc worden dus met vijf verhoogd zodat ze naar het eerste teken achter het getal wijzen; de routine gaat naar 'pointers'.
Als de accumulator niet de waarde 14 heeft, en ook niet 13, wordt er een terugsprong naar 'pointers' gemaakt.
Om dit stadium te bereiken, moest het ENTER-token zijn gevonden dat het einde van de regel aangeeft. Bc bevat nu de lengte van de regel en dit wordt dus in de regelpointer gePOKEd; het adres van de pointer is in de te vinden.
De routine keert daarna in BASIC terug.

7.7 Omzetten van kleine letter naar hoofdletter

Lengte: 41
Aantal variabelen: 0
Check sum: 4685

Werking
Deze routine zet alle kleine letters van een BASIC-programma om in hoofdletters, of andersom.

Call
RAND USR adres

Foutafhandeling
Als er geen BASIC-programma in het geheugen aanwezig is, keert de routine onmiddellijk in BASIC terug.

Commentaar
Om de routine zodanig te veranderen dat hij van de hoofdletters kleine letters maakt, moet u de aangegeven getallen als volgt veranderen:

97 * in 65
91 ** in 123

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23635)	42 83 92
	ld de, (23627)	237 91 75 92
jump	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
changed	inc hl	35
next byte	and a	167
	sbc hl,de	237 82
	ret nc	208
	add hl,de	25

ld a, (hl)	126
cp 13	254 13
jr z, jump	40 241
cp 14	254 14
inc hl	35
jr z, jump	40 236
sub 97	214 97*
jr c, next byte	56 237
sub 26	214 26
jr nc, next byte	48 233
add a,91	198 91**
dec hl	43
ld (hl), a	119
jr changed	24 226

Hoe het werkt

Het registerpaar hl wordt geladen met het adres van het BASIC-programma en de wordt geladen met het adres van het variabelengebied. hl wordt verhoogd om over de regelnummer/pointers heen te springen. Als hl niet kleiner is dan de, keert de routine in BASIC terug, omdat het einde van het programma(gebied) is bereikt.

De accumulator wordt geladen met de byte op adres hl. Als deze byte een ENTER-teken is, loopt de routine naar 'jump' terug. Als de byte het NUMBER-token is, loopt ook de routine naar 'jump' terug, maar eerst wordt hl verhoogd. Aldus worden de vijf bytes achter het teken 14 vermeden.

Er wordt 96 van de accumulator afgetrokken. Als het resultaat negatief is, loopt de routine naar 'next byte' omdat het teken geen kleine letter kan zijn. Er wordt dan 26 van de accumulator afgetrokken. Als het resultaat negatief is, wordt er naar 'next byte' gesprongen omdat het teken een te hoge code heeft om een kleine letter te zijn. Er wordt dan 19 bij de accumulator opgeteld om de code van de betreffende hoofdletter te verkrijgen. hl wordt verlaagd om naar het teken te wijzen dat vervangen moet worden. Dit adres wordt met de waarde van de accumulator gePOKEd en er wordt naar 'changed' gesprongen.

8. Routines als gereedschap

8.1 Hernummeren

Lengte: 382
Aantal variabelen: 2
Check sum: 41629

Werking

Deze routine hernummert een BASIC-programma met inbegrip van alle GOTO's, GOSUB's, enzovoort.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
first line no	2	23296	Nieuw regelnummer eerste regel.
step	2	23298	Verschil tussen 2 opeenvolgende regels.

Call

RAND USR adres

Foutafhandeling

Als het nummer van de eerste regel nul is, of als step nul is, komt de routine onmiddellijk in BASIC terug. Als er geen BASIC-programma in RAM zit, keert de routine in BASIC terug. Alle berekende goto's (bv. GOTO 7*A) getallen met komma (bv. GOTO 7.8), getallen kleiner dan nul (bv. GOTO −1) of getallen groter dan 9999 (GOTO 20170) worden genegeerd. Als step te groot is, kunnen regelnummers worden herhaald en het programma loopt vast. De routine vergroot de lengte van het BASIC-programma in RAM; er moet dus gecontroleerd worden of er voldoende RAM aanwezig is.

Commentaar

De door de routine gebruikte tijd is evenredig met de lengte van het BASIC-programma in RAM.
De routine is niet verplaatsbaar en hij zou gewoonlijk vanaf adres 32218 ingegeven moeten worden. De locatie ervan kan worden veranderd door de volgende methode:

(I) Let X=nieuw adres−32218
(II) Let H=INT (x/256)
Let L=x-256*h
(III) Voor elk stel getallen, in de lijst met (*) aangegeven:
Let LI=L+eerste getal
Let HI=H+tweede getal
Als LI groter is dan 255 LET HI=HI+1
let LI=LI−256
Vervang het stel getallen door LI en HI

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23296)	42 0 91
	ld a, h	124
	or L	181
	ret z	200
	ld hl, (23298)	42 2 91
	ld a,h	124
	or L	181
	ret z	200
	ld hl, (23635)	42 83 92
	ld de, (23296)	237 91 0 91
next line	call check	205 76* 127*
	jr nc, find GOTO	48 22
	ld b, (hl)	70
	ld (hl),d	114
	inc hl	35
	ld c, (hl)	78
	ld (hl),e	115
	inc hl	35
	ld (hl),c	113
	inc hl	35
	ld (hl),b	112
	inc hl	35
	push hl	229
	ld hl, (23298)	42 2 91
	add hl,de	25
	ex de, hl	235
	pop hl	225
	call end of line	205 65* 127*
	jr next line	24 229
find GOTO	ld hl, (23635)	42 83 92
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
search	call find	205 235* 126*
	jp nc, restore	210 184* 126*
	ld d,h	84
	ld e, l	93
	ld b, 0	6 0
next digit	inc b	4
	inc hl	35
	ld a, (hl)	126
	cp 46	254 46
	jr nz, continue	32 3
find next	ex de,hl	235
	jr search	24 236
continue	cp 14	254 14
	jr nz, next digit	32 242
	inc hl	35
	inc hl	35
	inc hl	35

	inc hl	35
	inc hl	35
	inc hl	35
	ld a, (hl)	126
	cp 58	254 58
	jr z, found	40 4
	cp 13	254 13
	jr nz, find next	32 234
found	ld a, b	120
compare	cp 4	254 4
	jr z, calculate	40 16
	jr nc, find next	48 227
	push de	213
	ld h,d	98
	ld l,e	107
	push af	245
	ld a, 48	62 48
	call 3976	205 136 15
	pop af	241
	inc a	60
	pop de	209
	jr compare	24 236
calculate	ld b,d	66
	ld c,e	75
	push de	213
	ld hl, 0	33 0 0
	ld de, 1000	17 232 3
	call add	205 226* 126*
	ld de, 100	17 100 0
	call add	205 226* 126*
	ld e, 10	30 10
	call add	205 226* 126*
	ld a, (bc)	10
	sub 48	214 48
	ld e, a	95
	add hl,de	25
	ld b,h	68
	ld c,l	77
	ld hl, (23635)	42 83 92
find line	inc hl	35
	inc hl	35
end of prog	call check	205 76* 127*
	jr c, exists	56 3
	pop hl	225
	jr search	24 153
exists	ld a, (hl)	126
	cp c	185
	jr nc, next byte	48 7
	inc hl	35
wrong line	inc hl	35
	call end of line	205 65* 127*
	jr find line	24 235

next byte	inc hl	35
	ld a, (hl)	126
	cp b	184
	jr c, wrong line	56 245
	dec hl	43
	dec hl	43
	ld c, (hl)	78
	dec hl	43
	ld h, (hl)	102
	ld l, c	105
	pop bc	193
	push bc	197
	push hl	229
	ld de, 1000	17 232 3
	call insert	205 212* 126*
	ld de, 100	17 100 0
	call insert	205 212* 126*
	ld e, 10	30 10
	call insert	205 212* 126*
	ld e, 1	30 1
	call insert	205 212* 126*
	inc bc	3
	sub a	151
	ld (bc),a	2
	inc bc	3
	ld (bc),a	2
	inc bc	3
	pop hl	225
	ld a,l	125
	ld (bc),a	2
	inc bc	3
	ld a,h	124
	ld (bc),a	2
	inc bc	3
	sub a	151
	ld (bc),a	2
	pop hl	225
	jp search	195 15* 126*
restore following line	ld hl, (23635)	42 83 92
	inc hl	35
	inc hl	35
	call check	205 76* 127*
	ret nc	208
	ld d,h	84
	ld e,l	93
	inc hl	35
	inc hl	35
	call end of line	205 65* 127*
	push hl	229
	scf	55
	sbc hl,de	237 82
	dec hl	43
	ex de,hl	235
	ld (hl),e	115

	inc hl	35
	ld (hl),d	114
	pop hl	225
	jr following line	24 231
insert	ld a, 48	62 48
subtract	and a	167
	sbc hl,de	237 82
	jr c, poke	56 3
	inc a	60
	jr subtract	24 248
poke	add hl,de	25
	ld (bc),a	2
	inc bc	3
	ret	201
add	ld a, (bc)	10
	inc bc	3
	sub 47	214 47
repeat	dec a	61
	ret z	200
	add hl,de	25
	jr repeat	24 251
find	ld a, (hl)	126
	call check	205 76* 127*
	ret nc	208
	cp 234	254 234
	jr nz not REM	32 13
find ENTER	inc hl	35
	ld a, (hl)	126
	cp 13	254 13
	jr nz, find ENTER	32 250
increase	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	jr find	24 234
not REM	cp 34	254 34
	jr nz, not string	32 9
next character	inc hl	35
	ld a, (hl)	126
	cp 34	254 34
	jr nz, next character	32 250
	inc hl	35
	jr find	24 221
not string	cp 13	254 13
	jr z, increase	40 232
	call 6326	205 182 24
	jr z, find	40 212
	cp 237	254 237
	jr z, check digit	40 27
	cp 236	254 236

	jr z, check digit	40 23
	cp 247	254 247
	jr z, check digit	40 19
	cp 240	254 240
	jr z, check digit	40 15
	cp 229	254 229
	jr z, check digit	40 11
	cp 225	254 225
	jr z, check digit	40 7
	cp 202	254 202
	jr z, check digit	40 3
	inc hl	35
	jr find	24 181
check digit	inc hl	35
	ld a, (hl)	126
	cp 48	254 48
	jr c, find	56 175
	cp 58	254 58
	jr nc, find	48 171
	ret	201
end of line	ld a, (hl)	126
again	call 6326	205 182 24
	jr z, again	40 251
	cp 13	254 13
	inc hl	35
	jr nz, end of line	32 245
check	push hl	229
	push de	213
	ld de, (23627)	237 91 75 92
	and a	167
	sbc hl,de	237 82
	pop de	209
	pop hl	225
	ret	201

Hoe het werkt

hl wordt geladen met het eerste regelnummer. Als dit nul is, komt de routine in BASIC terug. hl wordt dan geladen met step, en als dit nul is, komt de routine in BASIC terug.

hl wordt geladen met het adres van het BASIC-programma, en de wordt geladen met het eerste regelnummer. De subroutine 'check' wordt dan aangeroepen, en als het einde van het BASIC-programma bereikt is, wordt er naar 'find GOTO' gesprongen. bc wordt geladen met het gevonden oude regelnummer, en het getal wordt vervangen door de; bc wordt dan in de regelpointers gekopieerd.

hl wordt op de stapel opgeslagen, met de 'step' geladen, en met de verhoogd. Het resultaat wordt op de gekopieerd, als zijnde het volgende regelnummer. hl wordt dan van de stapel opgehaald, en door de subroutine 'end of line' verhoogd, zodat het naar de volgende regel wijst. De routine gaat dan naar 'next line' terug.

Bij 'find GOTO', wordt hl geladen met het adres van het BASIC-programma en dit wordt zodanig verhoogd dat het naar het eerste teken van de volgende regel wijst. De subroutine 'find' wordt dan aangeroepen. Als er geen GOTO's, GO-SUB's, enzovoort meer zijn die aangepast moeten worden, springt de routine

naar 'restore'. Anders, bij terugkeer uit de subroutine, bevat hl het adres van het eerste getal achter de GOTO, GOSUB, enzovoort. Dit wordt in de gekopieerd, en het b-register wordt op nul gezet. Het b-register wordt gebruikt voor het aantal cijfers van het volgende getal.
Register b wordt verhoogd, en ook hl, zodat deze naar het volgende teken wijst en dit teken wordt dan in de accumulator geladen. Als dit teken een decimale punt is, wordt hl geladen met de, en de routine keert naar 'search' terug, om de volgende GOTO te vinden. Als het teken niet het NUMBER-token is, gaat de routine naar 'next digit'.
hl wordt verhoogd zodat het naar het teken achter NUMBER wijst. Als dit niet een dubbele punt of het ENTER-token is, springt de routine naar 'find next' terug omdat de GOTO die getest wordt een berekende bestemming aangeeft. De accumulator wordt dan met de waarde van het b-register geladen. Als dit vier is, springt de routine naar 'calculate'; als dit hoger is dan vier, wordt er naar 'find next' gesprongen omdat regelnummers hoger dan 9999 ongeldig zijn.
de wordt dan op de stapel opgeslagen, en naar hl gekopieerd. De accumulator wordt op de stapel opgeslagen, en geladen met de code van het nulteken. Deze wordt op adres hl ingevoegd door de ROM-routine op adres 3976. De accumulator wordt dan van de stapel opgehaald en verhoogd. Het bevat dan het nieuwe aantal cijfers in het regelnummer. de wordt van de stapel opgehaald, en de routine gaat naar 'compare'.
Bij 'calculate', wordt het adres uit de, in bc gekopieerd, en dan op de stapel opgeslagen. hl wordt met nul geladen, en de wordt met 1000 geladen. De subroutine 'add' wordt dan aangeroepen om de duizendtallen van de onderzochte regel bij hl op te tellen. Dit wordt dan voor de honderdtallen, tientallen en eenheden herhaald, zodoende wordt hl geladen met het regelnummer. Register bc wordt met het resultaat geladen.
hl wordt geladen met het adres van het BASIC-programma. De subroutine 'check' wordt aangeroepen, en als het einde van het programma is bereikt, wordt hl van de stapel opgehaald en de routine gaat naar 'search', omdat de bestemming van de GOTO niet bestaat. Als de door hl geadresseerde byte minder is dan het c-register, wordt hl verhoogd zodat het naar de volgende regel wijst, en er wordt naar 'find line' gesprongen. Anders wordt hl verhoogd, zodat het naar de volgende byte uit de geteste regel wijst. Als dit kleiner is dan de waarde van register b, wordt er naar 'wrong line' gesprongen.

Om dit stadium te bereiken moet de bestemming van de GOTO zijn gevonden. hl wordt verlaagd om naar de start van de regel te wijzen, en wordt dan met zijn nieuwe regelnummer geladen. bc wordt geladen met het adres uit de stapel, en dan wordt hl op de stapel opgeslagen. bc bevat nu het adres waar het regelnummer naar toe gekopieerd moet worden. de wordt geladen met 1000, en de routine 'insert' wordt aangeroepen. Deze berekent het aantal duizendtallen in hl, telt er 48 bij op om een leesbaar teken te produceren, en deze waarde wordt in bc gePOKEd. bc wordt dan verhoogd zodat het naar het volgende teken wijst. Dit proces wordt voor de honderdtallen, tientallen en eenheden herhaald.
Dan wordt de binaire voorstelling van het getal opgebouwd; bc wordt verhoogd zodat hij naar het teken achter het NUMBER-token wijst en de volgende twee bytes worden met nul gePOKEd. hl wordt van de stapel opgehaald en in de volgende twee bytes gePOKEd. De vijfde byte van het getal wordt dan met nul gePOKEd. hl wordt van de stapel opgehaald en de routine gaat naar 'search' om met proces voor de volgende GOTO te herhalen.

Bij 'restore' wordt hl geladen met het adres van het BASIC-programmagebied en dan twee keer verhoogd zodat het de pointers van de volgende regel adresseert, die werkelijk het oude regelnummer bevatten. De subroutine 'check' wordt aangeroepen, en als het einde van het BASIC-programma is bereikt, keert de routine naar BASIC terug. bc wordt geladen met het adres in hl en de subroutine 'end of line' wordt aangeroepen. Deze komt terug met in hl de waarde van het adres van het volgende ENTER-token plus één. hl wordt op de stapel opgeslagen. bc wordt afgetrokken van hl, en dan wordt hl twee keer verlaagd, met als resultaat de nieuwe regelpointers, die in bc en bc+1 worden gePOKEd. hl wordt van de stapel opgehaald, en er wordt naar 'following line' gesprongen.

Subroutines

Insert:
De accumulator wordt geladen met de code van het nulteken. de wordt van hl afgetrokken, en als het resultaat negatief is, wordt er naar 'poke' gesprongen. Anders wordt de accumulator verhoogd en wordt er naar 'subtract' gesprongen.
Bij 'poke' wordt de bij hl opgeteld om een positief getal te produceren. bc wordt gePOKEd met de waarde uit de accumulator en dan verhoogd zodat het naar de volgende byte wijst. Er wordt dan een return uitgevoerd.

Add:
De accumulator wordt met de door bc geadresseerde byte geladen, en bc wordt verhoogd zodat het naar de volgende byte wijst. Er wordt 47 van de accumulator afgetrokken. De accumulator wordt verlaagd, en als het resultaat nul is, wordt er een return uitgevoerd. Anders wordt de bij hl opgeteld, en de routine gaat naar 'repeat'.

Find:
De accumulator wordt met de door hl geadresseerde byte geladen. De subroutine 'check' wordt dan aangeroepen, en als het einde van het BASIC-programma is bereikt, wordt er een return uitgevoerd. Als het teken in de accumulator niet het REM-token is, wordt er naar 'not REM' gesprongen. hl wordt herhaaldelijk verhoogd totdat het einde van de regel wordt gevonden. hl wordt verhoogd zodat het naar het eerste teken van de volgende regel wijst, en er wordt naar 'find' gesprongen.
Bij 'not REM' wordt er, als de accumulator niet de code van het aanhalingsteken bevat, naar 'not string' gesprongen. Anders wordt hl herhaaldelijk verhoogd, totdat het symbool van het tweede aanhalingsteken wordt gevonden. hl wordt nogmaals verhoogd zodat het naar het volgende teken wijst, en er wordt naar 'find' teruggesprongen.
Bij 'not string' wordt er, als de accumulator het ENTER-token bevat, een lus naar 'increase' gemaakt. Als hij het NUMBER-token bevat, gaat de routine naar 'find'. Als er geen GOTO-, GOSUB-, RUN-, LIST-, RESTORE-, LLIST-, LINE-opdrachten worden gevonden, wordt hl verhoogd en er wordt naar 'find' gesprongen. hl wordt verhoogd en de accumulator wordt geladen met het volgende teken. Als dit niet binnen het bereik 48-57 ligt, gaat de routine naar 'find'. De routine komt dan terug.

End of line:
De accumulator wordt met de door hl geadresseerde byte geladen. Als deze het

NUMBER-token is, wordt hl verhoogd en er wordt een lus naar 'again' gemaakt. Hl wordt verhoogd. Als de accumulator niet het ENTER-token bevat, springt de routine naar 'end of line'. Er wordt een test uitgevoerd om te kijken of het einde van het BASIC-programma is bereikt, en de routine komt dan terug.

8.2 Geheugen vrij

Lengte: 14
Aantal variabelen: 0
Check sum: 1443

Werking
Deze routine komt terug met de hoeveelheid vrije RAM, in bytes.

Call
PRINT USR adres

Foutafhandeling
Geen

Commentaar
Deze routine zou aangeroepen moeten worden voordat men routines gaat gebruiken die de lengte van het programma kunnen vergroten, om er zeker van te zijn dat genoeg vrije RAM aanwezig is.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, 0	33 0 0
	add hl, sp	57
	ld de, (23653)	237 91 101 92
	and a	167
	sbc hl,de	237 82
	ld b,h	68
	ld c,l	77
	ret	201

Hoe het werkt
Registerpaar hl wordt op nul gezet, en het adres van het einde van de vrije RAM wordt erbij opgeteld (het adres wordt in sp opgeslagen). Registerpaar de wordt geladen met het adres van de start van de vrije RAM, en wordt van hl afgetrokken. hl wordt in bc gekopieerd en de routine komt terug in BASIC.

8.3 Programmalengte

Lengte: 13
Aantal variabelen: 0
Check sum: 1544

Werking
Deze routine komt terug met de lengte van het BASIC-programma, in bytes.

Call
PRINT USR adres

Foutafhandeling
Geen

Commentaar
Geen

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23627)	42 75 92
	ld de, (23635)	237 91 83 92
	and a	167
	sbc hl,de	237 82
	ld b,h	68
	ld c,l	77
	ret	201

Hoe het werkt
Het hl-register wordt geladen met het adres van het variabelengebied, en het registerpaar de wordt geladen met het adres van het BASIC-programma. de wordt van hl afgetrokken, wat de lengte van het programma oplevert. hl wordt naar bc gekopieerd, en de routine komt in BASIC terug.

8.4 Adres programmaregel
Lengte: 29
Aantal variabelen: 1
Check sum: 2351

Werking
Deze routine komt terug met het adres van het eerste teken achter het 'REM'-token in een aangegegeven regel.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
line number	2	23296	Regelnummer waar 'REM' zou zitten.

Call
LET A = USR adres.

Foutafhandeling
Als de gespecificeerde regel niet bestaat of er is geen REM-opdracht aanwezig, geeft de routine de waarde nul terug.

Commentaar
Deze routine kan worden gebruikt om het adres te vinden waar machinetaal zou moeten worden gePOKEd zodat dit in een REM-opdracht zou komen te zitten. Eenmaal aangeroepen, wordt de variabele A (of een andere) geladen met het adres, of met nul als er een fout plaatsvindt. Geef geen regelnummers groter dan 9999.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23296)	42 0 91
	ld a,h	124
	or l	181
	jr z, error	40 5
	call 6510	205 110 25
	jr z, continue	40 4
error	ld bc, 0	1 0 0
	ret	201
continue	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	ld a, 234	62 234
	cp (hl)	190
	jr nz, error	32 243
	inc hl	35
	ld b,h	68
	ld c,l	77
	ret	201

Hoe het werkt

Registerpaar hl wordt geladen met het gespecificeerde regelnummer. Als dit getal nul is, wordt er naar 'error' gesprongen; anders wordt, bij terugkomst uit de subroutine, de ROM-routine op adres 6510 aangeroepen. hl wordt geladen met het adres van de regel. Als de nulvlag aan staat, wordt er naar 'continue' gesprongen. Als de nulvlag niet aan staat, bestaat de regel niet, en de routine gaat door naar 'error', waar bc met nul wordt geladen en de routine komt in BASIC terug.
Als de routine 'continue' bereikt, wordt hl met vier verhoogd zodat hij naar de eerste opdracht in de gespecificeerde regel wijst. Als deze instructie niet de waarde 234 bevat, wordt er naar 'error' gesprongen. Als de instructie een 'REM' is, wordt hl verhoogd zodat hij naar het volgende teken wijst. De waarde uit hl wordt in bc gekopieerd en de routine komt in BASIC terug.

8.5 Geheugen kopiëren

Lengte: 33
Aantal variabelen: 3
Check sum: 4022

Werking

Deze routine kopieert een gedeelte van het geheugen naar een andere plaats.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
Start	2	23296	Beginadres.
destination	2	23298	Bestemmingsadres.
length	2	23300	Aantal te kopiëren bytes.

Call
RAND USR adres

Foutafhandeling
Geen

Commentaar
Deze routine kan worden gebruikt om bewegende 'films' te maken, door de volgende methode te gebruiken:

(I) maak het eerste scherm met informatie klaar
(II) kopieer het display boven RAMTOP
(III) herhaal voor meerdere schermen
(IV) kopieer de display's één voor één in snel tempo terug

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23296)	42 0 91
	ld de, (23298)	237 91 2 91
	ld bc, (23300)	237 75 4 91
	ld a,b	120
	or c	177
	ret z	200
	and a	167
	sbc hl,de	237 82
	ret z	200
	add hl,de	25
	jr c, lddr	56 3
	ldir	237 176
	ret	201
lddr	ex de,hl	235
	add hl,bc	9
	ex de,hl	235
	add hl,bc	9
	dec hl	43
	dec de	27
	lddr	237 184
	ret	201

Hoe het werkt
hl wordt geladen met het adres van de eerste te kopiëren byte in het geheugen, register de wordt geladen met het adres waarheen gekopieerd wordt, en bc wordt geladen met het aantal te kopiëren bytes. Als bc nul is, of hl=de, komt de routine in BASIC terug. Als hl groter is dan de, wordt het gedeelte van het geheugen door middel van de 'ldir'-instructie gekopieerd, en de routine keert dan in BASIC terug.
Als de groter is dan hl, wordt bc-1 bij beide registers opgeteld, het geheugen wordt door middel van de 'lddr'-instructie gekopieerd, en de routine komt in BASIC terug.

8.6 Variabelen op nul stellen

Lengte: 108
Aantal variabelen: 0
Check sum: 10717

Werking

Alle numerieke variabelen krijgen de waarde nul, alle gedimensioneerde strings worden met spaties gevuld, en niet-gedimensioneerde strings krijgen de lengte nul (nul-strings).

Call

RAND USR adres

Foutafhandeling

Als er geen variabelen in het geheugen zijn, komt de routine onmiddellijk in BASIC terug.

Commentaar

Deze routine is een bruikbare hulp bij het foutvrij maken van programma's (debugging).

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23627)	42 75 92
check	ld a, (hl)	126
	cp 128	254 128
	ret z	200
	ld de, 1	17 1 0
	bit 7,a	203 127
	jr nz, next bit	32 32
	bit 5,a	203 111
	jr z, string	40 9
zero	ld b,5	6 5
next byte	inc hl	35
	ld (hl),d	114
	djnz next byte	16 252
	add hl,de	25
	jr check	24 232
string	inc hl	35
	ld c,(hl)	78
	ld (hl),d	114
	inc hl	35
	ld b,(hl)	70
	ld (hl),d	114
	inc hl	35
delete	ld a,b	120
	or c	177
	jr z, check	40 221
	push bc	197
	call 4120	205 24 16
	pop bc	193

```
                dec bc              11
                jr delete           24 244
next bit        bit 6,a             203 119
                jr nz, bit 5        32 45
                bit 5,a             203 111
                jr z, array         40 7
number          inc hl              35
                bit 7, (hl)         203 126
                jr z, number        40 251
                jr zero             24 213
array           sub a               151
find length     push af             245
                inc hl              35
                ld c, (hl)          78
                inc hl              35
                ld b, (hl)          70
                inc hl              35
                push hl             229
                ld l, (hl)          110
                ld h, d             98
                add hl,hl           41
                pop de              209
find elements   inc de              19
                dec bc              11
                dec hl              43
                ld a,h              124
                or l                181
                jr nz, find elements 32 249
                dec bc              11
rub out         inc de              19
                dec bc              11
                pop af              241
                push af             245
                ld (de),a           18
                ld a,b              120
                or c                177
                jr nz, rub out      32 247
                pop af              241
restore         inc de              19
                ex de,hl            235
                jr check            24 164
bit 5           bit 5,a             203 111
                jr z, string array  40 5
                ld de, 14           17 14 0
                jr 'zero'           24 170
string array    ld a, 32            62 32
                jr find length      24 210
```

Hoe het werkt

Registerpaar hl wordt geladen met het startadres van het variabelengebied. De accumulator wordt geladen met de op adres hl opgeslagen byte. Als de waarde van deze byte 128 is, keert de routine in BASIC terug, omdat de code 128 het

einde van de variabelen aangeeft. Register de wordt met 1 geladen voor later gebruik in de routine. Als bit 7 van de accumulator op één wordt gezet, springt de routine naar 'next bit'; daarna springt, als bit 5 op nul wordt gezet, de routine naar 'string'.
Om 'zero' te bereiken zonder naar voren in de routine te springen, moet de gevonden variabele een getal zijn waarvan de naam uit één letter bestaat. Register b wordt op vijf gezet om als teller te worden gebruikt, hl wordt verhoogd zodat hij naar de volgende byte wijst en deze wordt met nul gePOKEd. De teller wordt verlaagd en als het nog niet nul is springt de routine naar 'next byte' terug. De wordt dan bij hl opgeteld zodat deze naar de volgende variabele wijst en er wordt naar 'check' gesprongen.
Als de routine 'string' bereikt, wordt hl verhoogd zodat hij naar de bytes wijst die de lengte van de gevonden string bevatten. De oude lengte wordt bij bc geladen om als teller te worden gebruikt, en de nieuwe lengte wordt op nul gezet. hl wordt opnieuw verhoogd zodat het naar de tekst van de string wijst. Als de teller op nul wordt gezet, wijst hl nu naar de volgende variabele en dus wordt er naar 'check' gesprongen. Zo niet, dan wordt bc op de stapel opgeslagen en de ROM-routine op adres 4120 wordt aangeroepen om een teken te verwijderen. De teller wordt dan van de stapel opgehaald, verlaagd, en er wordt naar 'delete' teruggesprongen.
Bij 'next bit' wordt bit 6 van de accumulator bekeken. Als deze op één is gezet, wordt er naar 'bit 5' gesprongen omdat er een string array of een FOR/NEXT-controlevariabele is gevonden. Als deze op nul is gezet, en als bit vijf op nul wordt gezet, wordt er naar 'array' gesprongen.
Om 'number' te bereiken moest de gevonden variabele een getal zijn met een naam die langer is dan één teken. Register hl wordt dan verhoogd zodat het naar de volgende byte wijst, en dit wordt herhaald totdat er een byte gevonden wordt waarvan bit zeven op één is gezet. Wanneer dit wordt gevonden, springt de routine naar 'zero' om de variabele met niets te laden.
Als de routine 'array' bereikt, wordt de accumulator met nul geladen, omdat dit de waarde is die de elementen later moeten krijgen.
Bij 'find length' wordt de accumulator op de stapel opgeslagen en hl wordt verhoogd zodat het naar de bytes wijst die de lengte van de array bevatten. Dit wordt naar bc gekopieerd om als teller te worden gebruikt. hl wordt opnieuw verhoogd zodat het nu naar de byte wijst die het aantal dimensies bevat, en dan wordt hl op de stapel opgeslagen. hl wordt geladen met het aantal dimensies en dit wordt met twee vermenigvuldigd. de krijgt de inhoud van het in de stapel opgeslagen adres, de wordt dan hl keer verhoogd en bc wordt (hl+1) keer verlaagd. de wordt dan verhoogd en bc opnieuw verlaagd. de wijst nu naar het volgende element uit de array en bc bevat het aantal overgebleven bytes voordat het einde wordt bereikt. De accumulator wordt van de stapel opgehaald en deze wordt in de gePOKEd. De teller in bc wordt verlaagd en als deze niet nul bevat, gaat de routine naar 'rub out' terug. De waarde in hl wordt aangepast zodat hij naar de volgende variabele wijst, en er wordt naar 'check' gesprongen.
Bij 'bit 5' wordt er gekeken of er een string array werd gevonden. Zo ja, dan krijgt de accumulator de code van een spatie en er wordt naar 'find length' gesprongen. Om dit punt te bereiken moet de variabele een FOR/NEXT-controlevariabele zijn. De wordt op 14 gezet, zodat als dit bij (hl+5) wordt opgeteld, deze naar de volgende variabele wijst. De routine springt dan terug naar 'zero'.

8.7 Variabelenlijst

Lengte: 94
Aantal variabelen: 0
Check sum: 10295

Werking

Deze routine lijst de namen van alle in het geheugen aanwezige variabelen uit.

Call

RAND USR adres

Foutafhandeling

Als er geen variabelen in het geheugen zijn, komt de routine onmiddellijk in BASIC terug.

Commentaar

Dit is een bruikbare hulp voor het foutvrij maken van lange en/of gecompliceerde programma's.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	res 0, (IY + 2)	253 203 2 134
	ld hl, (23627)	42 75 92
next variable	ld a, 13	62 13
	rst 16	215
	ld a, 32	62 32
	rst 16	215
	ld a, (hl)	126
	cp 128	254 128
	ret z	200
	bit 7,a	203 127
	jr z, bit 5	40 62
	bit 6,a	203 119
	jr z, next bit	40 31
	bit 5,a	203 111
	jr z, string array	40 9
	sub 128	214 128
	ld de, 19	17 19 0
print	rst 16	215
	add hl,de	25
	jr next variable	24 225
string array	sub 96	214 96
	rst 16	215
	ld a, 36	62 36
brackets	rst 16	215
	ld a, 40	62 40
	rst 16	215
	ld a, 41	62 41
pointers	inc hl	35
	ld e, (hl)	94
	inc hl	35

	ld d, (hl)	86
	inc hl	35
	jr print	24 234
next bit	bit 5,a	203 111
	jr z, array	40 19
	sub 64	214 64
	rst 16	215
next character	inc hl	35
	ld a, (hl)	126
	bit 7,a	203 127
	jr nz, last character	32 3
	rst 16	215
	jr next character	24 247
last character	sub 128	214 128
jump	ld de, 6	17 6 0
	jr print	24 211
array	sub 32	214 32
	jr brackets	24 216
bit 5	bit 5,a	203 111
	jr nz, jump	32 243
	add a, 32	198 32
	rst 16	215
	ld a, 36	62 36
	jr pointers	24 211

Hoe het werkt

Bit 0 van de byte op adres 23612 wordt ge-reset om er zeker van te zijn dat elk te PRINTen teken op het bovenste gedeelte van het scherm verschijnt. hl wordt geladen met het adres van het variabelengebied. De accumulator wordt geladen met het ENTER-token en dit wordt via de ROM-routine op adres 16 gePRINT. De accumulator wordt dan geladen met de code van een spatie en deze wordt via dezelfde routine gePRINT.

De accumulator wordt geladen met de op adres hl opgeslagen byte. Als de waarde hiervan 128 is, keert de routine naar BASIC terug omdat het einde van het variabelengebied is bereikt.

Als bit 7 van de accumulator op nul wordt gezet, gaat de routine naar 'bit 5' omdat er een string is gevonden, of een getal waarvan de naam slechts één letter lang is. Bit 6 van de accumulator wordt bekeken. Als dit op nul is gezet, wordt er naar 'next bit' gesprongen omdat er een array werd gevonden, of een getal waarvan de naam langer is dan één letter. Als bit 5 van de accumulator nul is, gaat de routine naar 'string array'.

De routine bereikt dit punt als de gevonden variabele een controlevariabele van een FOR/NEXT-lus is; er wordt 128 van de accumulator afgetrokken, wat resulteert in de code van het te PRINTen teken. de wordt met 19 geladen zodat het naar de volgende variabele wijst, wanneer dit bij hl wordt opgeteld; het teken in de accumulator wordt gePRINT, de wordt bij hl opgeteld, en de routine gaat naar 'next variable' terug.

Als de routine 'string array' bereikt, wordt 96 van de accumulator afgetrokken om de code van de naam van de gevonden array te krijgen. Deze wordt via de ROM-routine gePRINT. Een dollar-teken en een open haakje worden dan gePRINT.

en de accumulator wordt met de code voor het gesloten haakje geladen. hl wordt verhoogd zodat het naar de bytes wijst die de lengte van de array bevatten. Deze wordt in de geladen, zodat als het bij hl wordt opgeteld, dit het adres geeft van de volgende variabele. Er wordt naar 'print' gesprongen waar het gesloten haakje wordt gePRINT en de wordt bij hl opgeteld.
Bij 'next bit' wordt bit 5 van de accumulator getest. Als deze op nul is gezet, wordt er naar 'array' gesprongen. Als deze op één is gezet, is er een getal gevonden waarvan de naam langer is dan één letter. Er wordt 64 van de accumulator afgetrokken en het resulterende teken wordt gePRINT. De routine gaat draaien, terwijl ieder teken wordt gePRINT, totdat bit 7 op één wordt gevonden. Er wordt 128 van de tekencode afgetrokken, de wordt geladen met de verplaatsingfactor naar de volgende variabele, en de routine springt naar 'print'.
Als er een array wordt gevonden, wordt er 32 van de accumulator afgetrokken om de correcte code te geven en er wordt naar 'brackets' gesprongen.
Bij 'bit 5' gaat, als er een getal is gevonden waarvan de naam slechts één letter is, de routine naar 'jump' terug.
Om tot dit gedeelte te komen, moet de gevonden variabele een string zijn. Door 32 van de accumulator af te trekken, wordt de code van het te PRINTen teken verkregen. De accumulator wordt tenslotte met de code van het dollar-teken geladen, en er wordt naar 'pointers' gesprongen.

8.8 Opzoeken en uitlijsten

Lengte: 155
Aantal variabelen: 2
Check sum: 17221

Werking

Deze routine zoekt een BASIC-programma door en lijst alle regels uit die een door de gebruiker aangegeven string van tekens bevatten.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
data start	2	23296	Adres eerste data-byte.
string length	1	23298	Aantal tekens in string.

Call

RAND USR adres

Foutafhandeling

Als er geen BASIC-programma in het geheugen aanwezig is of als de lengte van de string nul is, keert de routine onmiddellijk in BASIC terug.

Commentaar

De door de routine gebruikte tijd is evenredig aan zowel de lengte van de string als aan die van het BASIC-programma.
De te zoeken string moet boven RAMTOP worden gePOKEd en het adres van de eerste byte ervan moet in 23296/7 worden gePOKEd. De lengte van de string moet opgeslagen worden op 23298.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	res 0, (IY + 2)	253 203 2 134
	ld ix, (23296)	221 42 0 91
	ld hl, (23635)	42 83 92
restart	ld a, (23298)	58 2 91
	ld e, a	95
	cp 0	254 0
	ret z	200
	push hl	229
restore	push ix	221 229
	pop bc	193
	ld d, 0	22 0
	inc hl	35
	inc hl	35
	inc hl	35
check	inc hl	35
	push de	213
	ld de, (23627)	237 91 75 92
	and a	167
	sbc hl,de	237 82
	add hl,de	25
	pop de	209
	jr c, enter	56 4
	pop hl	225
	ret	201
long jump	jr restart	24 223
enter	ld a, (hl)	126
	cp 13	254 13
	jr nz, number	32 5
	inc hl	35
	pop bc	193
	push hl	229
	jr restore	24 221
number	call 6326	205 182 24
	jr nz, compare	32 8
	dec hl	43
different	push ix	221 229
	pop bc	193
	ld d, 0	22 0
	jr check	24 216
compare	ld a, (bc)	10
	cp (hl)	190
	jr nz, different	32 245
	inc bc	3
	inc d	20
	ld a,d	122
	cp e	187
	jr nz, check	32 206
	ld a, 13	62 13
	rst 16	215
	pop hl	225

	push hl	229
	ld b, (hl)	70
	inc hl	35
	ld l, (hl)	110
	ld h, b	96
	ld de, 1000	17 232 3
	ld a, 47	62 47
thousands	inc a	60
	and a	167
	sbc hl,de	237 82
	jr nc, thousands	48 250
	add hl, de	25
	rst 16	215
	ld de, 100	17 100 0
	ld a, 47	62 47
hundreds	inc a	60
	and a	167
	sbc hl, de	237 82
	jr nc, hundreds	48 250
	add hl,de	25
	rst 16	215
	ld de, 10	17 10 0
	ld a, 47	62 47
tens	inc a	60
	and a	167
	sbc hl,de	237 82
	jr nc, tens	48 250
	add hl, de	25
	rst 16	215
	ld a,l	125
	add a, 48	198 48
	rst 16	215
	pop hl	225
	inc hl	35
	inc hl	35
	inc hl	35
next character	inc hl	35
	ld a, (hl)	126
line end	cp 13	254 13
	jr nz, chr 14	32 4
	rst 16	215
	inc hl	35
	jr long jump	24 155
chr 14	call 6326	205 182 24
	jr z, line end	40 243
	cp 32	254 32
	jr c, next character	56 237
	rst 16	215
	jr next character	24 234

Hoe het werkt

Bit 0 van de op adres 23612 opgeslagen byte wordt ge-reset om er zeker van te zijn dat de te PRINTen tekens in het bovenste gedeelte van het scherm verschijnen. ix

wordt geladen met het adres van de eerste data-byte. Dit maakt het mogelijk het adres in andere registerparen te laden, met minder gebruik van referenties naar de printerbuffer. hl wordt geladen met het adres van het BASIC-programma. De accumulator wordt geladen met de lengte van de string en dit wordt in register e gekopieerd. Als de lengte ervan nul is, keert de routine onmiddellijk in BASIC terug. Het adres in hl wordt op de stapel opgeslagen; deze bevat het adres van de huidige regel die wordt onderzocht.
Het adres van de data wordt vanaf ix naar bc gekopieerd zodat het meer toegankelijk is. Register d wordt met nul geladen, dat wil zeggen het aantal gevonden tekens die overeenkomen met de aangegeven data. Het registerpaar hl wordt met drie verhoogd zodat het naar de hoge byte van het regelnummer wijst. hl wordt verhoogd zodat het naar het volgende teken wijst. Het registerpaar de wordt op de stapel opgeslagen.
de wordt geladen met het adres van het variabelengebied en dit wordt van hl afgetrokken. Als het resultaat negatief is, gaat de routine naar 'enter', na het herstellen van hl en na register de van de stapel op te halen. Als het resultaat positief was, krijgt de stapel zijn oorspronkelijke waarde en de routine komt in BASIC terug, omdat het einde van het BASIC-programma is bereikt.
Bij 'enter' wordt de accumulator geladen met de op hl opgeslagen byte. Als dit niet het ENTER-token is, wordt er naar 'number' gesprongen. Als het ENTER-token wordt gevonden, wordt hl verhoogd zodat het naar de start van de volgende regel wijst. Het adres van de vorige regel wordt van de stapel weggehaald en vervangen door de nieuwe waarde uit hl. Dan wordt er naar 'restore' gesprongen.
Bij 'number' wordt de ROM-routine op adres 6326 aangeroepen. Als het teken in de accumulator het NUMBER-token is, wordt hl verhoogd zodat het naar het eerste teken wijst achter de binaire voorstelling van het door de ROM-routine gevonden getal. Als het NUMBER-token niet wordt gevonden, gaat de routine naar 'compare', anders wordt hl verlaagd en de routine gaat door naar 'different'. bc wordt vanaf ix gekopieerd, het gevonden aantal tekens wordt op nul ge-reset, en er wordt naar 'check' gesprongen.
Bij 'compare' wordt de accumulator geladen met de byte waarvan het adres in bc is opgeslagen. Als deze niet dezelfde is als de in adres hl opgeslagen byte, gaat de routine naar 'different' terug. bc wordt verhoogd zodat het naar de volgende data-byte wijst, en het aantal gevonden tekens wordt verhoogd. Als dit niet gelijk is aan de lengte van de string, gaat de routine naar 'check' terug.
De accumulator wordt geladen met het ENTER-token en dit wordt via de ROM-routine op adres 16 gePRINT. Het adres van de te PRINTen regel wordt vanaf de stapel in hl geladen. Het regelnummer wordt dan via register b, in hl gekopieerd. Register de wordt met 1000 geladen, en de accumulator wordt geladen met één minder dan de code van het nulteken. De accumulator wordt verhoogd en de wordt herhaaldelijk van hl afgetrokken totdat hl negatief wordt. Dan wordt de nogmaals bij hl opgeteld om een positieve rest te produceren. Het teken in de accumulator wordt dan gePRINT.
Het proces hierboven wordt dan voor de=100 en de=10 herhaald. Daarna wordt de rest in de accumulator geladen, er wordt 48 bij opgeteld, en het resulterende teken wordt gePRINT.
Het startadres van de regel wordt van de stapel opgehaald en in hl geladen. Dan wordt hl verhoogd zodat het naar de hoge byte van de regelpointer wijst, hl wordt verhoogd, en de byte op adres hl wordt in de accumulator geladen. Als dit niet het ENTER-token is, wordt er naar 'chr 14' gesprongen, anders wordt ENTER gePRINT, hl verhoogd, en er wordt naar 'restart' teruggesprongen.

Bij 'chr 14' wordt de ROM-routine op adres 6326 aangeroepen. Als het teken in de accumulator het NUMBER-token is, wordt hl verhoogd zodat het naar het eerste teken achter het gevonden getal wijst, dit teken wordt in de accumulator geladen en er wordt naar 'line end' gesprongen. Dan, als het teken in de accumulator een code kleiner dan 32 heeft, gaat de routine naar 'next character' terug. Als de code groter is dan 31, wordt het gevonden teken gePRINT en er wordt naar 'next character' gesprongen.

8.9 Zoeken en vervangen

Lengte: 85
Aantal variabelen: 3
Check sum: 8518

Werking

Deze routine zoekt een BASIC-programma voor een string van tekens door en vervangt het, elke keer dat deze gevonden wordt, door een andere string van dezelfde lengte.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
old data start	2	23296	Adres van de te vervangen string.
string length	1	23298	Lengte van de te vervangen string.
new data start	2	23299	Adres vervangende string.

Call

RAND USR adres

Foutafhandeling

Als de lengte van de string nul is of als er geen BASIC-programma in het geheugen aanwezig is, keert de routine onmiddellijk in BASIC terug.

Commentaar

De door de routine gebruikte tijd is afhankelijk van de lengte van de string en van de lengte van het BASIC-programma in het geheugen.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld ix, (23296)	221 42 0 91
	ld hl, (23635)	42 83 92
	ld a, (23298)	58 2 91
	ld e, a	95
	cp 0	254 0
	ret z	200
	dec hl	43
new line	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	jr reset	24 23

check	inc hl	35
	push de	213
	ld de, (23627)	237 91 75 92
	and a	167
	sbc hl,de	237 82
	add hl,de	25
	pop de	209
	ret nc	208
	ld a, (hl)	126
	cp 13	254 13
	jr z, new line	40 233
	call 6326	205 182 24
	jr nz, compare	32 8
	dec hl	43
reset	push ix	221 229
	pop bc	193
	ld d, 0	22 0
	jr check	24 226
compare	ld a, (bc)	10
	cp (hl)	190
	jr nz, reset	32 245
	inc bc	3
	inc d	20
	ld a,d	122
	cp e	187
	jr nz, check	32 216
	push hl	229
	ld d, 0	22 0
	and a	167
	sbc hl,de	237 82
	ld d,e	83
	ld bc, (23299)	237 75 3 91
	inc d	20
next char	inc hl	35
	dec d	21
	jr z, finish	40 5
	ld a, (bc)	10
	ld (hl),a	119
	inc bc	3
	jr next char	24 247
finish	pop hl	225
	jr reset	24 215

Hoe het werkt

ix wordt geladen met het adres van de string die gezocht wordt. Deze moet boven RAMTOP zitten. hl wordt geladen met het adres van het programmagebied, en de accumulator wordt geladen met de lengte van de string. De lengte wordt naar register e gekopieerd voor later gebruik in het programma. Als de string-lengte nul is, komt de routine in BASIC terug. hl wordt aangepast zodat het naar de hoge byte van de volgende BASIC-regel wijst en er wordt naar 'reset' gesprongen. Bij 'check' wordt hl verlaagd zodat het naar het volgende teken wijst. de wordt op de stapel opgeslagen, en geladen met het adres van het variabelengebied. Als hl

niet kleiner is dan register de, is het einde van het programma bereikt en dus keert de routine na de van de stapel op te halen, in BASIC terug.
De accumulator wordt geladen met het door hl geadresseerde teken. Als dit het ENTER-token is, gaat de routine naar 'new line' terug. Als de accumulator niet het NUMBER-token bevat (teken 14) wordt er naar 'compare' gesprongen, anders wordt hl met vijf verhoogd, zodat hl naar de vijfde byte van het gevonden getal wijst.
Bij 'reset' wordt bc geladen met het adres van de gezochte string. Register d wordt op nul gezet om het aantal tot dan toe gevonden tekens in de string te bevatten. De routine gaat dan naar 'check' terug.
Bij 'compare' wordt de accumulator geladen met het teken uit de string dat door bc wordt aangeduid. Als dit anders is dan de door hl geadresseerde byte, gaat de routine naar 'reset'. bc wordt verhoogd zodat het naar het volgende teken in de string wijst, en de teller in d wordt verhoogd. Als dit niet gelijk is aan de lengte van de string, gaat de routine naar 'check' terug.
Als de string is gevonden, wordt hl op de stapel opgeslagen zodat de routine vanaf dit adres verder zoekt naar de volgende keer dat de string voorkomt. de wordt geladen met de lengte van de string en dit wordt van hl afgetrokken om als resultaat het startadres min één te krijgen. De lengte van de string wordt in d geladen om als teller te gebruiken. bc wordt geladen met het startadres van de nieuwe string en register d wordt verhoogd. Register hl wordt verhoogd zodat het naar de volgende locatie wijst en de teller wordt verlaagd. Als de teller nul bevat, wordt hl van de stapel opgehaald en er wordt naar 'reset' gesprongen om te kijken of de string nogmaals voorkomt. De accumulator wordt geladen met het door bc aangegeven teken en dit wordt in hl gePOKEd. bc wordt verhoogd om naar het volgende teken te wijzen en de routine gaat naar 'next char' terug.

8.10 ROM doorzoeken

Lengte: 58
Aantal variabelen: 3
Check sum: 6533

Werking

Deze routine zoekt de ROM door naar een door de gebruiker aangegeven bytepatroon.

Variabelen

Naam	*Lengte*	*Locatie*	*Commentaar*
start search	2	23296	Startpositie van te doorzoeken geheugen.
string length	1	23298	Aantal bytes in string.
data start	2	23299	String-adres in RAM.

Call

PRINT USR adres

Foutafhandeling

Als de lengte van de string nul is, komt de routine onmiddellijk in BASIC terug; daarbij geeft hij het adres van de start van de data terug. Als de string niet gevonden wordt geeft hij de waarde 65535 terug.

Commentaar
Wanneer men bezig is machinetaalprogramma's te schrijven, kan deze routine worden gebruikt om naar subroutines in de ROM te zoeken, als de gebruiker reeds weet hoe een gedeelte van de routine is geschreven.
Aangezien het grootste deel van de Spectrum-ROM een aanpassing is van dat van de ZX81, kunnen programma's die oorspronkelijk voor de ZX81 waren geschreven, makkelijk worden aangepast. Bijvoorbeeld, de Adres programmaregel routine roept de ROM-routine aan op adres 6510. Op de ZX81 start de routine op adres 2520. Een dis-assembly van deze routine geeft:

```
push hl
ld hl, program
ld d, h*
ld e, l*
pop bc*
call 09EA
```

De drie door een ster aangegeven bytes zijn dezelfde bij de Spectrum en kunnen worden gevonden via de zoek-routine. In feite geeft de routine de waarde 6514 terug, wat vier hoger is dan het startadres van de ROM-routine in kwestie.

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	ld hl, (23296)	42 0 91
	ld de, (23298)	237 91 2 91
restart	ld bc, (23299)	237 75 3 91
	ld a,e	123
	cp 0	254 0
	ret z	200
	push hl	229
	ld d, 0	22 0
compare	ld a, (bc)	10
	cp (hl)	190
	jr z, match	40 25
	pop hl	225
	inc hl	35
	push de	213
	push hl	229
	ld hl, 16384	33 0 64
	l d, 0	22 0
	and a	167
	sbc hl,de	237 82
	inc hl	35
	pop de	209
	and a	167
	sbc hl,de	237 82
	ex de,hl	235
	pop de	209
	jr nz, restart	32 220
	ld bc, 65535	1 255 255
	ret	201

```
match       inc d              20
            ld a,d             122
            cp e               187
            jr nz, next byte   32  2
            pop bc             193
            ret                201
next byte   inc hl             35
            inc bc             3
            jr compare         24  216
```

Hoe het werkt
Het registerpaar hl wordt geladen met het adres van de eerste locatie in het geheugen dat moet worden bekeken. Om dit de eerste keer in ROM te vinden moet dit op nul worden gezet. Het register e wordt geladen met het aantal bytes van de doorzochte string. Register bc wordt geladen met het adres van de string, door de gebruiker in RAM ingetoetst. De accumulator wordt geladen met de lengte van de string, en als dit nul is keert de routine naar BASIC terug.
Het adres in hl wordt op de stapel opgeslagen. De accumulator wordt geladen met de door het registerpaar bc aangegeven byte. Als deze dezelfde is als de door hl aangegeven byte, wordt er naar 'match' gesprongen. Als deze twee verschillend zijn, wordt hl met het adres uit de stapel geladen. Dit wordt dan verhoogd zodat het naar de volgende locatie in het geheugen wijst.
Registers de en hl worden op de stapel opgeslagen, hl wordt geladen met het adres van de eerste byte in RAM, en de wordt geladen met de lengte van de string. de wordt afgetrokken van hl om het hoogst mogelijke startadres van de string te geven. Dit wordt verhoogd zodat het naar het eerste adres wijst waar de string niet kan zitten.
Het adres van de top van de stapel wordt in de geladen en dit wordt van hl afgetrokken. Het resultaat van deze bewerking wordt onthouden, terwijl hl wordt geladen met de, en de wordt geladen met het getal uit de stapel. Als het resultaat nul was, wordt bc met 65535 geladen en de routine komt in BASIC terug, omdat de string niet in de ROM aanwezig is. Als het resultaat niet nul was, gaat de routine naar 'restart' terug.
Bij 'match' wordt register d verhoogd zodat hij het aantal gevonden bytes bevat dat met de string overeenkomt. Als dit gelijk is aan de lengte van de string, wordt bc van de stapel opgehaald en de routine komt in BASIC terug. Als register d niet de lengte van de string bevatte, worden zowel hl als bc verhoogd zodat ze naar de volgende bytes van de string wijzen en de routine gaat naar 'compare'.

8.11 Instr$
Lengte: 168
Aantal variabelen: 0
Check sum: 19875

Werking
Deze routine geeft de positie van een substring (B$) in een hoofdstring (A$) terug, of nul als er een fout optreedt.

Call
Let P=USR adres

Foutafhandeling
Als, hetzij de string niet bestaat, de lengte van de substring nul is, of de substring is langer dan de hoofdstring, geeft de routine de waarde nul terug.
Als er geen fout optreedt, maar de substring kan niet in de hoofdstring worden gevonden, geeft de routine ook nul terug.

Commentaar
Bij terugkomst van de routine, zal de variabele P (iedere andere variabele kan worden gebruikt) de retourwaarde houden. De betreffende strings kunnen niet geDIMensioneerd zijn als teken-arrays. Om de gebruikte strings te veranderen, moeten de met een ster aangegeven getallen worden veranderd. 66* is de substring, 65* de hoofdstring. Vervang om dit te veranderen, de getallen door de codes van de vereiste tekens (A tot Z = 65 tot 90).

Machinetaal-listing

Label	*Assembly-taal*	*In te toetsen getallen*
	sub a	151
	ld b,a	71
	ld c,a	79
	ld d, a	87
	ld e,a	95
	ld hl, (23627)	42 75 92
next variable	ld a, (hl)	126
	cp 128	254 128
	jr z, not found	40 95
	bit 7,a	203 127
	jr nz, for-next	32 41
	cp 96	254 96
	jr nc, number	48 29
	cp 65	254 65*
	jr nz, substring	32 2
	ld d,h	84
	ld e,l	93
substring	cp 66	254 66*
	jr nz, check	32 2
	ld b,h	68
	ld c,l	77
check	ld a,d	122
	or e	179
	jr z, string	40 4
	ld a, b	120
	or c	177
	jr nz, found	32 38
string	push de	213
	inc hl	35
	ld e, (hl)	94
	inc hl	35
	ld d, (hl)	86
add	add hl,de	25
	pop de	209
	jr increase	24 5

```
number          inc hl                  35
                inc hl                  35
                inc hl                  35
                inc hl                  35
                inc hl                  35
increase        inc hl                  35
                jr next variable        24  206
for-next        cp 224                  254  224
                jr c, next bit          56  6
                push de                 213
                ld de, 18               17  18  0
                jr add                  24  234
next bit        bit 5,a                 203  111
                jr z, string            40  225
next byte       inc hl                  35
                bit 7, (hl)             203  126
                jr z, next byte         40  251
                jr number               24  227
found           ex de,hl                235
                inc hl                  35
                inc hl                  35
                push hl                 229
                push hl                 229
                inc bc                  3
                push bc                 197
                ld a, (bc)              10
                ld e,a                  95
                inc bc                  3
                ld a, (bc)              10
                ld d,a                  87
                or e                    179
                jr z, zero length       40  11
                push de                 213
                ld a, (hl)              126
                dec hl                  43
                ld l, (hl)              110
                ld h,a                  103
                and a                   167
                sbc hl,de               237  82
                jr nc, continue         48  8
                pop bc                  193
zero length     pop bc                  193
                pop bc                  193
error           pop bc                  193
not found       ld bc, 0                1  0  0
                ret                     201
continue        pop ix                  221  225
                pop bc                  193
                ex de,hl                235
                pop hl                  225
                inc bc                  3
                inc bc                  3
```

save	inc hl	35
	push hl	229
	push bc	197
	push ix	221 229
	push de	213
compare	ld a, (bc)	10
	cp (hl)	190
	jr z, match	40 12
	pop de	209
	pop ix	221 225
	pop bc	193
	pop hl	225
	ld a,d	122
	or e	179
	jr z, error	40 225
	dec de	27
	jr save	24 234
match	inc hl	35
	inc bc	3
	push hl	229
	dec ix	221 43
	push ix	221 229
	pop hl	225
	ld a,h	124
	or l	181
	pop hl	225
	jr nz, compare	32 227
	pop de	209
	pop de	209
	and a	167
	sbc hl,de	237 82
	pop de	209
	pop de	209
	pop de	209
	and a	167
	sbc hl,de	237 82
	ld b,h	68
	ld c,l	77
	ret	201

Hoe het werkt

De accumulator, het registerpaar bc en registerpaar de worden alle met nul geladen. bc krijgt krijgt later in de routine het adres van B$ en de het adres van A$. hl wordt geladen met het adres van het variabelengebied.

De accumulator wordt geladen met de door hl geadresseerde byte. Als de accumulator 128 bevat, gaat de routine naar 'not found' omdat het einde van het variabelengebied is bereikt. Als bit 7 van de accumulator op één wordt gezet, wordt er naar 'for-next' gesprongen, omdat de gevonden variabele niet een string of een getal is waarvan de naam één letter lang is. Als de accumulator een getal bevat dat groter is dan 95, wordt er naar 'number' gesprongen.

Om dit stadium te bereiken, moet een string zijn gevonden. Als de accumulator de waarde 65 bevat, is A$ gelocaliseerd, en de inhoud van hl wordt in de gekopieerd. Als de accumulator de waarde 66 bevat, is B$ gevonden, en hl in bc geko-

pieerd. Als registers de en bc niet nul bevatten, zijn beide strings gelocaliseerd en dc routine gaat naar 'found'.
Als de routine 'string' bereikt, wordt de op de stapel opgeslagen en geladen met de lengte van de gevonden string. Dit wordt bij het adres van de hoge byte van de string-pointer opgeteld, en in hl opgeslagen. de wordt van de stapel opgehaald en er wordt naar 'increase' gesprongen.
Bij 'number' wordt hl vijf keer verhoogd zodat het naar de laatste byte wijst van ieder gevonden getal. hl wordt dan verhoogd zodat het naar de volgende variabele wijst, en er wordt naar 'next variabele' gesprongen.
Bij 'for-next' wordt, als de accumulator een getal onder 224 bevat, naar 'next bit' gesprongen omdat de gevonden variabele niet een FOR-NEXT-luscontrolevariabele is. Als de waarde in de accumulator hoger is dan 223, wordt er achttien bij hl opgeteld zodat het naar de laatste controle-byte wijst, en de routine gaat naar 'increase'.
Als de routine 'next bit' bereikt en als bit 5 van de accumulator op nul staat, wordt er naar 'string' gesprongen, om hl met het adres van de volgende variabele te laden, omdat er een array is gevonden.
Als de routine 'next byte' bereikt, is er een getal gevonden waarvan de naam een grotere lengte heeft dan één teken. hl wordt dus verhoogd zodat het naar het laatste teken van de variabelenaam wijst en er wordt naar 'number' gesprongen.
Bij 'found' wordt hl geladen met het adres van A$ en dit wordt twee keer verhoogd om het adres te geven van de hoge byte van de pointers. Deze waarde wordt dan twee keer op de stapel opgeslagen. bc wordt verhoogd zodat het naar de lage byte van de pointers voor B$ wijst. Het adres uit bc wordt dan op de stapel opgeslagen. de wordt geladen met de lengte van B$, en als dit nul is, wordt er naar 'zero length' gesprongen. de wordt dan op de stapel gePUSHed. hl wordt geladen met de lengte van A$ en als dit niet minder is dan de, gaat de routine naar 'continue'. De stapel wordt dan op zijn oorspronkelijke grootte gebracht, bv wordt met nul geladen, en de routine keert in BASIC terug.
Bij 'continue' wordt ix geladen met de lengte van B$, en in bc komt het adres van de lage byte van de pointers voor B$. de wordt geladen met het lengteverschil van A$ en B$, en hl wordt geladen met het adres van de hoge byte van de pointers voor A$. bc wordt dan twee keer verhoogd zodat dit het adres geeft van het eerste teken in B$. hl wordt verhoogd zodat het naar het volgende teken van A$ wijst. hl, bc, ix, en de worden dan op de stapel opgeslagen. De accumulator wordt geladen met de door bc geadresseerde byte, en als deze dezelfde is als de door hl geadresseerde byte, wordt er naar 'match' gesprongen. de, ix, bc en hl worden dan van de stapel opgehaald. Als register de nul bevat wordt er naar 'error' gesprongen omdat B$ niet in A$ voorkomt. De teller wordt dan verlaagd en de routine gaat naar 'save' terug.
Als dc routine 'match' bereikt, worden zowel bc als hl verhoogd zodat ze respectievelijk naar de volgende tekens van A$ en B$ wijzen. hl wordt dan op dc stapel opgeslagen. De teller in ix wordt dan verlaagd en als ix niet nul bevat, wordt er, na hl van de stapel op te halen, naar 'compare' teruggesprongen.
Om dit stadium te bereiken, moet B$ binnen A$ zijn gevonden. De lengte van B$ wordt dan van hl afgetrokken, en het adres van de hoge byte van de pointers voor A$ wordt van hl afgetrokken. Het resultaat is de positie van B$ binnen A$. Dit wordt naar register bc gekopieerd en de routine keert in BASIC terug.

Appendix A

Er zijn twee belangrijke tabellen van instructies in deze appendix. Tabel A2 laat de één-byte-instructies zien en de twee-byte-instructies die voorafgegaan worden door 203 (hexadecimaal CB) of 237 (hexadecimaal ED). Tabel A3 laat de index-register-instructies zien.
Er zijn veel patronen in de instructieset. Bijvoorbeeld, de registers staan bijna altijd in de volgorde b, c, d, e, h, l, (hl), a, zoals, bijvoorbeeld, de groep van 8-bit register-naar-register laadinstructies, met nummers 64 tot 127. Op dezelfde manier volgen de indexregistercodes de hl-codes, voorafgegaan door 221 (hexadecimaal DD) wanneer deze betrekking hebben op het ix-indexregister, en door 253 (hexadecimaal FD) wanneer deze betrekking hebben op het iy-indexregister.
Sommige instructies worden gekenmerkt door één of meer van de volgende tekens:

n —een één-byte geheel getal tussen 0 t/m 255.
d —een één-byte verplaatsingsfactor tussen 0 t/m 255 (indexregister-instructies), of tussen −127 en 128 (spronginstructies). Negatieve waardes van d worden verkregen door het positieve getal van 256 af te trekken.
nn —een twee-byte geheel getal tussen 0 t/m 65535. De meest significante byte zit op de tweede plaats, bijvoorbeeld 16384 (= 0 + 256*64) wordt als 0,64 opgeslagen.

Kenmerken worden altijd bij de byte of bytes geplaatst die de instructie volgen waarop deze betrekking hebben, behalve bij drie-byte indexregister-instructies (kolom 5 en 6 van tabel A3); in dit geval worden ze tussen de tweede en de derde byte geplaatst. Zie tabel A1 voor voorbeelden.

Tabel A1: Enkele voorbeelden van de Z80A-instructieset. Kolom één verwijst naar de betreffende kolom van tabel A2 en A3

Tabel en kolom-nummer	*Algemene vorm*	*Specifiek voorbeeld*	*Decimaal*
A2,3	- -	inc b	4
A2,3	ld e, n	ld c, 25	30 25
A2,3	ld a, (nn)	ld a, (23296)	58 0 91
A2,4	- -	res 2, d	203 146
A2,5	ld (nn), de	ld (23760), de	237 83 208 92
A3,3	- -	add ix, bc	221 9
A3,3	ld (ix+d), n	LD (ix+193),5	221 54 193 5
A3,4	- -	add iy, bc	253 9
A3,4	ld (nn), iy	ld (23760), iy	253 34 208 92
A3,5	rrc (ix+d)	rrc (ix+5)	221 203 5 14
A3,6	rrc (iy+d)	rrc (iy+5)	253 203 5 14

Tabel A2: Z80A-instructies, behalve voor de indexregister-codes (zie tabel A3)

Decimaal	*Hex*	*Opcode*	*Na 203 (hex CB)*	*Na 237 (hex ED)*
0	00	nop	rlc b	
1	01	ld bc,nn	rlc c	
2	02	ld (bc),a	rlc d	
3	03	inc bc	rlc e	
4	04	inc b	rlc h	
5	05	dec b	rlc l	
6	06	ld b,n	rlc (hl)	
7	07	rlca	rlc a	
8	08	ex,af,af'	rrc b	
9	09	add hl,bc	rrc c	
10	0A	ld a, (bc)	rrc d	
11	0B	dec bc	rrc e	
12	0C	inc c	rrc h	
13	0D	dec c	rrc l	
14	0E	ld c,n	rrc (hl)	
15	0F	rrca	rrc a	
16	10	djnz d	rl b	
17	11	ld de,nn	rl c	
18	12	ld (de),a	rl d	
19	13	inc de	rl e	
20	14	inc d	rl h	
21	15	dec d	rl l	
22	16	ld d,n	rl (hl)	
23	17	rla	rl a	
24	18	jr d	rr b	
25	19	add hl,de	rrc	
26	1A	ld a, (de)	rr d	
27	1B	dec de	rr e	
28	1C	inc e	rr h	
29	1D	dec e	rr l	
30	1E	ld e,n	rr (hl)	
31	1F	rra	rr a	
32	20	jr nz,d	sla b	
33	21	ld hl,nn	sla c	
34	22	ld (nn),hl	sla d	
35	23	inc hl	sla e	
36	24	inc h	sla h	
37	25	dec h	sla l	
38	26	ld h,n	sla (hl)	
39	27	daa	sla a	
40	28	jr z,d	sra b	
41	29	add hl,hl	sra c	
42	2A	ld hl, (nn)	sra d	
43	2B	dec hl	sra e	
44	2C	inc l	sra h	
45	2D	dec l	sra l	
46	2E	ld l,n	sra (hl)	
47	2F	cpl	sra a	
48	30	jr nc,d		

Decimaal	*Hex*	*Opcode*	*Na 203 (hex CB)*	*Na 237 (hex ED)*
49	31	ld sp,nn		
50	32	ld (nn),a		
51	33	inc sp		
52	34	inc (hl)		
53	35	dec (hl)		
54	36	ld (hl),n		
55	37	scf		
56	38	jr c,d	srl b	
57	39	add hl,sp	srl c	
58	3A	ld a, (nn)	srl d	
59	3B	dec sp	srl e	
60	3C	inc a	srl h	
61	3D	dec a	srl l	
62	3E	ld a,n	srl (hl)	
63	3F	ccf	srl a	
64	40	ld b,b	bit 0,b	in b, (c)
65	41	ld b,c	bit 0,c	out (c),b
66	42	ld b,d	bit 0,d	sbc hl,bc
67	43	ld b,e	bit 0,e	ld (nn),bc
68	44	ld b,h	bit 0,h	neg
69	45	ld b,l	bit 0,l	retn
70	46	ld b, (hl)	bit 0, (hl)	im 0
71	47	ld b,a	bit 0,a	ld i,a
72	48	ld c,b	bit 1,b	in c, (c)
73	49	ld c,c	bit 1,c	out (c),c
74	4A	ld c,d	bit 1,d	adc hl,bc
75	4B	ld c,e	bit 1,e	ld bc,(nn)
76	4C	ld c,h	bit 1,h	
77	4D	ld c,l	bit 1,l	reti
78	4E	ld c, (hl)	bit 1, (hl)	
79	4F	ld c,a	bit 1,a	ld r,a
80	50	ld d,b	bit 2,b	in d, (c)
81	51	ld d,c	bit 2,c	out (c),d
82	52	ld d,d	bit 2,d	sbc hl,de
83	53	ld d,e	bit 2,e	ld (nn),de
84	54	ld d,h	bit 2,h	
85	55	ld d,l	bit 2,l	
86	56	ld d, (hl)	bit 2, (hl)	im 1
87	57	ld d,a	bit 2,a	ld a,i
88	58	ld e,b	bit 3,b	in e,(c)
89	59	ld e,c	bit 3,c	out (c),e
90	5A	ld e,d	bit 3,d	adc hl,de
91	5B	ld e,e	bit 3,e	ld de,(nn)
92	5C	ld e,h	bit 3,h	
93	5D	ld e,l	bit 3,1	
94	5E	ld e, (hl)	bit 3, (hl)	im 2
95	5F	ld e,a	bit 3,a	ld a,r
96	60	ld h,b	bit 4,b	in h, (c)
97	61	ld h,c	bit 4,c	out (c),h
98	62	ld h,d	bit 4,d	sbc hl,hl
99	63	ld h,e	bit 4,e	ld (nn),hl

Decimaal	*Hex*	*Opcode*	*Na 203 (hex CB)*	*Na 237 (hex ED)*
100	64	ld h,h	bit 4,h	
101	65	ld h,l	bit 4,l	
102	66	ld h, (hl)	bit 4, (hl)	
103	67	ld h,a	bit 4,a	rrd
104	68	ld l,b	bit 5,b	in l,(c)
105	69	ld l,c	bit 5,c	out (c),l
106	6A	ld l,d	bit 5,d	adc hl,hl
107	6B	ld l,e	bit 5,e	ld hl,(nn)
108	6C	ld l,h	bit 5,h	
109	6D	ld l,l	bit 5,l	
110	6E	ld l, (hl)	bit 5, (hl)	
111	6F	ld l,a	bit 5,a	rld
112	70	ld (hl),b	bit 6,b	in f,(c)
113	71	ld (hl),c	bit 6,c	
114	72	ld (hl),d	bit 6,d	sbc hl,sp
115	73	ld (hl),e	bit 6,e	ld (nn),sp
116	74	ld (hl),h	bit 6,h	
117	75	ld (hl),l	bit 6,l	
118	76	halt	bit 6, (hl)	
119	77	ld (hl),a	bit 6,a	
120	78	ld a,b	bit 7,b	in a,(c)
121	79	ld a,c	bit 7,c	out (c),a
122	7A	ld a,d	bit 7,d	adc hl,sp
123	7B	ld a,e	bit 7,e	ld sp, (nn)
124	7C	ld a,h	bit 7,h	
125	7D	ld a,l	bit 7,l	
126	7E	ld a, (hl)	bit 7, (hl)	
127	7F	ld a,a	bit 7,a	
128	80	add a,b	res 0,b	
129	81	add a,c	res 0,c	
130	82	add a,d	res 0,d	
131	83	add a,e	res 0,e	
132	84	add a,h	res 0,h	
133	85	add a,l	res 0,l	
134	86	add a, (hl)	res 0, (hl)	
135	87	add a,a	res 0,a	
136	88	adc a,b	res 1,b	
137	89	adc a,c	res 1,c	
138	8A	adc a,d	res 1,d	
139	8B	adc a,e	res 1,e	
140	8C	adc a,h	res 1,h	
141	8D	adc a,l	res 1,l	
142	8E	adc a, (hl)	res 1, (hl)	
143	8F	adc a,a	res 1,a	
144	90	sub b	res 2,b	
145	91	sub c	res 2,c	
146	92	sub d	res 2,d	
147	93	sub e	res 2,e	
148	94	sub h	res 2,h	
149	95	sub l	res 2,l	
150	96	sub (hl)	res 2, (hl)	

Decimaal	*Hex*	*Opcode*	*Na 203 (hex CB)*	*Na 237 (hex ED)*
151	97	sub a	res 2,a	
152	98	sbc a,b	res 3,b	
153	99	sbc a,c	res 3,c	
154	9A	sbc a,d	res 3,d	
155	9B	sbc a,e	res 3,e	
156	9C	sbc a,h	res 3,h	
157	9D	sbc a,l	res 3,1	
158	9E	sbc a, (hl)	res 3, (hl)	
159	9F	sbc a,a	res 3,a	
160	A0	and b	res 4,b	ldi
161	A1	and c	res 4,c	cpi
162	A2	and d	res 4,d	ini
163	A3	and e	res 4,e	outi
164	A4	and h	res 4,h	
165	A5	and l	res 4,1	
166	A6	and (hl)	res 4, (hl)	
167	A7	and a	res 4,a	
168	A8	xor b	res 5,b	ldd
169	A9	xor c	res 5,c	cpd
170	AA	xor d	res 5,d	ind
171	AB	xor e	res 5,e	outd
172	AC	xor h	res 5,h	
173	AD	xor l	res 5,l	
174	AE	xor (hl)	res 5, (hl)	
175	AF	xor a	res 5,a	
176	B0	or b	res 6,b	ldir
177	B1	or c	res 6,c	cpir
178	B2	or d	res 6,d	inir
179	B3	or e	res 6,e	otir
180	B4	or h	res 6,h	
181	B5	or l	res 6,1	
182	B6	or (hl)	res 6, (hl)	
183	B7	or a	res 6,a	
184	B8	cp b	res 7,b	lddr
185	B9	cp c	res 7,c	cpdr
186	BA	cp d	res 7,d	indr
187	BB	cp e	res 7,e	otdr
188	BC	cp h	res 7,h	
189	BD	cp l	res 7,l	
190	BE	cp (hl)	res 7, (hl)	
191	BF	cp a	res 7,a	
192	C0	ret nz	set 0,b	
193	C1	pop bc	set 0,c	
194	C2	jp nz, nn	set 0,d	
195	C3	jp nn	set 0,e	
196	C4	call nz,nn	set 0,h	
197	C5	push bc	set 0,1	
198	C6	add a,n	set 0, (hl)	
199	C7	rst 0	set 0,a	
200	C8	ret z	set 1,b	
201	C9	ret	set 1,c	

Decimaal	*Hex*	*Opcode*	*Na 203 (hex CB)*	*Na 237 (hex ED)*
202	CA	jp z,nn	set 1,d	
203	CB	—	set 1,e	
204	CC	call z,nn	set 1,h	
205	CD	call nn	set 1,l	
206	CE	adc a,n	set 1, (hl)	
207	CF	rst 8	set 1,a	
208	D0	ret nc	set 2,b	
209	D1	pop de	set 2,c	
210	D2	jp nc,nn	set 2,d	
211	D3	out (n),a	set 2,e	
212	D4	call nc,nn	set 2,h	
213	D5	push de	set 2,l	
214	D6	sub,n	set 2, (hl)	
215	D7	rst 16	set 2,a	
216	D8	ret c	set 3,b	
217	D9	exx	set 3,c	
218	DA	jp c,nn	set 3,d	
219	DB	in a,(n)	set 3,e	
220	DC	call c,nn	set 3,h	
221	DD	—	set 3,l	
222	DE	sbc a,n	set 3, (hl)	
223	DF	rst 24	set 3,a	
224	E0	ret po	set 4,b	
225	E1	pop hl	set 4,c	
226	E2	jp po,nn	set 4,d	
227	E3	ex (sp),hl	set 4,e	
228	E4	call po,nn	set 4,h	
229	E5	push hl	set 4,l	
230	E6	and n	set 4, (hl)	
231	E7	rst 32	set 4,a	
232	E8	ret pe	set 5,b	
233	E9	jp (hl)	set 5,c	
234	EA	jp pe,nn	set 5,d	
235	EB	ex de,hl	set 5,e	
236	EC	call pe,nn	set 5,h	
237	ED	--	set 5,l	
238	EE	xor n	set 5, (hl)	
239	EF	rst 40	set 5,a	
240	F0	ret p	set 6,b	
241	F1	pop af	set 6,c	
242	F2	jp p,nn	set 6,d	
243	F3	di	set 6,e	
244	F4	call p,nn	set 6,h	
245	F5	push af	set 6,l	
246	F6	or n	set 6, (hl)	
247	F7	rst 48	set 6,a	
248	F8	ret m	set 7,b	
249	F9	ld sp,hl	set 7,c	
250	FA	jp m,nn	set 7,d	
251	FB	ei	set 7,e	
252	FC	call m,nn	set 7,h	

Decimaal	*Hex*	*Opcode*	*Na 203 (hex CB)*	*Na 237 (hex ED)*
253	FD	—	set 7,l	
254	FE	cp n	set 7, (hl)	
255	FF	rst 56	set 7,a	

Tabel A3: Indexregister-codes. Kolom 3 en 5 hebben betrekking op het ix-register. Kolom 4 en 6 hebben betrekking op het iy-register. Alle instructies uit deze tabel volgen de instructies voor het hl-registerpaar uit tabel A2.

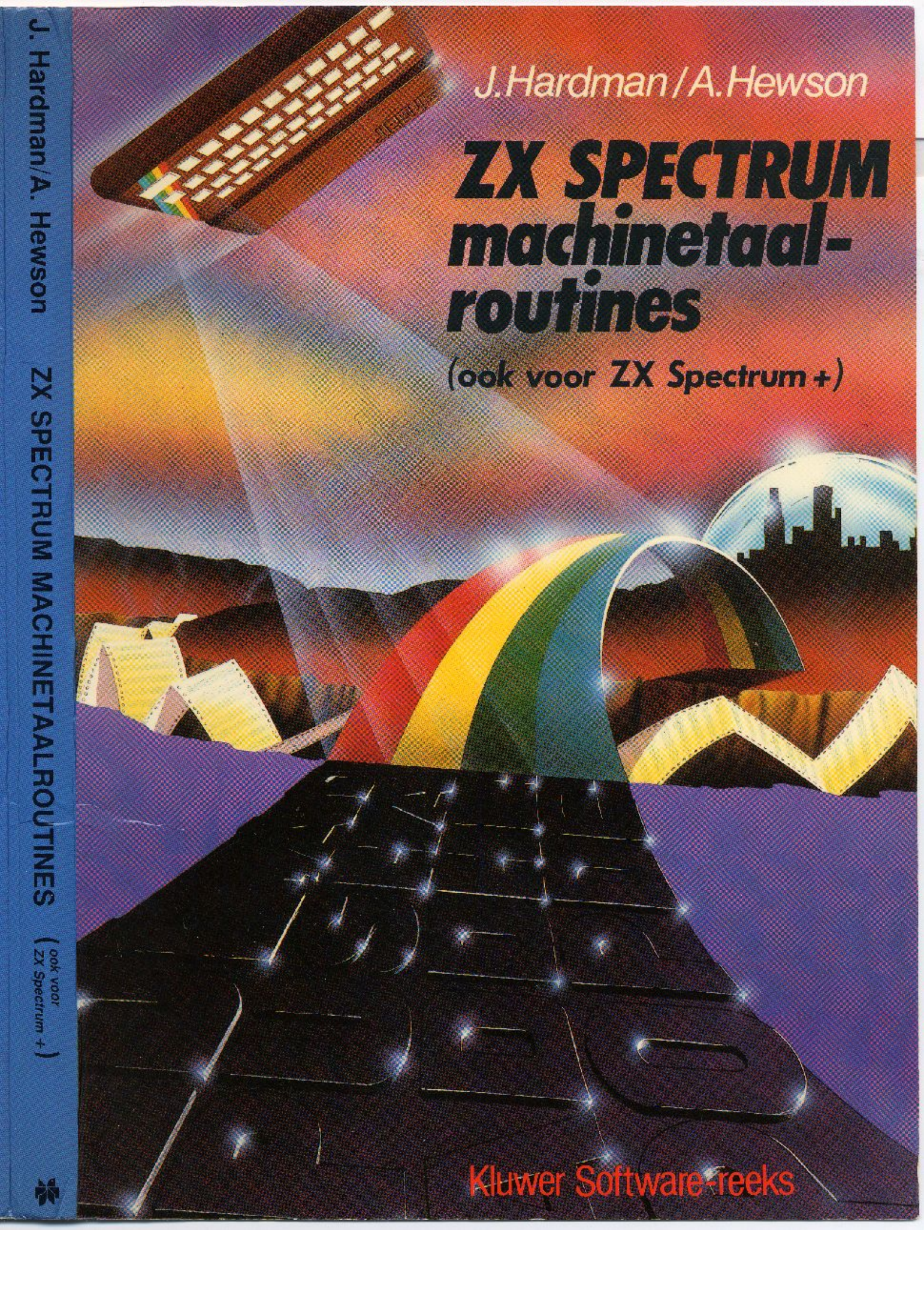
Decimaal	*Hex*	*Na 221 (hex DD)*	*Na 253 (hex FD)*	*Na 221, 203 (hex DD, CB)*	*Na 253, 203 (hex FD, CB)*
6	06			rlc (ix + d)	rlc (iy + d)
9	09	add ix, bc	add iy,bc		
14	0E			rrc (ix + d)	rrc (iy + d)
22	16			rl (ix + d)	rl (iy + d)
25	19	add ix,de	add iy,de		
30	1E			rr (ix + d)	rr (iy + d)
33	21	ld ix,nn	ld iy,nn		
34	22	ld (nn),ix	ld (nn),iy		
35	23	inc ix	inc iy		
38	26			sla (ix + d)	sla (iy + d)
41	29	add ix,ix	add iy,iy		
42	2A	ld ix,(nn)	ld iy,(nn)		
43	2B	dec ix	dec iy		
46	2E			sra (ix + d)	sra (iy + d)
52	34	inc (ix + d)	inc (iy + d)		
53	35	dec (ix + d)	dec (iy + d)		
54	36	ld (ix + d),n	ld (iy + d),n		
57	39	add ix,sp	add iy,sp		
62	3E			srl (ix + d)	srl (iy + d)
70	46	ld b, (ix + d)	ld b, (iy + d)	bit 0,(ix + d)	bit 0,(iy + d)
78	4E	ld c,(ix + d)	ld c,(iy + d)	bit 1,(ix + d)	bit 1,(iy + d)
86	56	ld d,(ix + d)	ld d,(iy + d)	bit 2,(ix + d)	bit 2,(iy + d)
94	5E	ld e,(ix + d)	ld e,(iy + d)	bit 3,(ix + d)	bit 3,(iy + d)
102	66	ld h,(ix + d)	ld h,(iy + d)	bit 4,(ix + d)	bit 4,(iy + d)
110	6E	ld l,(ix + d)	ld l,(iy + d)	bit 5,(ix + d)	bit 5,(iy + d)
112	70	ld (ix + d),b	ld (iy + d),b		
113	71	ld (ix + d),c	ld (iy + d),c		
114	72	ld (ix + d),d	ld (iy + d),d		
115	73	ld (ix + d),e	ld (iy + d),e		
116	74	ld (ix + d),h	ld (iy + d),h		
117	75	ld (ix + d),l	ld (iy + d),l		
118	76			bit 6, (ix + d)	bit 6, (iy + d)
119	77	ld (ix + d),a	ld (iy + d),a		
126	7E	ld a,(ix + d)	ld a,(iy + d)	bit 7, (ix + d)	bit 7,(iy + d)
134	86	add a,(ix + d)	add a,(iy + d)	res 0,(ix + d)	res 0,(iy + d)
142	8E	adc a,(ix + d)	adc a,(iy + d)	res 1,(ix + d)	res 1,(iy + d)
150	96	sub (ix + d)	sub (iy + d)	res 2,(ix + d)	res 2,(iy + d)
158	9E	sbc a,(ix + d)	sbc a,(iy + d)	res 3,(ix + d)	res 3,(iy + d)

Decimaal	*Hex*	*Na 221 (hex DD)*	*Na 253 (hex FD)*	*Na 221, 203 (hex DD, CB)*	*Na 253, 203 (hex FD, CB)*
166	A6	and (ix + d)	and (iy + d)	res 4,(ix + d)	res 4,(iy + d)
174	AE	xor (ix + d)	xor (iy + d)	res 5,(ix + d)	res 5,(iy + d)
182	B6	or (ix + d)	or (iy + d)	res 6,(ix + d)	res 6,(iy + d)
190	BE	cp (ix + d)	cp (iy + d)	res 7, (ix + d)	res 7, (iy + d)
198	C6			set 0,(ix + d)	set 0,(iy + d)
206	CE			set 1,(ix + d)	set 1,(iy + d)
214	D6			set 2,(ix + d)	set 2,(iy + d)
222	DE			set 3,(ix + d)	set 3,(iy + d)
225	E1	pop ix	pop iy		
227	E3	ex (sp),ix	ex (sp),iy		
229	E5	push ix	push iy		
230	E6			set 4,(ix + d)	set 4,(iy + d)
233	E9	jp ix	jp iy		
238	EE			set 5,(ix + d)	set 5,(iy + d)
246	F6			set 6,(ix + d)	set 6,(iy + d)
249	F9	ld sp,ix	ld sp,iy		
254	FE			set 7,(ix + d)	set 7,(iy + d)

Als gevolg van het succes van 'Machinetaal voor de ZX Spectrum' van W. Tang, hebben wij dit boek in ons fonds opgenomen.
De opzet van dit boek is zowel de beginnende als de meer ervaren computergebruiker te voorzien van een verzameling nuttige en interessante machinetaalroutines voor de ZX Spectrum. Uiteraard zijn deze routines ook geschikt voor het nieuwe model, de ZX Spectrum +.
Het eerste deel beschrijft de kenmerken van de Spectrum die van belang zijn voor de programmeur in machinetaal, wat wordt er bedoeld met machinetaal, de belangrijke interne eigenschappen en routines van de Spectrum en de structuur van de machinetaal.
Het tweede gedeelte van het boek bevat de machinetaalroutines. Hierbij wordt een uitgebreide toelichting gegeven op hun opbouw en werking. De routines zijn zodanig opgezet dat ze elk een afgerond onderdeel vormen.
Het is niet noodzakelijk om de werking van alke routine geheel te doorzien, omdat ze eenvoudig te laden zijn met behulp van het M/C-lader-programma dat aan het begin van het tweede deel wordt gegeven.

ISBN 90 201 1797 1

J. Hardman/A. Hewson ZX SPECTRUM MACHINETAALROUTINES (ook voor ZX Spectrum +)


J. Hardman / A. Hewson
ZX SPECTRUM machinetaal-routines
(ook voor ZX Spectrum +)
Kluwer Software-reeks
J. Hardman/A. Hewson
ZX SPECTRUM MACHINETAALROUTINES
(ook voor ZX Spectrum +)