

J. Vanderaart

John Vanderaart

# MACHINETAAL VOOR

## MSX COMPUTERS



Kluwer

MACHINETAAL VOOR MSX-COMPUTERS





# **Machinetaal voor MSX-computers**



John Vanderaart

# Machinetaal voor MSX-computers



Kluwer Technische Boeken B.V. Deventer – Antwerpen

Omslag: W. Niessink

ISBN 90 201 1999 0  
D/1987/0108/191  
NUGI 434

© 1987 Kluwer Technische Boeken B.V. Deventer

1e druk 1987

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

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

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

# Woord vooraf

Rechtvaardigheid! Eerlijkheidshalve moet ik zeggen dat dit boek een behoorlijke 'selfkick' is geworden. Wat mij persoonlijk aan het werken met MSX-computers stoorde, is het gebrek aan 'ter zake doende' informatie en het ontbreken van goed werkende ontwikkelingsprogrammatuur. Zelf programmeer ik **ALLEEN MAAR** in machinetaal en dat op onder andere MSX, Commodore 64/128/C16/+4, Atari ST en AMIGA. Voor de niet-MSX-machines is het allemaal wel in orde: voldoende informatie en goede programmeermogelijkheden. Met de MSX is het tot mijn spijt wat minder mooi gesteld!

De informatie ontbreekt veelal of blinkt uit in nutteloze zwaarlijvigheid en wat de ontwikkelingsprogrammatuur betreft, is het 'huilen met de pet af! Over die informatie volsta ik met op te merken dat deze vaak uit de fabrieksopgave(n) is overgenomen. Daarom is het soms onmogelijk om snel iets terug te vinden. De ontwikkelingsprogrammatuur draait meestal onder MSX-DOS waardoor het weer moeilijk wordt om de BIOS-routines aan te spreken. Ook in alle andere gevallen (misschien een kleine uitzondering voor ZEN) wordt er veel te slordig omgesprongen met het toch al veel te kleine werkgeheugen van de MSX.

Ik heb bovenal getracht een 'anders dan anders' boek te schrijven. De informatie is snel en sluit aan op het eigenlijke programmeren in machinetaal. Geen uitgebreide omschrijvingen dus, maar handige tabellen.

Als programmeeromgeving heb ik speciaal voor dit boek een Z80-monitor geschreven. Het voordeel van zo'n machinetaalmonitor is de directe aanpak! De code wordt ter plekke aangemaakt waardoor het geheugen uiterst efficiënt kan worden gebruikt. Daarnaast vindt U nog een 25-tal extra mogelijkheden als disassemblering, geheugen-dumps, tekst invoer, stelselconversie, logische bewerkingen enz.

Verder krijgt U een aantal treffende MSX-routines die U natuurlijk in uw eigen programma's kunt toepassen. Daarnaast nog een behoorlijk aanzetje tot een karakter-editor en een sprite-editor.

Kortom, efficiency voor alles. De dosis informatie is vrij stevig, maar de (uw) resultaten zijn dan ook navenant. Dat garandeer ik U!

John 'Drj' Vanderaart

# Inhoud

<b>Inleiding . . . . .</b>	11
<b>1 Z80 Machinetaal . . . . .</b>	13
1.1 Werken met een microprocessor . . . . .	13
1.2 Het invoeren van machinecode . . . . .	14
<b>2 De Z80-microprocessor . . . . .</b>	17
2.1 Inleiding . . . . .	17
2.2 De program counter of programmateller . . . . .	18
2.3 De stack pointer of stapelwijzer . . . . .	19
2.4 De indexregisters: IX en IY . . . . .	20
2.5 Het A-register of de accumulator . . . . .	20
2.6 De zes algemene registers: B, C, D, E, H en L . . . . .	20
2.7 Het flag-register F . . . . .	21
2.8 Het interrupt-register I . . . . .	22
2.9 Het refresh-register R . . . . .	22
2.10 De set 'verborgen' registers A' tot en met L' . . . . .	22
<b>3 Adresseringstechnieken . . . . .</b>	23
3.1 Implicite adressering . . . . .	24
3.2 Onmiddellijke adressering . . . . .	24
3.3 Absolute adressering . . . . .	24
3.4 Zero page-adressering . . . . .	25
3.5 Relatieve adressering . . . . .	25
3.6 Geïndexeerde adressering . . . . .	25
3.7 Indirecte adressering . . . . .	26
3.8 Bitadressering . . . . .	27
<b>4 De complete instructieset van de Z80 . . . . .</b>	28
4.1 Even vooraf . . . . .	28
4.2 De instructies op een rij . . . . .	29

<b>5</b>	<b>Getallen en hun representaties . . . . .</b>	56
5.1	Binair rekenen . . . . .	56
5.2	Hexadecimaal rekenen . . . . .	58
5.3	'Rekenen' in ASCII . . . . .	60
<b>6</b>	<b>Enkele basistechnieken . . . . .</b>	63
6.1	Relatieve lussen . . . . .	63
6.2	Logische bewerkingen . . . . .	64
6.3	Dat gedoe met die carry . . . . .	66
6.4	Rekenen in de praktijk . . . . .	67
<b>7</b>	<b>De MSX en de buitenwereld . . . . .</b>	69
7.1	De routines en hun doel . . . . .	69
7.2	Nog even vooraf . . . . .	69
<b>8</b>	<b>Tekstje printen . . . . .</b>	70
<b>9</b>	<b>Invoer via het toetsenbord . . . . .</b>	73
<b>10</b>	<b>De typemachinesimulatie . . . . .</b>	79
<b>11</b>	<b>Spelletje spelen? . . . . .</b>	82
<b>12</b>	<b>Bitbeschouwingen . . . . .</b>	86
<b>13</b>	<b>Hexadecimale sprongen . . . . .</b>	91
<b>14</b>	<b>Ongewoon gewoon! . . . . .</b>	96
<b>15</b>	<b>Tot de volgende keer . . . . .</b>	102
<b>16</b>	<b>MSX à la carte... . . . . .</b>	115
16.1	Achter de schermen van een MSX-computer . . . . .	115
16.2	De videochip . . . . .	116
16.3	Wat gaan we doen? . . . . .	116

<b>17</b>	<b>Tekst 40 × 24</b>	117
17.1	Scrollen	118
17.2	De routines op een rij	122
<b>18</b>	<b>Tekst 32 × 24</b>	124
18.1	Karakters ontwerpen	125
18.2	De routines op een rij	133
<b>19</b>	<b>High resolution</b>	136
19.1	Spelletje spelen?	136
19.2	De routines op een rij	145
<b>20</b>	<b>Multi Color</b>	147
20.1	Over lichtorgels	148
20.2	De routines op een rij	157
<b>21</b>	<b>Sprites dus</b>	159
21.1	Sprites ontwerpen	160
21.2	De routines op een rij	173
<b>22</b>	<b>De SPEEDTYPER</b>	176
22.1	De listing	176
22.2	Werken met de SPEEDTYPER	179
<b>23</b>	<b>De HERBYMON</b>	182
23.1	De programmalisting	182
23.2	Werken met de HERBYMON	192
<b>Appendices</b>		203
A	De Z80-instructieset	203
B	Karaktercodes	223
C	De memory-map	228
D	Kleurentabel	229
E	Video-RAM-tabel	230
F	VDP-registers	231
G	PSG-registers	232
H	IO-adressen	233

I	De BIOS-sprongtabel . . . . .	234
J	Het werkgebied . . . . .	236
K	Hooks . . . . .	243
<b>INDEX</b>	. . . . .	<b>246</b>

# Inleiding

Welkom nogmaals in dit ‘doe’-boek, waarde MSX-bezitter!

U heeft dit boek waarschijnlijk ter hand genomen omdat U benieuwd bent naar al die verborgen MSX-capaciteiten. Een MSX-computer kan echt verschrikkelijk veel, maar niet in BASIC. Tenminste niet SNEL. BASIC, en dan met name het MSX-BASIC, is erg flexibel maar daarmee verliest het meteen een fors stuk snelheid. Toegegeven, in BASIC is ook alles mogelijk, maar dan wél op een laag pitje... Probeert U maar eens 32 sprites te laten bewegen, teksten af te drukken, geluid te produceren, een proces te controleren, en dat alles het liefst ook nog eens tegelijk! De enige oplossing is machinetaal.

Machinetaal. Het klinkt vrij simpel, maar pas op! En zeker op een MSX-computer. Het eerste probleem wordt de ‘syntax’, want machinetaal is op het eerste gezicht niet te begrijpen. Dat klopt ook wel, aangezien de instructies in het begin wel erg op elkaar lijken. Het is een kwestie van gewenning en ervaring, dus zult U even door een zure appel heen moeten!

Het tweede probleem is de programmadefinitie. In een hogere programmeertaal wordt overal voor gezorgd. Iets afdrukken stelt helemaal niets voor, laat staan de rekenkundige bewerkingen of de foutafhandeling. Het bijhouden van het variabellengeheugen ziet U bijvoorbeeld al helemaal niet! In machinetaal daarentegen moet U alles met de hand doen. Vaak wordt dat heel omslachtig, maar het voordeel is een stukje razendsnelle precisie.

Een derde probleem is de kennis over de computer. Laten we uit uw aanschaf van dit boek concluderen dat U BASIC bent ontgroeid. De video-chip kent U natuurlijk al en ook de geluids-chip is inmiddels een open boek. Het is dan ook belachelijk om al die kennis nog eens voor te kauwen. In plaats daarvan vindt u een aantal tabellen met alle informatie die een machinetaal-programmeur zich maar kan wensen.

Vierde en laatste probleem is het invoeren van de gewenste machinecode. Geen probleem aangezien U achterin dit boek een speciaal ontwikkelde machinetaalmonitor zult vinden. Het is even een aardig intypklusje, maar daartegenover staat een stuk ontwikkelings-software dat de prijs van dit boek meer dan rechtvaardigt!

Bijkomend voordeel: U kunt alle listings uit dit boek meteen testen op hun deugdelijkheid...

Wat gaan we doen? We gaan aan het werk! Zoals gezegd is dit een DOE-boek en het is niet de bedoeling dat U zich er met een 'eitje' vanaf kunt maken. Het wordt zweten, maar daarna weet U dan ook precies hoe U volledig zelfstandig draaiende machinetaal-programma's kunt schrijven. Er wordt niets teveel beloofd, want U gaat zien hoe U(!) de video-chip nu ECHT kunt manipuleren, U zult leren hoe bestanden via de cassetterecorder worden geladen en opgeslagen. Ook gaat U werken met de printer, niet te vergeten de sprite-routines, wat spelmogelijkheden, tevens nog wat speciale effecten... Precies datgene wat nodig is voor een aantrekkelijk en supersnel programma!

## De frustratierempel

Voordat we gaan beginnen, moeten we iets afspreken: U klimt niet in de lamp! Wat hiermee wordt bedoeld?

Programmeren in machinetaal is iets dat een grote dosis zelfbeheersing vereist. De kleinste beredeneringsfout en de MSX slaat op 'TILT', met de hoofdletter T dus. Waar U in BASIC nog altijd een CTRL/STOP-combinatie heeft, kunt U het in machinetaal wel vergeten. In 99 van de 100 gevallen is de RESET-knop de enige uitweg en kunt U weer opnieuw beginnen. Maar ja, dan had U tenslotte maar niet aan machinetaal moeten beginnen!

Tegenover de voortdurende frustratie staat een zinderende kick als uw routines WEL werken. U (en ook uw omgeving!) zult werkelijk onder de indruk raken van de fabelachtige snelheid waarmee machinetaal de zaken regelt. Waar U in BASIC de zoveelste bak koffie wegslurpt, komt U om van de machinetaaldorst...

En in geen geval gaat de MSX het raam uit, want dat hebben we daarnet al afgesproken!

# 1 Z80 Machinetaal

## 1.1 Werken met een microprocessor

In iedere computer bevindt zich een microprocessor. Een microprocessor is in feite het 'kloppende hart' van de computer. Via deze processor worden alle zichtbare en onzichtbare computerfuncties aangestuurd, zoals het toetsenbord, het beeldscherm, het interne geheugen enz. In uw MSX-computer zit een Z80, een zeer krachtige en werkelijk razendsnelle 8-bits microprocessor.

Zo'n Z80-processor dient op een zeer speciale manier te worden aangesproken, in een speciaal taaltje: het bijna onbegrijpelijke 'machinetaal'. Wat is machinetaal nu precies?

Machinetaal is een op de microprocessor gerichte en daarom bovenal logisch opgebouwde, zeer directe programmeertaal. De niet al te veeleisende gebruiker merkt al gauw dat machinetaal in feite maar tot vier dingen in staat is:

- 1 simpele berekeningen (optellen, aftrekken, vergelijken)
- 2 dataverplaatsing (in elke mogelijke vorm)
- 3 logische bewerkingen of operaties (bijvoorbeeld 'AND', 'OR' en 'XOR')
- 4 tests en sprongen

Ondanks (misschien dankzij?) al deze schijnbare beperkingen is machinecode snel, zeer snel. De BASIC-interpreter bijvoorbeeld is in feite niets anders dan een programma dat geheel in machinetaal is geschreven. Deze BASIC-interpreter is tevens een voorbeeld van een programma dat in ROM (Read Only Memory – permanent beschreven geheugen, waaruit alleen kan worden gelezen) staat. Het voordeel van zo'n ROM-programma is de permanente aanwezigheid. Een nadeel is de onmogelijkheid om er veranderingen in aan te brengen.

Naast ROM-geheugen bestaat er ook RAM-geheugen (Random Access Memory – geheugen waar zowel uit gelezen als in geschreven kan worden) dat als enige nadeel heeft dat het wordt gewist zodra de computer wordt uitgeschakeld. Programma's die u schrijft en probeert, worden in eerste instantie in RAM-geheugen geschreven. Pas in een later stadium kunt u besluiten deze programma's in een EPROM te 'bakken'. Een EPROM is een iets minder permanent broertje van een

echte ROM-chip; de inhoud blijft bewaard als de stroom wordt uitgeschakeld, maar is door middel van ultraviolet licht te wissen.

Om even op die fabelachtige snelheid terug te komen: machinetaal (mits goed geschreven) kan vele honderden malen sneller zijn dan welk BASIC-programma dan ook. Waar het BASIC altijd eerst dient te worden vertaald, is machinetaal meteen al verwerkingsklaar.

Een nadeel van machinetaal is de geringere flexibiliteit en de noodzakelijke scherpere definitie. Een machinetaalprogramma is namelijk veel minder flexibel dan een programma dat is geschreven in een hogere programmeertaal (BASIC, PASCAL, C...) omdat het veel directer moet zijn geschreven. Denk bijvoorbeeld maar eens aan een simpele uitvoerroutine. In BASIC volstaat een PRINT-opdracht, die ook nog eens eenvoudig is te wijzigen. In machinetaal moet u al gauw een tiental instructies aanspreken en dan maakt u daarbij ook nog eens gebruik van systeem-ROM-routines. Om over het wijzigen nog maar niet te spreken!

Ook moet de definitie van een machinetaalprogramma vrij volledig zijn. In negen van de tien gevallen blijkt een zelfgeschreven routine niet te werken omdat u ergens iets vergeten bent! Een gedegen systeemkennis is dan ook een eerste vereiste. Sterker nog: u zult moeten weten hoe een MSX-computer in elkaar steekt, voordat het verantwoord is om ook maar een byte in machinecode te testen!

Gedegen systeemkennis dus. Waar hogere programmeertalen meestal vrij overdraagbaar zijn (een '10 PRINT "Hallo!" / 20 GOTO 10'-grapje in BASIC werkt immers altijd) is machinetaal uiterst persoonlijk.

Zelfs al beschikken verschillende computers over dezelfde microprocessor, dan nog heeft u meestal met een totaal ander systeem te maken. Andere systeemroutines, andere grafische mogelijkheden, andere randapparatuur enz... De machine-taalprogrammeur dient hier alles over te weten voordat hij ook één zinvolle byte programmeert!

## 1.2 Het invoeren van machinecode

Bij het invoeren van machinetaalprogramma's is de programmeeromgeving een probleem op zich. In BASIC kunt u alles klakkeloos intypen; het gaat goed totdat de computer met een 'OUT OF MEMORY ERROR' of iets van dien aard komt aanzetten. In machinetaal weet u eigenlijk nooit precies wanneer het is afgelopen met de vrije ruimte, want dat bepaalt u zelf! Toegegeven dat er een onder- en

bovengrens aan de vrije geheugenruimte zit, maar dan nog blijven er altijd slimme zijwegen over.

Maar hoe komt de zelfgeschreven machinecode dan in het geheugen terecht? U hoeft niet te proberen om in BASIC een machinecode-instructie te gebruiken. Hoe exact overgenomen ook, u komt altijd met een 'SYNTAX ERROR' te zitten. Slim als u bent, komt u op het idee om de machinecode met behulp van DATA-regels te 'POKEN'. Een programma bestaat tenslotte uit allemaal getallen tussen 0 en 255, zodat de juiste getallen in de juiste volgorde een werkend geheel moeten opleveren. U heeft volkomen gelijk, alleen is het een werkje waarmee zelfs de geduldigste monnik zwaar 'in de stress' raakt!

Om machinecode te verwerken, bestaan er dan ook speciale hulpprogramma's. Twee soorten eigenlijk: assemblers en monitors.

Een assembler is een veredelde tekst-editor waarmee u de machinecode ongeveer zoals teksten in een tekstverwerker kunt invoeren. Erg handig is het feit dat een assembler niet met exacte cijfers werkt maar met zogenaamde labels. Hierdoor kunt u het programma erg eenvoudig op een andere geheugenlocatie laten beginnen. Het nadeel van een assembler is de grootte en de 'lompheid'. Want echt snel en direct kunt u er meestal niet mee werken.

Het snellere broertje van een assembler luistert naar de dubieuze naam 'monitor'. Geen beeldbuis, maar wel iets om mee te 'kijken'. Met een monitor kunt u direct in het geheugen van uw computer gluren. U kunt het geheugen als een tekst-dump aanschouwen, misschien in hexadecimale getallen, maar ook als een disassembly. Een disassembly-listing is precies het tegenovergestelde van een assembleringsproces. Waar 'het laatste uw teksten omzet naar zinvolle getallen en dus te verwerken machinecode, is een disassembly een omzetting van schier onbegrijpelijke getallen naar redelijk leesbare machinecode.

Naast dit is het vanzelfsprekend ook mogelijk om met een monitor direct code te assembleren. Dit laatste gaat regelsgewijs maar het grote voordeel is wel dat u precies ziet WAT en WAAR u schrijft. Verder voorziet een monitor meestal in vele handige foefjes om het u allemaal zo gemakkelijk mogelijk te maken.

In dit boek staat de listing van een echte machinetaalmonitor, de HERBYMON. Een speciaal geschreven produkt dat het mogelijk maakt alle voorbeelden die in dit boek staan meteen te proberen.

Zoals gezegd een monitor... Een kwestie van even een paar slapeloze nachten in verband met het intypen, maar dan gaat er meteen een nieuwe wereld voor u open!

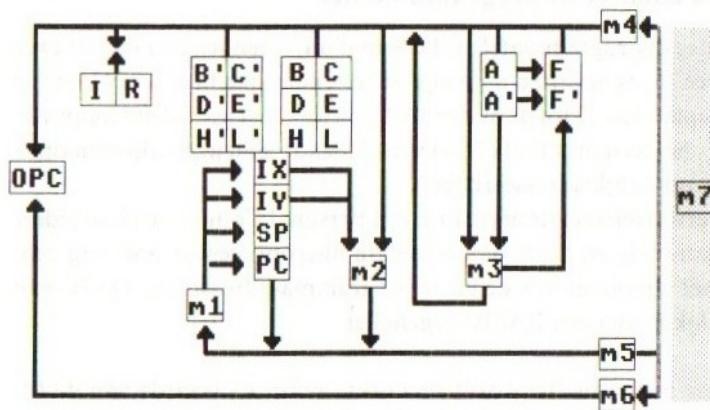
Het verwerken van de machinecode hoeft in het kader van dit boek dan ook helemaal geen problemen meer op te leveren. En natuurlijk kunt u de ingetypte monitor ook gebruiken om zelf programma's te schrijven.

# 2 De Z80-microprocessor

## 2.1 Inleiding

Zoals u inmiddels al weet, zit er in uw MSX een Z80-microprocessor. Allemaal leuk en aardig, maar veel verder helpt ons dat nog niet!

In diverse andere boeken over machinetaal krijgt u eerst een hele verhandeling over het werken met de diverse getallenstelsels, maar waarom zouden we niet eerst kijken wat zo'n Z80 allemaal in huis heeft en wat het ding allemaal kan! In hoofdstuk 5 kunnen we dan altijd nog met cijfertjes stoeien, of u kunt er terecht als iets in dit hoofdstuk u (nog) niet helemaal duidelijk is.



- m1: INCrement / DECrement
- m2: Indexcalculatie
- m3: Logische rekenenheid
- m4: Koppeling databus (8 bits)
- m5: Koppeling adresbus (16 bits)
- m6: Koppeling controlebus
- m7: Buiten de Z80-microprocessor (geheugen en IO)

**OPC:** Instructieregister / instructiedecoder / controle

Afb. 2.1. Schematisch overzicht van de Z80

Een Z80-microprocessor beschikt over een aantal zaken:

- de program counter of programmateller (PC)
- de stack pointer of stapelwijzer (SP)
- de indexregisters: IX en IY
- het A-register of de accumulator
- zes algemene registers: B, C, D, E, H en L (Ook als drie duo's aan te roepen: BC, DE en HL)
- het flag-register F (als registerpaar in AF)
- het interrupt-register I
- het refresh-register R
- een set 'verborgen' registers A' tot en met L'

De laatste drie zijn nogal gevaarlijk en worden in dit boek niet gebruikt, temeer daar ze vooral zijn bedoeld voor specifiek intern gebruik. Pas als u over zeer veel machinetaalervaring beschikt, is het zinnig om ermee te stoeien.

## 2.2 De program counter of programmateller

De program counter of programmateller, PC vanaf nu, is een register dat uit twee bytes bestaat. Twee bytes bevatten namelijk 16 afzonderlijke bits, waardoor het bereik meteen is bepaald tot '2 tot de macht 16-1', ofte wel het gebied dat loopt van \$0000 tot \$FFFF, hetgeen in totaal 65536 verschillende geheugenadressen oplevert. (Allemaal afzonderlijk adresseerbaar!)

Zo'n PC (niet te verwarren met de afkorting van Personal Computer!) is in iedere microprocessor aanwezig en heeft binnen iedere microprocessor ook nog eens dezelfde functie: het bijhouden van de juiste programma-adressering. De PC valt enigszins te vergelijken met een BASIC-regelteller.

Afhankelijk van de instructie die wordt verwerkt, wordt de waarde van de PC beïnvloed. Een 'gewone' instructie verhoogt de PC steeds met één, net genoeg om de volgende instructie te lezen. Andere instructies verhogen de PC soms wel met twee, drie of vier, afhankelijk van de hoeveelheid bytes die zij nodig hebben om aan de slag te kunnen. Ook zijn er speciale instructies, spronginstructies bijvoorbeeld, die de PC van een volstrekt nieuwe waarde voorzien.

Hoe 'werkt' een programma dan? Een voor de hand liggende vraag. U moet zich voorstellen dat er een koppeling bestaat tussen aan de ene kant de Z80-microprocessor en aan de andere kant het (meestal 64K) MSX-geheugen. De inhoud

van de PC wordt op een speciale 16-bits brede adreslijn gezet en naar het MSX-geheugen gestuurd, vervolgens wordt er een byte vanaf het zo geadresseerde PC-adres via een even speciale 8-bits brede datalijn naar de Z80-microprocessor gestuurd...Enzovoort! Afhankelijk van de opgestuurde bytes verzorgt de Z80-processor de te volgen strategie en uw programma werkt – mits goed geprogrammeerd.

## 2.3 De stack pointer of stapelwijzer

De stack pointer, stapelwijzer of (vanaf nu) SP doet vermoeden dat er ook wel een 'stapel' zal zijn! Vanzelfsprekend is dit het geval.

Een stapel is een heel speciaal onderdeel van een microprocessor. Via deze stapel kan een aantal gegevens worden overgedragen. Het kan om gewone data gaan, maar ook adressen worden via de stapel doorgegeven.

Het principe is uitermate simpel. Wat u als laatste op de stapel plaatst, komt er als eerste af. Denk maar eens aan de vuile vaat: het bovenste bord moet u eigenlijk als eerste afwassen. Zo niet dan komt alles naar beneden!

Net als bij dit vergezochte voorbeeld kan er ook op stapelniveau van alles fout gaan. Als de stapel te hoog (letterlijk!) wordt, komt de boel naar beneden, hetgeen ook het geval is als u van onderen probeert weg te nemen. Heel goed oppassen dus! Het eigenlijke werken met zo'n stapel gaat door middel van twee speciale stapelopdrachten. Als eerste is daar een speciale PUSH- of plaatsinstructie. Met zo'n instructie plaatst u iets bovenop de stapel.

Behalve deze instructie bestaat er ook nog een POP- of pakinstructie, een instructie waarmee u het laatst opgestapelde byte wegneemt.

Ook de SP is een 16-bits register en kan het gehele geheugen adresseren. Een goede programmeur haalt evenveel van de stapel AF als hij er OP zet, waardoor een stapel eigenlijk nooit erg groot hoeft te worden. Als geheugengebied voor de stapel kiest u een slimme plaats die u toch niet gebruikt. (In BASIC voorziet het CLEAR-commando in een dergelijke mogelijkheid.)

De stapel wordt voornamelijk gebruikt door de CALL-instructie. Een CALL is een subroutine in machinetaal. Wederom alles leuk en aardig maar als u een subroutine aanroeft, wilt u dat na afloop het programma terugkeert op de juiste plaats: precies na die CALL-instructie. Hiertoe wordt de PC op de stapel geplaatst, vervolgens krijgt de PC het adres van de subroutine als waarde. Is de subroutine eenmaal uitgevoerd dan wordt de 'oude' waarde weer van de stapel

genomen en gaat het programma weer verder vanaf de juiste plaats. Nu begrijpt u meteen waarom u zeer voorzichtig moet zijn met deze stapel; als u er teveel opzet of afhaalt gaat er onherroepelijk ergens iets fout.

## 2.4 De indexregisters: IX en IY

Indexregisters voorzien in een speciale manier van adresseren. De inhoud van een indexregister wordt (uiteraard op voorwaarde dat er een index-adresseringsvorm is gebruikt) automatisch opgeteld bij een eveneens opgegeven basiswaarde. Hierdoor kan de machinetaalprogrammeur vrij makkelijk een reeks van opeenvolgende waarden bereiken. Denk hiervoor aan het werken met tabellen van dezelfde lengte. De indexregisters worden ook vaak gebruikt bij de behandeling van **ARRAY**-achtige structuren.

In speciale gevallen kan het erg handig zijn om gebruik te maken van deze registers, maar dat moet iedere programmeur maar voor zichzelf uitmaken. U kunt ook heel goed zonder!

We komen hierop nog terug in het volgende hoofdstuk, waarin we alle mogelijke vormen van adresseren bekijken.

## 2.5 Het A-register of de accumulator

De accumulator is verreweg het meest gebruikte register in Z80-machinecodeprogramma's. Bijna alle belangrijke instructies doen iets met de inhoud van de accumulator. Vaak vindt een operatie plaats OP de waarde die zich op dat moment in de accumulator bevindt, waarna het resultaat weer IN deze accumulator wordt teruggezet.

Er zijn zelfs diverse handige dataverplaatsinstructies die alleen maar met behulp van de accumulator hun werk kunnen doen.

## 2.6 De zes algemene registers: B, C, D, E, H en L

Deze registers werken ongeveer als de accumulator, alleen is het gebruik helaas veel beperkter. Een zeer groot voordeel is het feit dat deze registers in drie paren aan elkaar kunnen worden gekoppeld, waardoor 16-bits berekeningen mogelijk zijn.

Ook kunt u de registerparen gebruiken voor adresseringsdoeleinden. Met name het HL-paar is dan erg handig!

Aan het B-register is trouwens ook nog een speciale 'DJNZ'-instructie gekoppeld. Meer hierover in een later stadium.

## 2.7 Het flag-register F

Minstens net zo belangrijk, zo niet belangrijker dan de accumulator is het flag-register. Dit register voorziet onder andere in speciale testmogelijkheden.

In dit F-register bevinden zich namelijk zes onderdelen met ieder een aparte functie. Sommige zijn afzonderlijk te testen en worden binnen een programma gebruikt om te kijken of bepaalde condities optreden: nul, niet nul, groter dan, kleiner dan, enz. Andere zijn alleen behulpzaam binnen bepaalde berekeningsvormen.

7	6	5	4	3	2	1	0
S	Z		H		P/U	M	C

Afb. 2.2. Het flag-register

- 1 De carry ofte wel het C-bit. De carry wordt gebruikt als extra bit bij berekeningen en/of schuifoperaties. U kunt testen met 'C' voor een carry op 1, en met 'NC' voor een carry op 0.
- 2 De subtract flag of ook wel het N-bit. Dit bit wordt geset bij een aftrekking. Het is alleen van belang bij BCD-berekeningen.
- 3 De parity/overflow-flag, het P/V-bit. Bij rekenkundige bewerkingen wordt deze flag geset als er een overflow optreedt. Bij logische bewerkingen wordt dit bit geset als het resultaat een even aantal enen oplevert, anders wordt dit bit 0 gemaakt. U test met 'PO' op een oneven pariteit en met 'PE' op een even pariteit.
- 4 De half carry-flag of het H-bit. Dit bit is alleen van belang bij BCD-berekeningen. Het H-bit wordt geset als er van bit 3 naar bit 4 een carry optreedt van bit 3 naar bit 4.
- 5 De zero-flag of het Z-bit. Een heel belangrijke test-mogelijkheid aangezien dit Z-bit wordt geset als het resultaat van een bewerking nul oplevert. Er staat een 0

in het Z-bit als een resultaat ongelijk is aan nul. U kunt testen met 'Z' op een resultaat dat gelijk is aan nul, u test met 'NZ' op een resultaat dat ongelijk is aan nul.

- 6 De sign-flag of het S-bit. Dit bit wordt 1 als het resultaat van een bewerking negatief is en 0 als dat resultaat positief is. U test met 'P' op een positieve waarde en met 'M' op een negatieve waarde.

Nog even alle tests op een rijtje:

<i>test</i>	<i>betekenis</i>
Z	resultaat gelijk aan nul
NZ	resultaat ongelijk aan nul
C	carry
NC	geen carry
PO	oneven pariteit
E	even pariteit
P	positief resultaat
M	negatief resultaat

## 2.8 Het interrupt-register I

Dit register wordt exclusief gebruikt in interrupt mode 2 (IM 2) om een speciale vector-call te genereren. In het I-register wordt het high-byte voor de sprong bewaard, terwijl het apparaat dat de interrupt aanlevert voor het low-byte zorgt (hierover straks meer).

## 2.9 Het refresh-register R

Dit register wordt gebruikt voor een automatische refresh van het dynamische geheugen. U zult dit register waarschijnlijk nooit in zelfgeschreven programma's gebruiken. Voor de volledigheid misschien leuk om te weten dat het bestaat.

## 2.10 De set 'verborgen' registers A' tot en met L'

U kunt deze 'extra' set registers met bijvoorbeeld het EXX-commando 'swappen' (verwisselen) met de gewone registers. Dit kan in enkele gevallen handig zijn. Normaal gesproken is het beter om alleen met de gewone register-set te werken, in verband met verwarringsverschijnselen.

### 3 Adresseringstechnieken

Laten we aannemen dat u nu begrijpt wat een Z80-microprocessor in huis heeft. Voordat we alle instructies gaan bespreken, is het misschien zinvol om eens te bekijken hoe u de diverse commando's kunt aanroepen, of hoe u ze gaat 'adresseren'!

Adresseren zullen we binnen dit kader verbasteren tot 'aanroepen'. Want in de verschillende vormen van aanroepmogelijkheden ligt de kracht van het werken met machinetaal.

Waar moet u aan denken? Een simpel voorbeeld: u wilt (gewoon in het dagelijks leven) informatie doorspelen. Het eenvoudigst en snelst gaat zoiet als u meteen spreekt met degene voor wie die informatie is bestemd. U kunt ook een brief posten, maar dan zit de PTT ertussen. Ook kunt u de telefoon gebruiken, of VAX, telex enz. Het hele systeem is erop gebaseerd dat de informatie op/aan het juiste adres komt. Dus: dezelfde informatie, verschillende doorgeefmethoden!

Ongeveer zo werkt u met de diverse aanroepformaten in machinecode. En er zijn nogal wat van die verschillende Z80-mogelijkheden:

- impliciete adressering
- onmiddellijke adressering
- absolute adressering
- zero page adressering
- relatieve adressering
- geïndexeerde adressering
- indirekte adressering
- bitadressering

Het is heel erg belangrijk dat u weet wat er allemaal mogelijk is binnen machine-taalprogrammering en dan met name bij het werken met de Z80-microprocessor. Slaat u het komende onderdeel dan ook zeker niet over. Het gebrek aan kennis kan u dan lelijk opbreken als u weer eens op zoek gaat naar het hoe en waarom van een instructie!

### **3.1 Impliciete adressering**

Bij 'impliciete adressering' gaat het vaak om bewerkingen die van toepassing zijn op één of twee specifieke registers. Het gaat om simpele instructies waarbij geen adres nodig is, omdat de instructie zichzelf al documenteert.

Hierbij moet u denken aan korte instructies die meestal ook zeer snel zijn, bijvoorbeeld: INC B, RLA, SBC A, OR A, enz.

Met de Z80-microprocessor in gedachten kunnen we eigenlijk nog wel wat instructies onder deze noemer meenemen. Hierbij gaat het om instructies die van toepassing zijn op onder andere 'registerparen'. Bijvoorbeeld: EX AF,AF', INC DE, DEC SP of LD A,(HL).

### **3.2 Onmiddellijke adressering**

In dit geval wordt de code die aangeeft om welke instructie het gaat, gevolgd door nog een enkel byte of in sommige gevallen zelfs door twee bytes (een word). Een byte voor een 8-bits getal, een word voor een 16-bits getal.

Het gaat hier dus altijd om zeer directe gevallen zoals bijv. LD BC,\$1234, LD B,\$00, SBC A,\$FF of AND \$11.

### **3.3 Absolute adressering**

Er zijn altijd drie bytes nodig voor absolute adressering. Het eerste byte bevat de opcode en de twee volgende bytes geven het gewenste adres. Het eerste byte is het 'low-byte' en levert een waarde tussen 0 en 255, het tweede byte noemen we het 'high-byte' en staat voor een waarde tussen 256 en 65280. Met andere woorden: het adres is de waarde 'high byte'  $\times$  256 + 'low byte'.

Dit gegeven wil nogal eens wat verwarring stichten. Let daarom goed op de tegenstrijdige verschijningsvormen, want in het geheugen staat een word dus in de volgorde laag/hoog opgeslagen; een monitor of assembler verwacht echter de (ons vertrouwde) hoog/laag-volgorde!

Het gaat dus om een adres. Voor bepaalde instructies zoals bijvoorbeeld de spronginstructie en de subroutine-instructie is dit adres alleen van belang als PC-waarde. Voor andere instructies, bijvoorbeeld een load-instructie of een store-instructie, is het adres nodig in verband met de waarde die zich op die geadresseerde geheugenplaats bevindt. Een paar voorbeelden: JP \$0000, CALL \$00C0, LD A,(\$E000), LD (\$E000),A enz.

### **3.4 Zero page-adressering**

Voor sommige microprocessors geldt dat het werken met zero page-adressering is verheven tot een aparte kunstvorm. In een Z80 daarentegen is er in feite maar één instructie die het speciaal in de zero page (De eerste 256 bytes van het geheugen, de geheugenplaatsen \$0000 tot en met \$00FF) zoekt.

Het gaat hier om een speciale CALL-instructie die in acht verschillende formaten voorkomt. Een zeer snelle instructie, die voorziet in een routineruimte van 8 bytes. Bijvoorbeeld: RST 00, RST 08, RST 16 enz.

### **3.5 Relatieve adressering**

Relatieve adressering gebruikt twee bytes. Het eerste byte verstrekkt de opcode en het tweede byte verzorgt de verplaatsing. In tegenstelling tot de absolute adresseringsvorm is de relatieve adresseringsvorm verplaatsbaar. Logisch, want de positieve of negatieve sprongcalculatie hangt alleen maar af van de waarde van de PC op dat moment. Een positieve verplaatsing (In het tweede byte staat bit 7 op nul: %0xxxxxxxx) wordt bij de PC opgeteld. Een negatieve verplaatsing (Bit 7 van het tweede byte is geset: %1xxxxxxxx) wordt van de PC afgetrokken. Bijvoorbeeld: JR \$C000 op het adres \$C010 is toegestaan. JR \$C000 op het adres \$D010 is niet toegestaan!

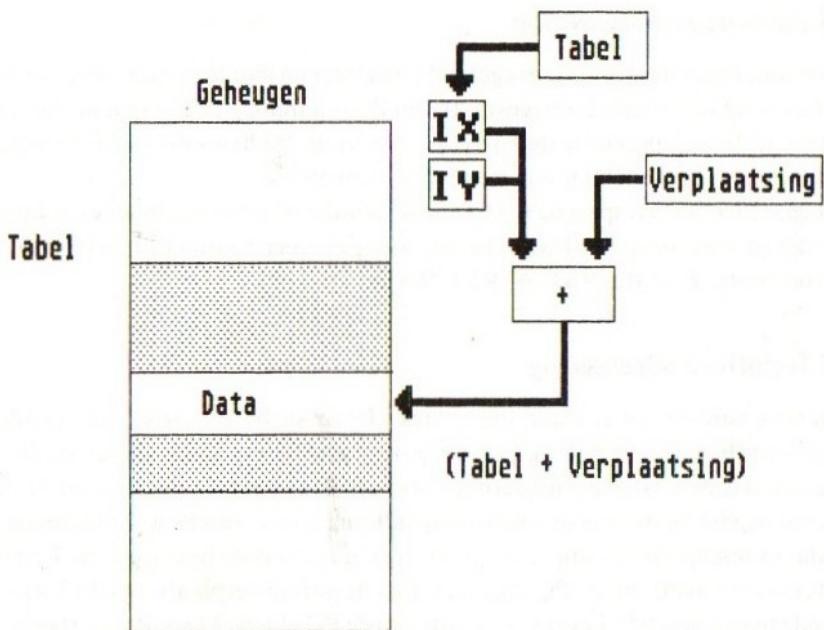
Let wel goed op, aangezien u heel gemakkelijk deze 'branch'-grens overschrijdt (De HERBYMON geeft gelukkig een waarschuwing).

### **3.6 Geïndexeerde adressering**

Geïndexeerde adressering is een adresseringsmogelijkheid die vooral handig is voor het werken met tabellen. Heel erg nuttig als de elementen uit zo'n tabel elkaar opvolgen, aangezien u dan van dezelfde bewerkingsinstructie gebruik kunt blijven maken terwijl u toch alle tabelingangen kunt adresseren.

Hoe werkt zoets? In het indexregister (IX of IY) staat een adres, meestal het begin van een tabel. Bij dit adres wordt een eventuele verplaatsing opgeteld en zodoende ontstaat een eindadres dat op ongeveer dezelfde manier als bij de directe adresseringsvorm kan worden gebruikt.

U heeft nu verschillende mogelijkheden. U verandert het indexregister (met speciale increment- of decrement-instructies) of u zorgt voor variërende verplaatsingen. In het ene geval doorloopt u een tabel met variërende lengte en in het



Afb. 3.1. Geïndexeerde adressering

andere geval kunt u een meervoudige tabel eenvoudig bewerken.

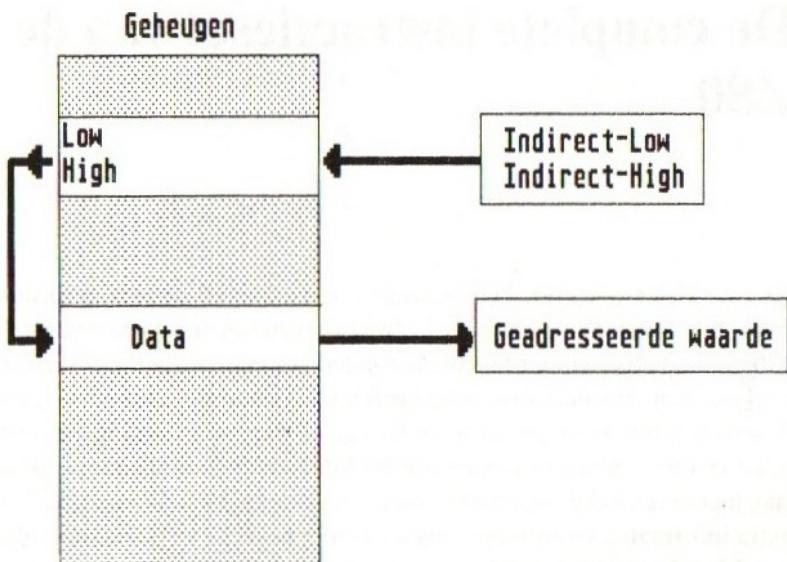
In Z80-machinecode kunt u deze geïndexeerde adressering op een aantal instructies loslaten. Bijvoorbeeld: ADD, INC, RLC, LD, SET enz.

### 3.7 Indirecte adressering

Indirecte adressering is een redelijk complexe adresseringsvorm waarbij gebruik wordt gemaakt van een pointer-structuur. U moet zich voorstellen dat u een bepaald adres aanspreekt. Op dit adres bevindt zich een 16-bits waarde die wijst naar een bepaalde geheugenlocatie. En pas in deze locatie vindt u het gewenste getal.

Eigenlijk is het een 'dubbele' adressering. U gebruikt deze adressering voornamelijk om erg snel een aantal verplaatste tabelgebieden te kunnen bereiken!

Met de Z80-microprocessor doet u dit voornamelijk met de registerparen BC, DE en HL. Deze dienen dan als link-adres. Bijvoorbeeld: LD A,(DE), LD B,(HL), OR (HL) enz.



Afb. 3.2. Indirecte adressering

### 3.8 Bitadressering

Bitadressering duidt op de mogelijkheid om aparte byte-onderdelen te kunnen aanspreken. Een byte bestaat uit acht afzonderlijke bits, genummerd van 0 tot en met 7.

In sommige gevallen heeft u voldoende aan bitinformatie en het zou in zo'n geval erg handig zijn om zo'n bit rechtstreeks te kunnen aanspreken. De Z80-microprocessor heeft ook hiervoor een aantal mogelijkheden, zoals de instructies BIT, RES en SET.

# 4 De complete instructieset van de Z80

U weet inmiddels van de hoed en de rand voor wat betreft de belangrijkste Z80-principes. We gaan nu dan ook de hele Z80-instructieset de revue laten passeren. In alfabetische volgorde worden alle afzonderlijke instructies besproken. Als u reeds bekend bent met de instructieset heeft u voldoende aan appendix A, waarin u ook terecht kunt voor de juiste codering, de flag-beïnvloeding en de tijdsduur...en niet te vergeten de juiste HERBYMON-syntax en alle mogelijkheden! We gaan hier alleen kijken wat die instructies nu precies doen. Waar nodig krijgt u ook extra informatie. Geen zorgen als u iets niet begrijpt; in de hierop volgende hoofdstukken komen we overal nog uitgebreid op terug.

## 4.1 Even vooraf

In appendix A staan alle bestaande Z80-instructies nog eens goed beschreven. Wilt u precies weten wat er mogelijk is of van welke HERBYMON-syntax u zich moet bedienen dan kunt u zich beter in deze appendix oriënteren. In het nu volgende overzicht onderzoeken we alleen hoe alle instructies nu precies werken. Het valt daarom buiten het kader van dit overzicht om van alle verschijningsvormen een instructie te geven.

We maken slim gebruik van enige afkortingen:

<i>rr</i>	<i>r</i>	<i>b</i>	<i>co</i>	<i>zp</i>
AF	A	0	Z	00
BC	B	1	NZ	08
DE	C	2	C	10
HL	D	3	NC	18
IX	E	4	PO	20
IY	H	5	PE	28
SP	L	6	P	30
		7	M	38

- Bij *rr* gaat het veelal om een speciale selectie, zie hiervoor de appendix!
- Kijk ook voor *co* nog even in de appendix. Afwijking in een heel enkel geval.
- *nn* wil zeggen dat we met een byte-waarde te maken hebben.
- *ii* is een afkorting van 'IX + *nn*' of 'IY + *nn*'.
- *nnnn* is een adres of een word.
- Een logische AND wordt als .and. genoteerd.
- Een logische OR wordt als .or. genoteerd.
- Een relatieve sprongcalculatie wordt als .rel. genoteerd.
- Een logische XOR wordt als .xor. genoteerd.)

## 4.2 De instructies op een rij

### ADC

Dit is een algemene optel-met-carry-instructie. De gegeven (tweede) operand wordt samen met de carry-flag uit het statusregister bij de eerste operand opgeteld waarna het resultaat weer in de eerste operand terechtkomt.

<i>Formaat</i>	<i>Functie</i>
ADC A, <i>r</i>	$A \leftarrow A + r + \text{carry}$
ADC A, <i>nn</i>	$A \leftarrow A + nn + \text{carry}$
ADC A,(HL)	$A \leftarrow A + (\text{HL}) + \text{carry}$
ADC A,( <i>ii</i> )	$A \leftarrow A + (ii) + \text{carry}$
ADC HL, <i>rr</i>	$HL \leftarrow HL + rr + \text{carry}$

### ADD

Dit is een algemene optelinstructie. De tweede operand wordt bij de eerste opgeteld waarna het resultaat in de eerste wordt geplaatst. (Let wel, zonder carry!)

<i>Formaat</i>	<i>Functie</i>
ADD A, <i>r</i>	$A \leftarrow A + r$
ADD A, <i>nn</i>	$A \leftarrow A + nn$
ADD A,(HL)	$A \leftarrow A + (\text{HL})$
ADD A,( <i>ii</i> )	$A \leftarrow A + (ii)$
ADD HL, <i>rr</i>	$HL \leftarrow HL + rr$
ADD IX, <i>rr</i>	$IX \leftarrow IX + rr$
ADD IY, <i>rr</i>	$IY \leftarrow IY + rr$

## **AND**

Dit is een logische AND-operatie die wordt uitgevoerd op de accumulator en de aangegeven operand. Het resultaat wordt weer in de accumulator geplaatst (let ook even op de waarheidstabbel).

<i>Formaat</i>	<i>Functie</i>
AND <i>r</i>	$A \leftarrow A \text{ .and. } r$
AND <i>nn</i>	$A \leftarrow A \text{ .and. } nn$
AND (HL)	$A \leftarrow A \text{ .and. } (HL)$
AND ( <i>ii</i> )	$A \leftarrow A \text{ .and. } (ii)$

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

## **BIT**

Dit is een test op het opgegeven bit uit een geadresseerde operand. Het resultaat beïnvloedt de zero-flag overeenkomstig de bitwaarde. (Bit '0' dan zero-flag geset, bit '1' dan zero-flag gecleared.)

<i>Formaat</i>	<i>Functie</i>
BIT <i>b,r</i>	$Z \leftarrow \text{'Test' bit } b \text{ in } r$
BIT <i>b,(HL)</i>	$Z \leftarrow \text{'Test' bit } b \text{ in } (HL)$
BIT <i>b,(ii)</i>	$Z \leftarrow \text{'Test' bit } b \text{ in } (ii)$

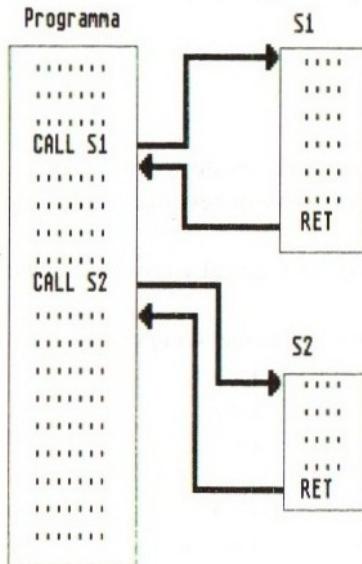
## **CALL**

Er wordt een subroutine op het opgegeven adres aangeroepen.

De inhoud van de programmateller wordt op de stapel gezet. Uit de twee bytes die op de instructiecode volgen, komt het nieuwe adres; dit wordt in de PC geplaatst.

*Formaat*  
CALL *nnnn*

*Functie*  
 $(SP - 1) \leftarrow PC\text{-high}$   
 $(SP - 2) \leftarrow PC\text{-low}$   
 $PC \leftarrow nnnn$   
 $SP \leftarrow SP - 2$



Afb. 4.1. Werken met subroutines

### CALL *cc*

Indien aan de voorwaarde *cc* wordt voldaan, wordt een subroutine op het meegegeven adres aangeroepen. (De inhoud van de huidige programmateller wordt op de stapel gezet. Uit de twee bytes die volgen op de instructiecode komt het nieuwe adres, dit wordt in de PC geplaatst.) Als daarentegen niet aan de voorwaarde *cc* wordt voldaan, wordt de instructie overgeslagen.

*Formaat*  
CALL *cc,nnnn*

*Functie*  
 $(SP - 1) \leftarrow PC\text{-high}$   
 $(SP - 2) \leftarrow PC\text{-low}$   
 $SP \leftarrow SP - 2$   
 $PC \leftarrow nnnn$   
\* \* \* mits *cc* waar\* \* \*

## CCF

Complementeer de carry-flag. ( $0 \rightarrow 1$  /  $1 \rightarrow 0$ )

<i>Formaat</i>	<i>Functie</i>
CCF	carry $\leftarrow$ carry .xor. 1

## CP

Vergelijk de opgegeven operand met de accumulator.

Dit komt in principe neer op een aftrekking zonder carry waarna het resultaat wordt weggegooid.

Na afloop kan eventueel op een aantal voorwaarden worden getest :

<i>Test</i>	<i>Accumulator is...in vergelijking met de operand</i>
NC	groter/gelijk ( $>=$ )
C	kleiner ( $<$ )
NZ	ongelijk ( $<>$ )
Z	gelijk ( $=$ )
NC/NZ	groter ( $>$ )
C/Z	kleiner/gelijk ( $= <$ )

<i>Formaat</i>	<i>Functie</i>
CP r	flags o.i.v. A - r
CP nn	flags o.i.v. A - nn
CP (HL)	flags o.i.v. A - (HL)
CP (ii)	flags o.i.v. A - (ii)

## CPD

Vergelijking met automatische verlaging.

De inhoud van het door (HL) geadresseerde byte wordt van de accumulator afgetrokken. Het resultaat wordt niet gebruikt. Vervolgens wordt de inhoud van zowel HL als BC met één verlaagd. (Wordt veel gebruikt voor gecontroleerde blokvergelijkingen.)

<i>Formaat</i>	<i>Functie</i>
CPD	flags o.i.v. A – (HL)
	HL $\leftarrow$ HL – 1
	BC $\leftarrow$ BC – 1

## CPDR

Blokvergelijking met automatische verlaging, alleen wordt dit net zolang herhaald tot ‘BC = 0’ of als ‘A = (HL)’.

De inhoud van het door (HL) geadresseerde byte wordt van de accumulator afgetrokken, het resultaat wordt niet gebruikt. Vervolgens wordt de inhoud van zowel HL als BC met één verlaagd. Nu wordt gekeken of ‘BC = 0’ of ‘A = (HL)’... is dit niet het geval dan wordt de instructie nogmaals uitgevoerd.

<i>Formaat</i>	<i>Functie</i>
CPDR	flags o.i.v. A – (HL)
	HL $\leftarrow$ HL – 1
	BC $\leftarrow$ BC – 1
	net zolang herhaald tot ‘BC = 0’ of ‘A = (HL)’

## CPI

Vergelijking met automatische verhoging.

De inhoud van het door (HL) geadresseerde byte wordt van de accumulator afgetrokken, het resultaat wordt niet gebruikt. Vervolgens wordt de inhoud van HL met één verhoogd en de inhoud van BC met één verlaagd. Wordt veel gebruikt voor gecontroleerde blokvergelijkingen.

<i>Formaat</i>	<i>Functie</i>
CPI	flags o.i.v. A – (HL)
	HL $\leftarrow$ HL + 1
	BC $\leftarrow$ BC – 1

## CPIR

Blokvergelijking met automatische verhoging, alleen wordt dit net zolang herhaald totdat ‘BC = 0’ of ‘A = (HL)’.

De inhoud van de door (HL) geadresseerde geheugenplaats wordt van de accumulator afgetrokken, het resultaat wordt niet gebruikt. Vervolgens wordt de inhoud van HL met één verhoogd en de inhoud van BC met één verlaagd. Nu wordt gekeken of 'BC = 0' of 'A = (HL)'. Is dit niet het geval dan wordt de instructie nogmaals uitgevoerd.

<i>Formaat</i>	<i>Functie</i>
CPIR	flags o.i.v. A – (HL)
	HL ← HL + 1
	BC ← BC – 1
	net zolang herhaald tot 'BC = 0' of 'A = (HL)'

## **CPL**

Complementeer de accumulator.

De inhoud van de accumulator wordt geïnverteerd (Bits die '0' zijn worden '1' en andersom) en weer in de accumulator geplaatst.

<i>Formaat</i>	<i>Functie</i>
CPL	A ← A .xor. \$FF

## **DAA**

Maak de inhoud van de accumulator 'packed BCD'.

Er wordt voorwaardelijk zes (6) opgeteld bij het rechter-en/of linkernybble van de accumulator. Deze optelling is onder meer afhankelijk van de inhoud van het statusregister. U gebruikt deze instructie veelal na een rekenkundige bewerking, om zo het resultaat (weer) BCD te krijgen.

<i>Formaat</i>	<i>Functie</i>
DAA	A ← packed BCD

## **DEC**

De geadresseerde operand wordt met één verlaagd waarna het resultaat weer in diezelfde operand terechtkomt.

<i>Formaat</i>	<i>Functie</i>
DEC r	$r \leftarrow r - 1$
DEC (HL)	$(HL) \leftarrow (HL) - 1$
DEC (ii)	$(ii) \leftarrow (HL) - 1$
DEC rr	$rr \leftarrow rr - 1$
DEC IX	$IX \leftarrow IX - 1$
DEC IY	$IY \leftarrow IY - 1$

## DI

Na deze instructie worden geen interrupts meer uitgevoerd. Een speciale interrupt-flag wordt 0 gemaakt, met als gevolg dat er geen maskeerbare interrupts meer worden ‘doorgelaten’. Na de EI-instructie worden wel weer interrupts toegestaan. (U kunt deze instructie ook in een interrupt gebruiken, om eventueel volgende interrupts te elimineren.)

<i>Formaat</i>	<i>Functie</i>
DI	$IFF \leftarrow 0$

## DJNZ

Het register B wordt met één verlaagd en er wordt relatief gesprongen als het resultaat van de verlaging NZ opleverde. (Deze instructie is natuurlijk ideaal voor programmalussen die altijd een van tevoren vastgestelde ‘doorlooplengte’ hebben. FOR...NEXT-simulatie dus.)

<i>Formaat</i>	<i>Functie</i>
DJNZ nn	$B \leftarrow B - 1$ als NZ dan $PC \leftarrow PC .rel. nn$ als Z dan volgende instructie

## EI

Na deze instructie worden weer interrupts uitgevoerd. Een speciale interrupt-flag wordt geset met als gevolg dat er weer maskeerbare interrupts worden doorgelaten.

<i>Formaat</i>	<i>Functie</i>
EI	IFF $\leftarrow 1$

## EX

Verwissel de opgegeven operands met elkaar. (Te vergelijken met het BASIC SWAP-commando.)

De inhoud van de geadresseerde operands wordt verwisseld met die van hun tegenhangers. Deze alternatieve registers bevinden zich in een aparte registerbank.

<i>Formaat</i>	<i>Functie</i>
EX AF,AF'	AF $\leftrightarrow$ AF'
EX DE,HL	DE $\leftrightarrow$ HL
EX (SP),HL	H $\leftrightarrow$ (SP + 1) L $\leftrightarrow$ (SP)
EX (SP),IX	IX-high $\leftrightarrow$ (SP + 1) IX-low $\leftrightarrow$ (SP)
EX (SP),IY	IY-high $\leftrightarrow$ (SP + 1) IY-low $\leftrightarrow$ (SP)

## EXX

Verwissel de BC-, DE- en HL- 'gewone' registers met hun 'alternatieve' tegenhangers.

<i>Formaat</i>	<i>Functie</i>
EXX	BC $\leftrightarrow$ BC' DE $\leftrightarrow$ DE' HL $\leftrightarrow$ HL'

## HALT

De werking van de Z80-microprocessor wordt stilgelegd.

Na de HALT-instructie worden alleen nog maar NOP's uitgevoerd. Dit alles gebeurt totdat een interrupt plaatsvindt of een reset wordt gegeven.

<i>Formaat</i>	<i>Functie</i>
HALT	CPU gaat ‘op slot’

## IM 0

Kies interrupt-mode 0. (Het apparaat of de chip dat de interrupt genereert, zet **de** uit te voeren instructie op de databus.)

<i>Formaat</i>	<i>Functie</i>
IM 0	Interrupt-mode 0

## IM 1

Kies interrupt-mode 1. (Er wordt een ‘RST 38’-instructie uitgevoerd als er een interrupt optreedt.)

<i>Formaat</i>	<i>Functie</i>
IM 1	Interrupt-mode 1

## IM 2

Kies interrupt-mode 2. (Het apparaat of de chip dat de interrupt genereert, zet **het** low-byte van een sprongadres op de databus. Het I-register zorgt voor het high-byte.)

<i>Formaat</i>	<i>Functie</i>
IM 2	Interrupt-mode 2

## IN r,(C)

Via het C-register wordt een randapparaat of een chip (zie appendix) geadresseerd. Dit randapparaat wordt uitgelezen en het resultaat komt in het gewenste register te staan.

<i>Formaat</i>	<i>Functie</i>
IN r,(C)	$r \leftarrow (C)$

### **IN A,(nn)**

Het door middel van ‘nn’ geadresseerde randapparaat (of een geadresseerde chip) wordt uitgelezen. Het resultaat wordt in de accumulator geplaatst.

<i>Formaat</i>	<i>Functie</i>
IN A,(nn)	$A \leftarrow (nn)$

### **INC**

De geadresseerde operand wordt met één verhoogd, waarbij het resultaat weer in diezelfde operand terecht komt.

<i>Formaat</i>	<i>Functie</i>
INC r	$r \leftarrow r + 1$
INC (HL)	$(HL) \leftarrow (HL) + 1$
INC (ii)	$(ii) \leftarrow (HL) + 1$
INC rr	$rr \leftarrow rr + 1$
INC IX	$IX \leftarrow IX + 1$
INC IY	$IY \leftarrow IY + 1$

### **IND**

Input met automatische verlaging. Het randapparaat dat door het C-register is geadresseerd, wordt uitgelezen. Het verkregen resultaat komt in het adres waar (HL) op dat moment naar wijst. Na afloop worden zowel HL als B met één verlaagd.

<i>Formaat</i>	<i>Functie</i>
IND	$(HL) \leftarrow (C)$
	$B \leftarrow B - 1$
	$HL \leftarrow HL - 1$

## **INDR**

Blok-input met automatische verlaging. Het randapparaat dat door het C-register is geadresseerd, wordt uitgelezen. Het resultaat komt in het adres waar (HL) op dat moment naar wijst. Na afloop worden zowel HL als B met één verlaagd. Als B ongelijk aan nul is, wordt de instructie nogmaals uitgevoerd.

<i>Formaat</i>	<i>Functie</i>
INDR	$(HL) \leftarrow (C)$ $B \leftarrow B - 1$ $HL \leftarrow HL - 1$ net zolang herhaald tot ' $B = 0$ '

## **INI**

Input met automatische verhoging. Het randapparaat dat door het C-register is geadresseerd, wordt uitgelezen. Het resultaat komt in het adres waar (HL) op dat moment naar wijst. Na afloop wordt HL met één verhoogd en B met één verlaagd.

<i>Formaat</i>	<i>Functie</i>
INI	$(HL) \leftarrow (C)$ $B \leftarrow B - 1$ $HL \leftarrow HL + 1$

## **INIR**

Blok-input met automatische verhoging. Het randapparaat dat door het C-register is geadresseerd, wordt uitgelezen, het resultaat komt in het adres waar (HL) op dat moment naar wijst. Na afloop wordt HL met één verhoogd en B met één verlaagd. Als B ongelijk aan nul is, wordt de instructie nogmaals uitgevoerd.

<i>Formaat</i>	<i>Functie</i>
INIR	$(HL) \leftarrow (C)$ $B \leftarrow B - 1$ $HL \leftarrow HL + 1$ net zolang herhaald tot ' $B = 0$ '

## **JP**

Er wordt gesprongen naar het adres dat in de opgegeven operand staat.  
De JR-instructie levert twee bytes aan die in de PC worden geplaatst. Bij de eerstvolgende instructie werkt het programma vanaf die plaats.

<i>Formaat</i>	<i>Functie</i>
JP nnnn	PC $\leftarrow$ nnnn
JP (HL)	PC $\leftarrow$ HL
JP (IX)	PC $\leftarrow$ IX
JP (IY)	PC $\leftarrow$ IY

## **JP cc**

Indien aan *cc* wordt voldaan, wordt een sprong naar het opgegeven adres gemaakt.

Indien *cc* klopt, levert de JR-instructie twee bytes aan die in de PC worden geplaatst. Bij de eerstvolgende instructie werkt het programma vanaf die plaats. Wordt niet aan *cc* voldaan dan wordt de instructie verder genegeerd.

<i>Formaat</i>	<i>Functie</i>
JP cc,nnnn	PC $\leftarrow$ nnnn
* * * mits <i>cc</i> waar* * *	

## **JR**

Er wordt een relatieve sprong gecalculeerd en gemaakt.

Er wordt gebruik gemaakt van het positief of negatief zijn van de meegeleverde waarde. Deze waarde wordt bij de PC opgeteld (of afgetrokken) om zo het nieuwe adres te verkrijgen. Hierdoor zijn sprongen van - 126 tot + 129 mogelijk.

<i>Formaat</i>	<i>Functie</i>
JR nn	PC $\leftarrow$ PC .rel. nn

## JR cc

Indien aan *cc* wordt voldaan, wordt een sprong gecalculoerd en gemaakt.

Als *cc* klopt, wordt er gebruik gemaakt van het positief of negatief zijn van de meegeleverde waarde. Deze waarde wordt bij de PC opgeteld (of ervan afgetrokken) om zo het nieuwe adres te verkrijgen. Als niet aan de gestelde conditie *cc* wordt voldaan, wordt de instructie verder genegeerd.

<i>Formaat</i>	<i>Functie</i>
JR <i>cc,nn</i>	$PC \leftarrow PC .rel. nn$ * * * mits <i>cc</i> waar* * *

## LD

Er wordt een ‘load/store’-instructie uitgevoerd. Dit is de meest veelzijdige Z80-instructie. In principe komt het neer op ‘data-verplaatsing’.

De inhoud van de tweede geadresseerde operand wordt in de eerste geadresseerde operand geplaatst. De LD-instructie dient dus tegelijkertijd als ‘load’- en ‘store’-instructie.

<i>Formaat</i>	<i>Functie</i>
LD <i>rr,(nnnn)</i>	$rr\text{-low} \leftarrow (nn)$ $rr\text{-high} \leftarrow (nn + 1)$
LD <i>rr,nnnn</i>	$rr \leftarrow nnnn$
LD <i>r,nn</i>	$r \leftarrow nn$
LD <i>r,r'</i>	$r \leftarrow r'$
LD <i>(rr),A</i>	$(rr) \leftarrow A$
LD <i>(HL),nn</i>	$(HL) \leftarrow nn$
LD <i>(HL),r</i>	$(HL) \leftarrow r$
LD <i>r,(ii)</i>	$r \leftarrow (ii)$
LD <i>(ii),nn</i>	$(ii) \leftarrow nn$
LD <i>(ii),r</i>	$(ii) \leftarrow r$
LD <i>A,(nnnn)</i>	$A \leftarrow (nnnn)$
LD <i>(nnnn),A</i>	$(nnnn) \leftarrow A$
LD <i>(nnnn),rr</i>	$(nn) \leftarrow rr\text{-low}$ $(nn + 1) \leftarrow rr\text{-high}$
LD <i>A,(rr)</i>	$A \leftarrow (rr)$
LD <i>A,I</i>	$A \leftarrow I$

LD I,A	$I \leftarrow A$
LD A,R	$A \leftarrow R$
LD R,A	$R \leftarrow A$
LD SP,rr	$SP \leftarrow rr$
LD r,(HL)	$r \leftarrow (HL)$

## LDD

Laden met automatische verlaging.

De inhoud van de door (HL) geadresseerde geheugenplaats wordt in de door (DE) geadresseerde locatie geplaatst. Vervolgens worden BC, DE en HL met één verlaagd.

<i>Formaat</i>	<i>Functie</i>
LDD	$(DE) \leftarrow (HL)$
	$DE \leftarrow DE - 1$
	$HL \leftarrow HL - 1$
	$BC \leftarrow BC - 1$

## LDLR

Blok laden met automatische verlaging. De inhoud van de door (HL) geadresseerde geheugenplaats wordt in de door (DE) geadresseerde locatie geplaatst. Vervolgens worden BC, DE en HL met één verlaagd. Nu wordt gekeken of 'BC = 0'. Als dit het geval is, wordt de lus onderbroken. Zo niet dan wordt de instructie nogmaals uitgevoerd.

<i>Formaat</i>	<i>Functie</i>
LDLR	$(DE) \leftarrow (HL)$
	$DE \leftarrow DE - 1$
	$HL \leftarrow HL - 1$
	$BC \leftarrow BC - 1$
	net zolang herhaald tot 'BC = 0'

## **LDI**

Laden met automatische verhoging.

De inhoud van de door (HL) geadresseerde geheugenplaats wordt in de door (DE) geadresseerde locatie geplaatst. Vervolgens worden DE en HL met één verhoogd. BC wordt met één verlaagd.

<i>Formaat</i>	<i>Functie</i>
LDI	(DE) $\leftarrow$ (HL)
	DE $\leftarrow$ DE + 1
	HL $\leftarrow$ HL + 1
	BC $\leftarrow$ BC - 1

## **LDIR**

Blok laden met automatische verhoging.

De inhoud van de door (HL) geadresseerde geheugenplaats wordt in de door (DE) geadresseerde locatie geplaatst. Vervolgens worden DE en HL met één verhoogd. BC wordt met één verlaagd. Nu wordt gekeken of BC gelijk is aan nul. Is dit het geval dan wordt de lus onderbroken. Zo niet dan wordt de instructie nogmaals uitgevoerd.

<i>Formaat</i>	<i>Functie</i>
LDIR	(DE) $\leftarrow$ (HL)
	DE $\leftarrow$ DE + 1
	HL $\leftarrow$ HL + 1
	BC $\leftarrow$ BC - 1
	net zolang herhaald tot 'BC = 0'

## **NEG**

Trek de accumulator af van nul (0).

<i>Formaat</i>	<i>Functie</i>
NEG	A $\leftarrow$ 0 - A

## NOP

'Loze' operatie, er wordt gedurende 1 cyclus niets gedaan.

<i>Formaat</i>	<i>Functie</i>
NOP	niets

## OR

Dit is een logische OR-operatie die wordt uitgevoerd op de accumulator en de aangegeven operand. Het resultaat wordt weer in de accumulator geplaatst. Let ook even op de waarheidstabbel.

<i>Formaat</i>	<i>Functie</i>
OR <i>r</i>	$A \leftarrow A . or. r$
OR <i>nn</i>	$A \leftarrow A . or. nn$
OR (HL)	$A \leftarrow A . or. (HL)$
OR ( <i>ii</i> )	$A \leftarrow A . or. (ii)$

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

## OTDR

Blok-output met automatische verlaging.

De inhoud van de door (HL) geadresseerde geheugenplaats wordt in de door (C) geadresseerde output-poort geplaatst. HL en B worden dan met één verlaagd. Als B nog niet gelijk aan nul is, wordt de instructie nogmaals uitgevoerd, in het andere geval wordt de lus onderbroken.

<i>Formaat</i>	<i>Functie</i>
OTDR	$(C) \leftarrow (HL)$
	$B \leftarrow B - 1$
	$HL \leftarrow HL - 1$
	net zolang herhaald tot ' $B = 0$ '

## **OTIR**

Blok-output met automatische verhoging.

De inhoud van de door (HL) geadresseerde geheugenplaats wordt in de door (C) geadresseerde output-poort geplaatst. HL wordt met één verhoogd en B wordt met één verlaagd. Als B nog niet gelijk aan nul is, wordt de instructie nogmaals uitgevoerd, in het andere geval wordt de lus onderbroken.

<i>Formaat</i>	<i>Functie</i>
OTIR	$(C) \leftarrow (HL)$
	$B \leftarrow B - 1$
	$HL \leftarrow HL + 1$
	net zolang herhaald tot ' $B = 0$ '

## **OUT (C),r**

Plaats het register  $r$  in de via '(C)' geadresseerde output-poort.

<i>Formaat</i>	<i>Functie</i>
OUT (C),r	$(C) \leftarrow r$

## **OUT (nn),A**

Plaats de accumulator in de door 'nn' geadresseerde output-poort.

<i>Formaat</i>	<i>Functie</i>
OUT (nn),A	$(nn) \leftarrow A$

## **OUTD**

Output met automatische verlaging.

De inhoud van de door (HL) geadresseerde geheugenplaats wordt in de door (C) geadresseerde output-poort geplaatst. HL en B worden dan met één verlaagd.

<i>Formaat</i>	<i>Functie</i>
OUTD	$(C) \leftarrow (HL)$
	$B \leftarrow B - 1$
	$HL \leftarrow HL - 1$

## OUTI

Output met automatische verhoging.

De inhoud van de door (HL) geadresseerde geheugenplaats wordt op de door (C) geadresseerde output-poort geplaatst. HL wordt met één verhoogd en B wordt met één verlaagd.

<i>Formaat</i>	<i>Functie</i>
OUTI	$(C) \leftarrow (HL)$
	$B \leftarrow B - 1$
	$HL \leftarrow HL + 1$

## POP

Haal twee bytes (een word) van de stapel en plaats deze in het geadresseerde registerpaar.

De inhoud van het byte bovenop de stapel komt in het low-byte van het geadresseerde registerpaar. Vervolgens wordt SP met één verhoogd. De inhoud van het byte dat daarna bovenop de stapel staat, komt in het high-byte van het geadresseerde registerpaar. Tot slot wordt SP nog met één verhoogd.

<i>Formaat</i>	<i>Functie</i>
POP rr	$rr\text{-low} \leftarrow (SP)$ $rr\text{-high} \leftarrow (SP + 1)$
	$SP \leftarrow SP + 2$
POP IX	$IX\text{-low} \leftarrow (SP)$ $IX\text{-high} \leftarrow (SP + 1)$
	$SP \leftarrow SP + 2$
POP IY	$IY\text{-low} \leftarrow (SP)$ $IY\text{-high} \leftarrow (SP + 1)$
	$SP \leftarrow SP + 2$

## PUSH

Plaats het geadresseerde registerpaar op de stapel.

SP wordt met één verlaagd en het high-byte van het geadresseerde registerpaar wordt op de stapel geplaatst. SP wordt opnieuw met één verlaagd en het low-byte van het geadresseerde registerpaar komt op de stapel.

<i>Formaat</i>	<i>Functie</i>
PUSH rr	$(SP - 1) \leftarrow rr\text{-high}$ $(SP - 2) \leftarrow rr\text{-low}$ $SP \leftarrow SP - 2$
PUSH IX	$(SP - 1) \leftarrow IX\text{-high}$ $(SP - 2) \leftarrow IX\text{-low}$ $SP \leftarrow SP - 2$
PUSH IY	$(SP - 1) \leftarrow IY\text{-high}$ $(SP - 2) \leftarrow IY\text{-low}$ $SP \leftarrow SP - 2$

## RES b

Dit is een reset op het opgegeven bit uit de geadresseerde operand.

<i>Formaat</i>	<i>Functie</i>
RES b,r	bit <i>b</i> in <i>r</i> $\leftarrow 0$
RES b,(HL)	bit <i>b</i> in (HL) $\leftarrow 0$
RES b,(ii)	bit <i>b</i> in (ii) $\leftarrow 0$

## RET

Een subroutine wordt afgebroken. De PC wordt (net als bij POP) van de stapel gehaald en het programma 'keert terug'.

<i>Formaat</i>	<i>Functie</i>
RET	PC-low $\leftarrow (SP)$ PC-high $\leftarrow (SP + 1)$ $SP \leftarrow SP + 2$

## **RET cc**

Indien aan *cc* wordt voldaan, wordt een subroutine afgebroken. Zie ook bij RET.

<i>Formaat</i>	<i>Functie</i>
RET cc	PC-low $\leftarrow$ (SP)
	PC-high $\leftarrow$ (SP + 1)
	SP $\leftarrow$ SP + 2
	* * * mits <i>cc</i> waar* * *

## **RETI**

Terugkeer van een interrupt.

Deze instructie gedraagt zich ongeveer als een gewone RET-instructie, met als enige verschil dat de processor nu het einde van een maskeerbare interrupt herkent. Het is in verband met eventuele interrupt-nesting zeer verstandig om vóór deze instructie een EI-instructie te plaatsen!

<i>Formaat</i>	<i>Functie</i>
RETI	PC-low $\leftarrow$ (SP)
	PC-high $\leftarrow$ (SP + 1)
	SP $\leftarrow$ SP + 2

## **RETN**

Terugkeer van een niet-maskeerbare interrupt.

Het gaat wederom om de terugkeer uit een interrupt, alleen wordt nu een speciale interrupt-flag hersteld. Dit om aan te geven dat maskeerbare interrupts weer zijn toegestaan. Een niet maskeerbare interrupt heeft altijd een hogere prioriteit dan een ‘gewone’ interrupt. Een hardware-reset heeft de allerhoogste prioriteit!

<i>Formaat</i>	<i>Functie</i>
RETN	PC-low $\leftarrow$ (SP)
	PC-high $\leftarrow$ (SP + 1)
	SP $\leftarrow$ SP + 2
	IFF1 $\leftarrow$ IFF2

## **RL**

Roteer de gespecificeerde operand en de carry naar links.

De inhoud van de carry komt in bit 0, bit 0 gaat naar bit 1,...,bit 6 gaat naar bit 7 en bit 7 komt in de carry te staan.

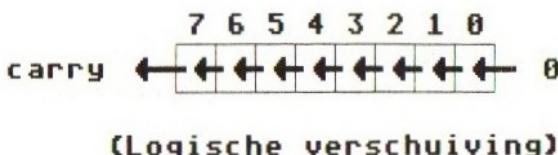
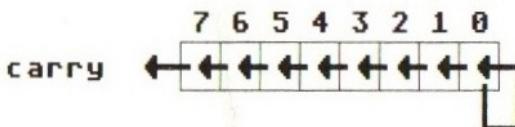
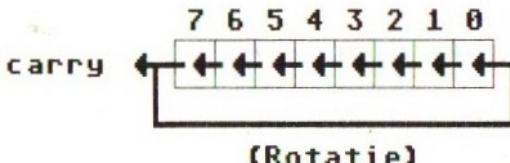
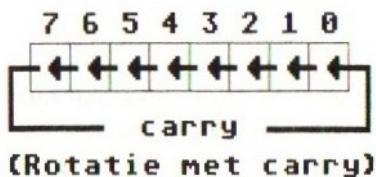
<i>Formaat</i>	<i>Functie</i>
RLA	carry $\leftarrow (7 \dots 0) \leftarrow$ carry A
RL r	carry $\leftarrow (7 \dots 0) \leftarrow$ carry r
RL (HL)	carry $\leftarrow (7 \dots 0) \leftarrow$ carry (HL)
RL (ii)	carry $\leftarrow (7 \dots 0) \leftarrow$ carry (ii)

## **RLC**

Roteer de gespecificeerde operand naar links.

Bit 7 gaat naar bit 0, bit 0 gaat naar bit 1,..., bit 6 gaat naar bit 7, bit 7 gaat naar de carry.

<i>Formaat</i>	<i>Functie</i>
RLCA	carry $\leftarrow (7 \dots 0) \leftarrow (7)$ A
RLC r	carry $\leftarrow (7 \dots 0) \leftarrow (7)$ r
RLC (HL)	carry $\leftarrow (7 \dots 0) \leftarrow (7)$ (HL)
RLC (ii)	carry $\leftarrow (7 \dots 0) \leftarrow (7)$ (ii)



Afb. 4.2. Verschuiven en roteren

## RLD

Roteer decimaal naar links.

Het low-nybble van de door (HL) geadresseerde geheugenplaats komt op het high-nybble van hetzelfde adres. Het high-nybble van dit adres zit inmiddels in het low-nybble van de accumulator, het low-nybble van de accumulator zit dan al in het low-nybble van de door (HL) geadresseerde geheugenplaats.

<i>Formaat</i>	<i>Functie</i>
RLD	$\rightarrow$ $(7..4)(3..0) (7..4)(3..0)$ $\leftarrow \leftarrow$ A            (HL)

## **RR**

Roteer de gespecificeerde operand en de carry naar rechts.

De inhoud van de carry komt in bit 7, bit 7 gaat naar bit 6,..., bit 1 gaat naar bit 0, bit 0 gaat naar de carry.

<i>Formaat</i>	<i>Functie</i>
RRA	carry → (7...0) → carry A
RR r	carry → (7...0) → carry r
RR (HL)	carry → (7...0) → carry (HL)
RR (ii)	carry → (7...0) → carry (ii)

## **RCR**

Roteer de gespecificeerde operand naar rechts. Bit 0 gaat naar bit 7, bit 7 gaat naar bit 6,..., bit 1 gaat naar bit 0, bit 0 gaat naar de carry.

<i>Formaat</i>	<i>Functie</i>
RRCA	(0) → (7...0) → carry A
RRC r	(0) → (7...0) → carry r
RRC (HL)	(0) → (7...0) → carry (HL)
RRC (ii)	(0) → (7...0) → carry (ii)

## **RRD**

Roteer decimaal naar rechts.

Het high-nybble van de door (HL) geadresseerde geheugenplaats komt op het low-nybble van hetzelfde adres. Het low-nybble van dit adres zit inmiddels in het

low-nybble van de accumulator, het low-nybble van de accumulator zit dan al in het high-nybble van de door (HL) geadresseerde geheugenplaats.

<i>Formaat</i>	<i>Functie</i>
RRD	$\rightarrow \rightarrow$
	(7..4)(3..0) (7..4)(3..0)
	$\leftarrow$
A	(HL)

### RST *zp*

Herstart op adres *zp* in de zero page.

Net als bij een CALL-instructie wordt de PC op de stapel gezet. Het high-byte voor de PC wordt 0, en het low-byte bevat de meegegeven waarde. Hierna komt het programma in een stukje zero page terecht.

<i>Formaat</i>	<i>Functie</i>
RST <i>zp</i>	$(SP - 1) \leftarrow PC\text{-high}$
	$(SP - 2) \leftarrow PC\text{-low}$
	$SP \leftarrow SP - 2$
	$PC\text{-high} \leftarrow 0$
	$PC\text{-low} \leftarrow zp$

### SBC

Dit is de algemene aftrek-minus-carry-instructie. De gegeven (tweede) operand wordt samen met de carry van de eerste operand afgetrokken, waarna het resultaat weer in de eerste operand terechtkomt.

<i>Formaat</i>	<i>Functie</i>
SBC A, <i>r</i>	$A \leftarrow A - r - \text{carry}$
SBC A, <i>nn</i>	$A \leftarrow A - nn - \text{carry}$
SBC A,(HL)	$A \leftarrow A - (HL) - \text{carry}$
SBC A,( <i>ii</i> )	$A \leftarrow A - (ii) - \text{carry}$
SBC HL, <i>rr</i>	$HL \leftarrow HL - rr - \text{carry}$

## **SCF**

Set de carry-flag.

<i>Formaat</i>	<i>Functie</i>
SCF	carry $\leftarrow 1$

## **SET**

Dit is een ‘set’ van het opgegeven bit uit de geadresseerde operand.

<i>Formaat</i>	<i>Functie</i>
SET $b,r$	bit $b$ in $r \leftarrow 1$
SET $b,(HL)$	bit $b$ in $(HL) \leftarrow 1$
SET $b,(ii)$	bit $b$ in $(ii) \leftarrow 1$

## **SLA**

Schuif de operand logisch naar links.

Bit 0 gaat naar bit 1,..., bit 6 gaat naar bit 7, bit 7 gaat naar de carry. Bit 0 wordt 0.

<i>Formaat</i>	<i>Functie</i>
SLA $r$	carry $\leftarrow (7\ldots0) \leftarrow 0$ $r$
SLA $(HL)$	carry $\leftarrow (7\ldots0) \leftarrow 0$ $(HL)$
SLA $(ii)$	carry $\leftarrow (7\ldots0) \leftarrow 0$ $(ii)$

## **SLI**

Schuif de operand rekenkundig naar links.

Bit 0 gaat naar bit 0, bit 0 gaat naar bit 1,..., bit 6 gaat naar bit 7, bit 7 gaat naar de carry.

<i>Formaat</i>	<i>Functie</i>
SLI <i>r</i>	$\text{carry} \leftarrow (7\ldots0) \leftarrow (0)$ <i>r</i>
SLI (HL)	$\text{carry} \leftarrow (7\ldots0) \leftarrow (0)$ (HL)
SLI ( <i>ii</i> )	$\text{carry} \leftarrow (7\ldots0) \leftarrow (0)$ ( <i>ii</i> )

## SRA

Schuif de operand rekenkundig naar rechts.

Bit 7 gaat naar bit 7, bit 7 gaat naar bit 6,..., bit 1 gaat naar bit 0, bit 0 gaat naar de carry.

<i>Formaat</i>	<i>Functie</i>
SRA <i>r</i>	$(7) \rightarrow (7\ldots0) \rightarrow \text{carry}$ <i>r</i>
SRA (HL)	$(7) \rightarrow (7\ldots0) \rightarrow \text{carry}$ (HL)
SRA ( <i>ii</i> )	$(7) \rightarrow (7\ldots0) \rightarrow \text{carry}$ ( <i>ii</i> )

## SRL

Schuif de operand logisch naar rechts.

Bit 7 gaat naar bit 6,..., bit 1 gaat naar bit 0, bit 0 gaat naar de carry. Bit 7 wordt 0.

<i>Formaat</i>	<i>Functie</i>
SRL <i>r</i>	$0 \rightarrow (7\ldots0) \rightarrow \text{carry}$ <i>r</i>
SRL (HL)	$0 \rightarrow (7\ldots0) \rightarrow \text{carry}$ (HL)
SRL ( <i>ii</i> )	$0 \rightarrow (7\ldots0) \rightarrow \text{carry}$ ( <i>ii</i> )

## SUB

Dit is een algemene aftrekinstructie. De tweede operand wordt van de eerste operand afgetrokken, waarna het resultaat weer in de eerste operand terechtkomt.

Formaat	Functie
SUB <i>r</i>	$A \leftarrow A - r$
SUB <i>nn</i>	$A \leftarrow A - nn$
SUB (HL)	$A \leftarrow A - (HL)$
SUB ( <i>ii</i> )	$A \leftarrow A - (ii)$

## XOR

Dit is een logische XOR-operatie die wordt uitgevoerd op de accumulator en de aangegeven operand. Het resultaat wordt weer in de accumulator geplaatst. Let ook even op de waarheidstabellen.

Formaat	Functie
XOR <i>r</i>	$A \leftarrow A . xor. r$
XOR <i>nn</i>	$A \leftarrow A . xor. nn$
XOR (HL)	$A \leftarrow A . xor. (HL)$
XOR ( <i>ii</i> )	$A \leftarrow A . xor. (ii)$

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

# 5 Getallen en hun representaties

Er bestaan verschillende manieren om over getallen en aantallen te spreken. U kunt op de markt een hand opsteken en de koopman geeft u meteen vijf van die super-grote-super-rode paprika's. Twee keer met uw hoofd knikken en de frietbakker pompt een 'dubbel met'.

De representatie van getallen zoals wij die kennen, is vrij simpel. We tellen van 0 tot en met 9, daarna schuiven we alles naar links om te kunnen geven dat er nog een tiental (een honderdtal, duizendtal enz.) voor staat.

*Voor de duidelijkheid nog een opmerking, om u straks de draad niet te laten verliezen:*

*Een binair getal wordt vaak voorafgegaan door een '%' -teken, dus %01010101 of %10101010101010 zijn binaire getallen.*

*Een decimaal getal herkent u aan het '#' -teken. #123, #4567 en #54321 zijn dus decimale getallen.*

*Een hexadecimaal getal wordt voorafgegaan door het '\$' -teken, waardoor \$EF, \$FEDC en \$0101 als hexadecimale getallen zijn bestempeld.*

## 5.1 Binair rekenen

Heel eenvoudig, dat gereken, maar een computer begrijpt er decimaal gezien absoluut NIETS van. Het willoze apparaat kan alleen maar in termen van schakelaartjes denken. Zo'n schakelaar heeft maar twee standen, AAN of UIT, '0' of '1'!

Eén zo'n nulletje of eentje heet een bit. 4 bits naast elkaar noemen we een nybble, 8 bits een byte, 16 bits een word en 32 bits een longword.

Een bit kan dus maar twee waarden representeren: 0 of 1. En nu komt de grap, want net als met 'ons' decimale stelsel kunnen we naar links opschuiven. Laten we dus ook eens twee bits naast elkaar zetten! Vier mogelijkheden: 00, 01, 10 en 11. De getallen 0 tot en met 3 natuurlijk, maar hoe komen we daar dan aan?

Het eerste bit geeft de 'enkel'-tallige waarden. Gezien als een macht van twee wordt dit:  $0 \times 0 = 0$  of  $1 \times 0 = 1$ .

Het tweede bit geeft de tien'-tallige waarden. Wederom gezien als een macht van twee:  $0 \times 2 = 0$  of  $1 \times 2 = 2$ .

Zo kunnen we verder denken, totdat we bij acht bits (een byte) komen:

<i>Bit</i>	<i>Maal</i>	<i>macht van 2</i>
0	1	$2^0$
1	2	$2^1$
2	4	$2^2$
3	8	$2^3$
4	16	$2^4$
5	32	$2^5$
6	64	$2^6$
7	128	$2^7$

Drie voorbeelden:

$$\begin{array}{rl} 00000000: & 0 \times 128 \\ & 0 \times 64 \\ & 0 \times 32 \\ & 0 \times 16 \\ & 0 \times 8 \\ & 0 \times 4 \\ & 0 \times 2 \\ & 0 \times 1 \\ \hline \text{totaal:} & 0 \end{array}$$

$$\begin{array}{rl} 11111111: & 1 \times 128 \\ & 1 \times 64 \\ & 1 \times 32 \\ & 1 \times 16 \\ & 1 \times 8 \\ & 1 \times 4 \\ & 1 \times 2 \\ & 1 \times 1 \\ \hline \text{totaal:} & 255 \end{array}$$

01110001:	$0 \times 128$
	$1 \times 64$
	$1 \times 32$
	$1 \times 16$
	$0 \times 8$
	$0 \times 4$
	$0 \times 2$
	$1 \times 1$

$$\text{totaal:} \quad \underline{\hspace{2cm}} \quad 113$$

Voor 16-bits waarden is het enige dat u hoeft te doen de tabel te verlengen. Als we een 16-bits getal als twee bytes zien, wordt het resultaat: ‘high-byte’  $\times$  256 + ‘low-byte’. Een voorbeeld:

$$0101010110101010 \rightarrow (\%01010101 \times \#256) + \%10101010$$

01010101:	$0 \times 128$	10101010:	$1 \times 128$
	$1 \times 64$		$0 \times 64$
	$0 \times 32$		$1 \times 32$
	$1 \times 16$		$0 \times 16$
	$0 \times 8$		$1 \times 8$
	$1 \times 4$		$0 \times 4$
	$0 \times 2$		$1 \times 2$
	$1 \times 1$		$0 \times 1$

$$\begin{array}{ll} \text{subtotaal:} & 85 \\ \text{totaal:} & 85 \times 256 + \end{array} \qquad \begin{array}{ll} \text{subtotaal:} & 170 \\ & 170 \rightarrow 21930 \end{array}$$

## 5.2 Hexadecimaal rekenen

Het vervelende van het rekenen met binaire getallen is de lengte en de onoverzichtelijkheid. Om een getal tussen 0 en 255 in te voeren, moet u acht keer op een toets drukken, terwijl de decimale invoer (met nullen aangevuld) slecht drie duwtjes vereist. Zeven, acht of negen bits... Van een afstand en/of zonder leesbril is het allemaal hetzelfde, dus onoverzichtelijk!

Decimaal en binair verdragen elkaar NIET, maar hexadecimaal en binair gelukkig WEL! Het hexadecimale stelsel is het 16-tallige stelsel. Binair rekenen gebruikt 2 waarden: 0 en 1. Decimaal rekenen gebruikt 10 waarden: 0, 1, 2, 3, 4, 5, 6, 7, 8 en

9. Hexadecimaal rekenen gebruikt er natuurlijk zestien: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

U voelt de bui al hangen, want ‘ $15 + 16 \times 15$ ’ is 255. Met andere woorden, aan twee hexadecimale digits heeft u al voldoende om een 8-bits waarde te representeren. Even een verhelderend staatje:

<i>Decimaal</i>	<i>Binair</i>	<i>Hexadecimaal</i>
00	0000	0
01	0001	1
02	0010	2
03	0011	3
04	0100	4
05	0101	5
06	0110	6
07	0111	7
08	1000	8
09	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Een hexadecimaal getal kan dus dezelfde waarde bereiken als vier binaire digits. Vier binaire digits noemden we een nybble. Het eerste nybble, het low-nybble, is dan te zien als de ‘enkele’ waarden. In een macht van 16 wordt dit:  $0 \times 0 = 0$  of  $1 \times 0 = 1$ ... of  $15 \times 0 = 15$ .

Het tweede nybble, het high-nybble, is dan te zien als de ‘tien’-tallige waarden, wederom als een macht van zestien:  $0 \times 16 = 0$  of  $1 \times 16 = 16$ ... of  $15 \times 16 = 240$ .

Zo kunnen we verder denken, totdat we bij acht bits (een byte) komen:

<i>Nybble</i>	<i>Maal</i>	<i>2-macht</i>
low	1	$16^0$
high	16	$16^1$

Drie voorbeelden:

$$\begin{array}{r} 00: \quad 0 \times 16 \\ \quad \quad 0 \times 1 \\ \hline \end{array}$$

totaal: 0

$$\begin{array}{r} FF: \quad 15 \times 16 \\ \quad \quad 15 \times 1 \\ \hline \end{array}$$

totaal: 255

$$\begin{array}{r} 71: \quad 7 \times 16 \\ \quad \quad 1 \times 1 \\ \hline \end{array}$$

totaal: 113

Voor 16-bits waarden is wederom het enige dat u hoeft te doen, de tabel te verlengen. Of, als we een 16-bits getal als twee bytes zien: het resultaat wordt 'high-byte'  $\times$  256 + 'low-byte'.

Een voorbeeld:

$$55AA \rightarrow (\$55 \times \#256) + \$AA$$

$$\begin{array}{r} 55: \quad 5 \times 16 \\ \quad \quad 5 \times 1 \\ \hline \end{array}$$

$$\begin{array}{r} AA: \quad 10 \times 16 \\ \quad \quad 10 \times 1 \\ \hline \end{array}$$

$$\begin{array}{r} \text{subtotaal: } 85 \\ \text{totaal: } 85 \times 256 + \end{array}$$

$$\begin{array}{r} \text{subtotaal: } 170 \\ \text{totaal: } 170 \rightarrow 21930 \end{array}$$

### 5.3 'Rekenen' in ASCII

Een heel ander verhaal is het werken met karakters. Cijfertjes leveren geen probleem op, maar de computer moet ook nog iets zichtbaar kunnen maken. Als op een gegeven moment een teken op uw beeldscherm staat, kan het zijn dat het onze A voorstelt. Een computer rekent absoluut niet in ABC-tjes, dus moeten we terug denken, ver terug...

Om te beginnen bestaat een 'A' uit een aantal puntjes. Als de MSX in de  $32 \times 24$

tekst-mode staat, bestaat een karakter uit een  $8 \times 8$ -raampje. 8 puntjes in de breedte staan voor een lijn-bit-patroon. 8 bits is wederom een heel byte, dus waar een pixel 'aan' staat, staat een bit op '1'. Waar een pixel 'uit' staat, staat een bit op '0'. Iedere lijn verbruikt één byte, dus 8 lijnen beslaan 8 bytes. Een voorbeeld:

...*....	%00100000	32
*.*....	%01010000	80
*...*....	%10001000	136
*...*....	%10001000	136
*****....	%11111000	248
*...*....	%10001000	136
*...*....	%10001000	136
.....	%00000000	0

Nu is de computer wel zo slim om u niet al die afzonderlijke puntjes te laten plotten. Ergens in het video-RAM bevindt zich een tabel waarin 256 verschillende  $8 \times 8$  patronen staan (zie appendix). Zo is er ook een bepaald stukje videogeheugen waar zich  $32 \times 34$  is 768 bytes bevinden. Deze bytes beslaan het beeldscherm. Elke waarde in dit stukje van 768 bytes is niet meer dan een wijzer naar die karakertabel met  $8 \times 8$  blokjes. Nu wordt het simpel. Als er bijvoorbeeld een 0 op het scherm staat, komt het nulde patroon uit de karakertabel op het beeld. Een 1 heeft het eerste patroon, enz. (Een simpele berekening: tabelpositie = tabelbegin +  $8 \times$  schermwaarde)

Een waarde staat voor een schermcode. We hebben 256 8-bits waarden en dus 256 verschillende karakters. U geeft een POKE en op het scherm wordt de representatie van deze POKE-waarde meteen zichtbaar!

Een heel ander verhaal krijgt u te horen bij het door de computer afdrukken van een teken. In het systeem-ROM (Ook wel BIOS genoemd) zitten een aantal handige routines die via een hoofd-ingang (CHPUT) worden aangeroepen. We noemen: 'het scherm wissen', 'cursor links-boven', 'regel wissen', 'cursor naar beneden', 'een linefeed', enz. En niet te vergeten... Het afdrukken van een teken! AL deze bewerkingen moeten dus ook worden aangeroepen. Ook hier hebben we weer 256 verschillende mogelijkheden. Als we de afdrukroutine met de code 32 aanroepen, wordt er een spatie op het beeldscherm afgedrukt. De waarde 13 daarentegen zet de cursor in de meest linkse kolom. Ook zijn er codes voor 'HOME', 'CLS', enz.

Alle ASCII-codes (Want afdrukken door het systeem vindt via dit speciale gestandaardiseerde protocol plaats!) hebben een speciale betekenis en ook werken zij meestal op een eventueel aangesloten printer.

Op dit alles komen we in de programmeervoorbeelden nog uitgebreid terug.

# 6 Enkele basistechnieken

Om u alvast een klein beetje op weg te helpen, volgen nu een aantal simpele voorbeelden, geschreven in HERBYMON. Per voorbeeld zullen we kijken wat het doet, maar vooral in welke programmeersituaties u zich van dergelijke technieken kunt bedienen.

Zoals het voorbeeld is afgedrukt, kunt u het met het assembleercommando invoeren. Ieder voorbeeld is in subroutineformaat geschreven. Als u vanuit de HERBYMON de C-instructie gebruikt, keert de routine dus netjes terug!

## 6.1 Relatieve lussen

```
.; C000 06 10          LD B,$10
.; C002 C5            PUSH BC
.; C003 CD C0 00      CALL $00C0
.; C006 C1            POP BC
.; C007 10 F9          DJNZ @C002
.; C009 C9            RET
.; CO0A 00            NOP
.; CO0B 00            NOP
.; CO0C 00            NOP
.; CO0D 00            NOP
.; CO0E 00            NOP
.; CO0F 00            NOP
.; CO10 3E 10          LD A,$10
.; CO12 32 00 E0      LD ($E000),A
.; CO15 CD C0 00      CALL $00C0
.; CO18 21 00 E0      LD HL,$E000
.; CO1B 35            DEC (HL)
.; CO1C 20 F7          JR NZ,@CO15
.; CO1E C9            RET
.; CO1F 00            NOP
.; CO20 3E 08          LD A,$08
.; CO22 32 00 E0      LD ($E000),A
.; CO25 3E 10          LD A,$10
.; CO27 32 01 E0      LD ($E001),A
.; CO2A CD C0 00      CALL $00C0
.; CO2D 21 01 E0      LD HL,$E001
```

```

.; C030 35      DEC (HL)
.; C031 20 F7    JR NZ,@C02A
.; C033 2B      DEC HL
.; C034 35      DEC (HL)
.; C035 20 EE    JR NZ,@C025
.; C037 C9      RET

```

Heel vaak wilt u een aantal handelingen herhalen. Hiervoor moet u dus een lus maken. Een lus kunt u op verschillende manieren ontwerpen. Hieronder vindt u de beschrijving van een drietal mogelijkheden.

Nog een algemene opmerking: de JR-instructies zijn ook te vervangen door JP-instructies. Houd er echter rekening mee dat dit uw routines langer maakt.

De eerste lus is geïndexeerd. Het programma maakt handig gebruik van het B-register en de DJNZ-instructie. U kunt de instructie LD B,\$.. eenvoudig aanpassen om zo tot het gewenste aantal herhalingen te komen (start met C C000).

Het tweede voorbeeld gebruikt een geheugenplaats voor het tellen van de herhalingen. Een groot voordeel van een dergelijke lus is het feit dat u vrij van de registers gebruik kunt maken. Pas als u aan het einde van de lus bent gekomen, test u op het al dan niet herhalen (start met C C010).

Het derde lus is een dubbele lus. In dit geval heeft u een binnen- en een buitenlus (start met C C020).

## 6.2 Logische bewerkingen

```

.; C000 5F      LD E,A
.; C001 E6 00    AND $00
.; C003 47      LD B,A
.; C004 7B      LD A,E
.; C005 E6 FF    AND $FF
.; C007 4F      LD C,A
.; C008 7B      LD A,E
.; C009 E6 55    AND $55
.; C00B 57      LD D,A
.; C00C 7B      LD A,E
.; C00D E6 AA    AND $AA
.; C00F 5F      LD E,A
.; C010 C9      RET
.; C011 00      NOP

```

. ; C012 00	NOP
. ; C013 00	NOP
. ; C014 00	NOP
. ; C015 00	NOP
. ; C016 00	NOP
. ; C017 00	NOP
. ; C018 00	NOP
. ; C019 00	NOP
. ; C01A 00	NOP
. ; C01B 00	NOP
. ; C01C 00	NOP
. ; C01D 00	NOP
. ; C01E 00	NOP
. ; C01F 00	NOP
. ; C020 5F	LD E,A
. ; C021 F6 00	OR \$00
. ; C023 47	LD B,A
. ; C024 7B	LD A,E
. ; C025 F6 FF	OR \$FF
. ; C027 4F	LD C,A
. ; C028 7B	LD A,E
. ; C029 F6 55	OR \$55
. ; C02B 57	LD D,A
. ; C02C 7B	LD A,E
. ; C02D F6 AA	OR \$AA
. ; C02F 5F	LD E,A
. ; C030 C9	RET
. ; C031 00	NOP
. ; C032 00	NOP
. ; C033 00	NOP
. ; C034 00	NOP
. ; C035 00	NOP
. ; C036 00	NOP
. ; C037 00	NOP
. ; C038 00	NOP
. ; C039 00	NOP
. ; C03A 00	NOP
. ; C03B 00	NOP
. ; C03C 00	NOP
. ; C03D 00	NOP
. ; C03E 00	NOP
. ; C03F 00	NOP
. ; C040 5F	LD E,A
. ; C041 EE 00	XOR \$00
. ; C043 47	LD B,A
. ; C044 7B	LD A,E
. ; C045 EE FF	XOR \$FF
. ; C047 4F	LD C,A
. ; C048 7B	LD A,E
. ; C049 EE 55	XOR \$55
. ; C04B 57	LD D,A

```
.; C04C 7B          LD A,E
.; C04D EE AA      XOR $AA
.; C04F 5F          LD E,A
.; C050 C9          RET
```

We gaan eens kijken wat het effect is van de drie logische bewerkingen of operaties AND, OR en XOR. Met het \*-commando plaatst u een waarde in het A-register. De routine laat met de logische bewerking een viertal waarden 'los' waarna het resultaat in de B-, C-, D- en E-registers terechtkomt. (Het verkregen resultaat ziet u als de routine terugkeert).

Het eerste voorbeeld demonstreert de OR-instructie (start met C C000).

Het tweede voorbeeld demonstreert de AND-instructie (start met C C020). De AND-instructie wordt ook wel gebruikt om bepaalde bitpatronen te testen. U wilt bijvoorbeeld weten of in een bepaalde byte het getal 3 zit. In %000000011, %11000011, %01010111, enz. Als de laatste twee bits allebei op 1 staan, zit er dus een drie in het getal verborgen. In zo'n geval test u met AND \$03, direct gevolgd door CP \$03. Nu kunt u aan de hand van de Z-test te weten komen hoe het met die 3 zat!

Het derde voorbeeld demonstreert de XOR-instructie (start met C C040).

### 6.3 Dat gedoe met die carry

```
.; C000 37          SCF
.; C001 3F          CCF
.; C002 ED 5A      ADC HL,DE
.; C004 C9          RET
.; C005 00          NOP
.; C006 00          NOP
.; C007 00          NOP
.; C008 37          SCF
.; C009 ED 5A      ADC HL,DE
.; C00B C9          RET
.; C00C 00          NOP
.; C00D 00          NOP
.; C00E 00          NOP
.; C00F 00          NOP
.; C010 37          SCF
.; C011 3F          CCF
.; C012 19          ADD HL,DE
```

```

.; C013 C9      RET
.; C014 00      NOP
.; C015 00      NOP
.; C016 00      NOP
.; C017 00      NOP
.; C018 37      SCF
.; C019 19      ADD HL,DE
.; C01A C9      RET
.; C01B 00      NOP
.; C01C 00      NOP
.; C01D 00      NOP
.; C01E 00      NOP
.; C01F 00      NOP
.; C020 37      SCF
.; C021 3F      CCF
.; C022 ED 52      SBC HL,DE
.; C024 C9      RET
.; C025 00      NOP
.; C026 00      NOP
.; C027 00      NOP
.; C028 37      SCF
.; C029 ED 52      SBC HL,DE
.; C02B C9      RET
.; C02C 00      NOP
.; C02D 00      NOP
.; C02E 00      NOP
.; C02F 00      NOP

```

HERBYMON (V1.1)

SP	IY	IX	HL	DE	BC	A
----	----	----	----	----	----	---

\* 9D17 DE92 F38B BFA3 0007 0300 00

16-bits rekenen gaat prima met een Z80-microprocessor. Het enige probleem vormt de carry. Wanneer moet dat ding nu WEL en wanneer NIET geset zijn? Hieronder zes voorbeelden voor de ADC-, ADD- en SBC-instructies. Met het \*-commando voert u een aantal probeerwaarden in. U roept de subroutines aan en na afloop moet u eens goed op de waarde van het HL-registerpaar letten.

Zes routines, zes startinstructies: C C000, C C008, C C010, C C018, C C020 en C C028.

## 6.4 Rekenen in de praktijk

```

.; C000 2A 00 E0      LD HL,($E000)
.; C003 ED 5B 02 E0      LD DE,($E002)
.; C007 19      ADD HL,DE

```

```

.; C008 22 04 E0 LD ($E004),HL
.; CO0B C9 RET
.; CO0C 00 NOP
.; CO0D 00 NOP
.; CO0E 00 NOP
.; CO0F 00 NOP
.; C010 21 01 E0 LD HL,$E001
.; C013 3A 00 E0 LD A,($E000)
.; C016 86 ADD A,(HL)
.; C017 32 02 E0 LD ($E002),A
.; C01A C9 RET
.; C01B 00 NOP
.; C01C 00 NOP
.; C01D 00 NOP
.; C01E 00 NOP
.; C01F 00 NOP
.; C020 B7 OR A
.; C021 2A 00 E0 LD HL,($E000)
.; C024 ED 5B 02 E0 LD DE,($E002)
.; C028 ED 52 SBC HL,DE
.; C02A 22 04 E0 LD ($E004),HL
.; C02D C9 RET
.; C02E 00 NOP
.; C02F 00 NOP
.; C030 21 01 E0 LD HL,$E001
.; C033 3A 00 E0 LD A,($E000)
.; C036 96 SUB (HL)
.; C037 32 02 E0 LD ($E002),A
.; C03A C9 RET

```

Optellen en aftrekken.. Een viertal voorbeelden waarin zowel met 16 bits als met 8 bits wordt gewerkt. Het optellen demonstreert het  $A + B \rightarrow C$ -principe. Het aftrekken gaat volgens de methode  $A - B \rightarrow C$ .

Helaas bestaat er geen instructie  $SUB rr,rr$ . Hierdoor moeten we bij het derde voorbeeld eerst de carry clearen. In plaats van SCF/CCF gebruiken we  $OR A$ , wat precies hetzelfde oplevert: de carry op 0...

U start met C C000, C C010, C C020 en C030. Om de gevolgen te bekijken, geeft u een memory-display met M E000 E006. Verander ook de E000-E004 waarden eens!

# 7 De MSX en de buitenwereld

## 7.1 De routines en hun doel

In dit deel gaan we een aantal invoer- en uitvoertechnieken beschouwen. Invoertechnieken in verband met alles dat de computer IN gaat. Uitvoertechnieken in verband met alles dat de computer UIT gaat.

Vooral dit laatste is van belang. Er bestaan eigenlijk geen computerprogramma's die iets ondernemen zonder dat de programmeur (of de gebruiker) daarvan in kennis wordt gesteld. Toegegeven: sommige processen zijn zó geheim dat een programmeur wil dat de gebruiker daarvan niet op de hoogte wordt gesteld, maar u als software-ontwikkelaar wilt natuurlijk precies weten wat er op welk moment staat te gebeuren en of dat allemaal wel correct gebeurt!

Aan de andere kant moet een computer de te verwerken informatie kunnen opnemen, iets dat op verschillende manieren mogelijk is: via het toetsenbord, door middel van de joystick, door iets van een tape te laden enz.

In de volgende hoofdstukken krijgt u een achttal listings voorgesloteld. Deze omvatten de belangrijkste IO-principes; van het doodgewone 'tekstje printen' tot en met een ingewikkeld 'cassette operating systeem'.

## 7.2 Nog even vooraf

De zo dadelijk te verwerken listings van subroutines komen in drie delen. Eerst krijgt u het gedeelte als een labelassemblerlisting (beter leesbaar en met enige structuur); dit is de listing die we zullen bespreken. Ten tweede ziet u een geheugendump, die u klakkeloos met HERBYMON kunt invoeren, temeer daar er vaak teksten moeten worden overgenomen. Het derde en laatste deel is een stukje 'pure disassembly'. Dit is de code die u in de HERBYMON te voorschijn haalt en gaat gebruiken.

Het startadres van een demonstratieprogramma is heel gemakkelijk te traceren. U kijkt in de assemblerlisting naar de Z80-code rond het label INIT, en u zoekt in de door u geproduceerde code naar dezelfde serie instructies.

**TIP:** Het is ook mogelijk om de code te verplaatsen. U doet dit met de T-optie uit de HERBYMON, waarna u niet moet vergeten om de sprongadressen aan te passen.

# 8 Tekstje printen

Dit voorbeeld demonstreert een simpele afdrukroutine die op de twee eerste scherm-modes (40\*24 en 321\*24) van toepassing is. We gaan ervan uit dat er ergens in het geheugen een aantal ASCII-codes werden geplaatst waarvan de laatste byte een nul is.

Het is de bedoeling dat u het adres van deze string in het HL-registerpaar plaatst en vervolgens naar onze PRINT-subroutine springt.

De ASCII-codes vindt u in appendix B. In principe kunt u gedocumenteerde ASCII-codes gebruiken om de tekst precies daar te plaatsen waar u vindt dat deze dient op te dagen.

TIP: gebruik eventueel de BIOS-routine POSIT om de cursor ergens neer te zetten.

```
10000 ORIGIN ($c000)
10010 --
10020 $00a2 CONSTANT ChPut
10030 $00ba CONSTANT IscNtc
10040 --
10050 CODE
10060 --
10070 :Tekst1 .ch "hallo wereld!"
10080         .by $0a,$0d,$00
10090 :Tekst2 .ch "dit is een simpel voorbeeld van
ascii-uitvoer..."
10100        .by $0a,$0d,$00
10110 :Tekst3 .ch "het werkt in 'screen mode 0' en 'screen mode
1'."
10120         .by $0a,$0d,$00
10130 --
10140 :Print
10150     ld a,(hl)
10160     or a
10170     retif z
10180     call #ChPut
10190     inc hl
10200     jp #Print
10210 --
10220 :Init
10230     ld hl,#Tekst1
```

```

10240    call #Print
10250    ld hl,#Tekst2
10260    call #Print
10270    ld hl,#Tekst3
10280    call #Print
10290    :Wacht
10300    call #IscNtc
10310    jp #wacht
10320    --
10330    ENDCODE

```

.: C000 48 41 4C 4C	"HALL"	.: C04C 20 49 4E 20	" IN "
.: C004 4F 20 57 45	"O WE"	.: C050 27 53 43 52	"'SCR"
.: C008 52 45 4C 44	"RELD"	.: C054 45 45 4E 20	"EEN "
.: C00C 21 0A 0D 00	"!...."	.: C058 4D 4F 44 45	"MODE"
.: C010 44 49 54 20	"DIT "	.: C05C 20 30 27 20	" O '
.: C014 49 53 20 45	"IS E"	.: C060 45 4E 20 27	"EN '"
.: C018 45 4E 20 53	"EN S"	.: C064 53 43 52 45	"SCRE"
.: C01C 49 4D 50 45	"IMPE"	.: C068 45 4E 20 4D	"EN M"
.: C020 4C 20 56 4F	"L VO"	.: C06C 4F 44 45 20	"ODE "
.: C024 4F 52 42 45	"ORBE"	.: C070 31 27 2E 0A	"1'.."
.: C028 45 4C 44 20	"ELD "	.: C074 0D 00 7E B7	"...."
.: C02C 56 41 4E 20	"VAN "	.: C078 C8 CD A2 00	"...."
.: C030 41 53 43 49	"ASCI"	.: C07C 23 C3 76 CO	"#.v."
.: C034 49 2D 55 49	"I-UI"	.: C080 21 00 C0 CD	"!..."
.: C038 54 56 4F 45	"TVOE"	.: C084 76 C0 21 10	"v.!."
.: C03C 52 2E 2E 2E	"R..."	.: C088 C0 CD 76 C0	"..v."
.: C040 0A 0D 00 48	"...H"	.: C08C 21 43 C0 CD	"!C.."
.: C044 45 54 20 57	"ET W"	.: C090 76 C0 CD BA	"v..."
.: C048 45 52 4B 54	"ERKT"	.: C094 00 C3 92 CO	"...."

.: C076 7E	LD A,(HL)
.: C077 B7	OR A
.: C078 C8	RET Z
.: C079 CD A2 00	CALL \$00A2
.: C07C 23	INC HL
.: C07D C3 76 CO	JP \$C076
.: C080 21 00 C0	LD HL,\$C000
.: C083 CD 76 CO	CALL \$C076
.: C086 21 10 C0	LD HL,\$C010
.: C089 CD 76 CO	CALL \$C076
.: C08C 21 43 C0	LD HL,\$C043
.: C08F CD 76 CO	CALL \$C076
.: C092 CD BA 00	CALL \$00BA
.: C095 C3 92 CO	JP \$C092

**PRINT** is de eigenlijke subroutine. U heeft het adres van de string in HL staan en u maakt vervolgens een **CALL** naar deze subroutine.

Eerst vergelijken we de door HL geadresseerde waarde met 0 (die OR A dient om de zero-flag wel of niet te zetten). Is de waarde gelijk aan nul dan is de subroutine klaar en keren we netjes terug naar het hoofdprogramma. Zo niet dan nemen we het karakter mee naar de BIOS-routine **CHPUT** en het systeem zorgt voor de schermuitvoer. Daarna verhogen we HL om zo de volgende waarde uit de string te adresseren, waarna we weer naar **PRINT** springen.

ISCNTC controleert op CTRL/STOP en springt terug naar BASIC als deze toetscombinatie is ingedrukt.

# 9 Invoer via het toetsenbord

Een voorbeeld van een routine die invoer via het toetsenbord accepteert. U roept deze routine aan zonder parameters, waarna de cursor verschijnt. Nu kan de gebruiker een ASCII-string met een door u (de programmeur) bepaalde lengte invoeren. Deze string wordt zowel op het scherm als op een door de programmeur bepaalde geheugenplaats gebufferd. Met Backspace wist u het laatst ingevoerde karakter. Ongewenste toetsen worden 'wegge-beep'. Deze routine is zeer simpel te wijzigen qua invoerlengte en invoergedrag.

Met deze subroutine kunt u nu alle toetsenbordinvoer regelen om deze later op uw gemak te bekijken. Pas als de gebruiker op de RETURN-toets drukt, wordt de routine afgebroken.

TIP: wijzig de lengtetest voor geformateerde invoer eens. Verander de test voor ASCII-uitsluiting, bijvoorbeeld alleen numerieke tekens, of juist alleen alfanumerieke tekens.

```
10000 ORIGIN ($c000)
10010 --
10020 $006c CONSTANT Initxt
10030 $009f CONSTANT ChGet
10040 $00a2 CONSTANT ChPut
10050 $00ba CONSTANT IscNtc
10060 $00c0 CONSTANT Beep
10070 $0156 CONSTANT KilBuf
10080 --
10090 CODE
10100 --
10110 :Teller .by $00
10120 :Wijzer .wo $0000
10130 --
10140 :Regel .ch "....."
10150 .by $0a,$0d,$00
10160 :Prompt .ch "invoer:"
10170 .by $00
10180 :Show .by $0a,$0d
10190 .ch "->"
10200 .by $00
10210 --
```

```
10220 :Print
10230     ld a,(hl)
10240     or a
10250     retif z
10260     call #ChPut
10270     inc hl
10280     jp #Print
10290 --
10300 :SetUp
10310     ld b,#16
10320     ld hl,#Regel
10330     ld (#Wijzer),hl
10340 :setUp0
10350     ld a,#46
10360     ld (hl),a
10370     inc hl
10380     djnz #setUp0
10390     ld a,#0
10400     ld (#Teller),a
10410     call #KilBuf
10420     ld hl,#Prompt
10430     jp #Print
10440 --
10450 :Delete
10460     ld a,(#Teller)
10470     cp #0
10480     IF nz
10490         THEN
10500             dec a
10510             ld (#Teller),a
10520             dec hl
10530             ld hl,(#Wijzer)
10540             dec hl
10550             ld (#Wijzer),hl
10560             ld a,#46
10570             ld (hl),a
10580             ld a,#$08
10590             call #ChPut
10600             ld a,#$20
10610             call #ChPut
10620             ld a,#$08
10630             call #ChPut
10640     FI
10650     ret
10660 --
10670 :Erbij
10680     cp #32
10690     jpif cs,#Beep
10700     cp #127
10710     jpif nc,#Beep
10720     push af
10730     ld a,(#Teller)
```

```

10740 cp #16
10750 IF nz
10760 THEN
10770 inc a
10780 ld (#Teller),a
10790 ld hl,(#Wijzer)
10800 pop af
10810 ld (hl),a
10820 inc hl
10830 ld (#Wijzer),hl
10840 call #ChPut
10850 ret
10860 FI
10870 pop af
10880 ret
10890 --
10900 :Invoer
10910 call #SetUp
10920 :Invoer0
10930 call #IscNtc
10940 call #ChGet
10950 cp #13
10960 retif z
10970 cp #8
10980 IF z
10990 THEN
11000 call #Delete
11010 ELSE
11020 call #Erbij
11030 FI
11040 jp #Invoer0
11050 --
11060 :Init
11070 call #InitTxt
11080 :Init0
11090 call #Invoer
11100 ld hl,#Show
11110 call #Print
11120 ld hl,#Regel
11130 call #Print
11140 jp #Init0
11150 --
11160 ENDCODE

```

..: C000 00 00 00 2E "...."	..: C010 2E 2E 2E 0A "...."
..: C004 2E 2E 2E 2E "...."	..: C014 0D 00 49 4E ".IN"
..: C008 2E 2E 2E 2E "...."	..: C018 56 4F 45 52 "VOER"
..: C00C 2E 2E 2E 2E "...."	..: C01C 3A 00 0A 0D "...."

```

.: C020 2D 3E 00 7E    ">.."
.: C024 B7 C8 CD A2    "...."
.: C028 00 23 C3 23    ".#.#"
.: C02C C0 06 10 21    "...!"
.: C030 03 C0 22 01    "...."
.: C034 C0 3E 2E 77    ".>.w"
.: C038 23 10 FA 3E    "#..>"
.: C03C 00 32 00 C0    "...."
.: C040 CD 56 01 21    ".V.!"
.: C044 16 C0 C3 23    "...#"
.: C048 C0 3A 00 C0    "...."
.: C04C FE 00 CA 6F    "...o"
.: C050 C0 3D 32 00    "...=2."
.: C054 C0 2B 2A 01    ".+*."
.: C058 C0 2B 22 01    ".+.."
.: C05C C0 3E 2E 77    ".>.w"
.: C060 3E 08 CD A2    ">.."
.: C064 00 3E 20 CD    ".> ."
.: C068 A2 00 3E 08    "...>."
.: C06C CD A2 00 C9    "...."
.: C070 FE 20 DA C0    "...."
.: C074 00 FE 7F D2    "..."


```

```

.; C023 7E          LD A,(HL)
.; C024 B7          OR A
.; C025 C8          RET Z
.; C026 CD A2 00    CALL $00A2
.; C029 23          INC HL
.; C02A C3 23 C0    JP $C023
.; C02D 06 10        LD B,$10
.; C02F 21 03 C0    LD HL,$C003
.; C032 22 01 C0    LD ($C001),HL
.; C035 3E 2E        LD A,$2E
.; C037 77          LD (HL),A
.; C038 23          INC HL
.; C039 10 FA        DJNZ @C035
.; C03B 3E 00        LD A,$00
.; C03D 32 00 C0    LD ($C000),A
.; C040 CD 56 01    CALL $0156
.; C043 21 16 C0    LD HL,$C016
.; C046 C3 23 C0    JP $C023
.; C049 3A 00 C0    LD A,($C000)
.; C04C FE 00        CP $00
.; C04E CA 6F C0    JP Z,$C06F
.; C051 3D          DEC A
.; C052 32 00 C0    LD ($C000),A
.; C055 2B          DEC HL
.; C056 2A 01 C0    LD HL,($C001)
.; C059 2B          DEC HL


```

```

.; C05A 22 01 C0 LD ($C001),HL
.; C05D 3E 2E LD A,$2E
.; C05F 77 LD (HL),A
.; C060 3E 08 LD A,$08
.; C062 CD A2 00 CALL $00A2
.; C065 3E 20 LD A,$20
.; C067 CD A2 00 CALL $00A2
.; C06A 3E 08 LD A,$08
.; C06C CD A2 00 CALL $00A2
.; C06F C9 RET
.; C070 FE 20 CP $20
.; C072 DA C0 00 JP C,$00C0
.; C075 FE 7F CP $7F
.; C077 D2 C0 00 JP NC,$00C0
.; C07A F5 PUSH AF
.; C07B 3A 00 C0 LD A,($C000)
.; C07E FE 10 CP $10
.; C080 CA 93 C0 JP Z,$C093
.; C083 3C INC A
.; C084 32 00 C0 LD ($C000),A
.; C087 2A 01 C0 LD HL,($C001)
.; C08A F1 POP AF
.; C08B 77 LD (HL),A
.; C08C 23 INC HL
.; C08D 22 01 C0 LD ($C001),HL

```

Het enige dat u zelf hoeft te doen, is een CALL-opdracht naar INVOER te geven. Eerst doet de invoerroutine nog even een SETUP, waarin de invoerregel wordt gewist. De invoer heeft een maximumlengte van 16 tekens. Op het scherm wordt de invoerruimte voor de duidelijkheid gevuld met punten. De toetsenbordbuffer wordt geleegd en er wordt een invoer-prompt afgedrukt op de plaats waar de cursor zich op dat moment bevindt.

INVOER0 test op de CTRL/STOP-combinatie en probeert vervolgens een karakter van het toetsenbord te halen. Het karakter dat werd opgehaald, vergelijken we met 13 (carriage return) om eventueel de subroutine te onderbreken.

Geen 13, dus een ander karakter. Misschien wel ASCII-code 8 (backspace)? Zo ja, dan naar de DELETE-routine. Zo niet dan naar de ERBIJ-routine.

**TIP:** wijzig de stringlengte om wat variatie aan te brengen. De eerste ERBIJ-tests zorgen voor de juiste teken-acceptatie. Deze kunt u natuurlijk ook veranderen. Als voorbeeld noemen we CP \$30 en CP \$39 om alleen maar getallen door te laten.

```
.; C090 C3 A2 00      JP $00A2
.; C093 F1          POP AF
.; C094 C9          RET
.; C095 CD 2D C0      CALL $C02D
.; C098 CD BA 00      CALL $00BA
.; C09B CD 9F 00      CALL $009F
.; C09E FE OD          CP $0D
.; COA0 C8          RET Z
.; COA1 FE 08          CP $08
.; COA3 C2 AC C0      JP NZ,$COAC
.; COA6 CD 49 C0      CALL $C049
.; COA9 C3 AF C0      JP $COAF
.; COAC CD 70 C0      CALL $C070
.; COAF C3 98 C0      JP $C098
.; COB2 CD 6C 00      CALL $006C
.; COB5 CD 95 C0      CALL $C095
.; COB8 21 1E C0      LD HL,$C01E
.; COBB CD 23 C0      CALL $C023
.; COBE 21 03 C0      LD HL,$C003
.; COC1 CD 23 C0      CALL $C023
.; COC4 C3 B5 C0      JP $C0B5
```

### Toelichting

ERBIJ kijkt of het teken is toegestaan. Zo niet dan keren we terug. Zo ja, dan controleren we nog even de lengte. Is de string te lang dan keren we terug. Als alles in orde is, plaatsen we dit teken even op de stapel om vervolgens de teller en de string-pointer te verhogen. Daarna halen we de waarde van de stapel af (OPGELET!) en plaatsen deze in de string.

Als de invoerroutine is beëindigd, weet u waar de string zich bevindt. Ook weet u de lengte. Het is verder aan uw programma om deze invoer verder te verwerken.

DELETE kijkt naar de lengte van de invoer. Is deze gelijk aan nul dan valt er niets te wissen en keert de routine terug. Is de invoerlengte ongelijk aan nul dan verlagen we de teller en de string-pointer, we plaatsen een punt in de string en de cursor moet terug naar voren.

# 10 De typemachinesimulatie

Als u een printer tot uw beschikking heeft, kunt u eens kijken hoe u deze vanuit een machinetaalprogramma aanstuurt. Voor het gemak even een directe toets/printer-schakeling. Het hangt van uw printer af hoe de codes worden verwerkt. Sommige printers wachten met uitvoeren totdat een CR (carriage return) wordt gesignaleerd, andere printers duwen alles meteen door de kop op het papier.

Let vooral op de speciale uitzondering die wordt gemaakt voor een linefeed, en op de escape-uitsluiting. U kunt zelf beslissen of u een andere uitzondering wilt maken, of misschien wilt u juist wél gebruik maken van alle extra printer-mogelijkheden. In dit laatste geval verdient het enige aanbeveling om de handleiding van uw afdrukker eens netjes uit te spitten.

TIP: bouw deze routine om naar een 'string-afdrukker' en u bent al aardig op weg naar een simpele tekstverwerker! Gebruik hiervoor bijvoorbeeld de afdrukroutine uit hoofdstuk 8.

```
10000 ORIGIN ($c000)
10010 --
10020 $006c CONSTANT InitTxt
10030 $009f CONSTANT ChGet
10040 $00a2 CONSTANT ChPut
10050 $00a5 CONSTANT LptOut
10060 $00c0 CONSTANT Beep
10070 $0156 CONSTANT KilBuf
10080 --
10090 $f3ae CONSTANT Lin40
10100 --
10110 CODE
10120 --
10130 :CharOut
10140     push af
10150     call #ChPut
10160     call #LptOut
10170     pop af
10180     ret
10190 --
10200 :TypeMach
10210     call #KilBuf
10220     REPEAT
```

```

10230      call #ChGet
10240      call #CharOut
10250      cp #13
10260      IF z
10270          THEN
10280              ld a,#10
10290                  call #CharOut
10300                  push af
10310                  call #Beep
10320                  pop af
10330          FI
10340          cp #27
10350      UNTIL z
10360      ret
10370  --
10380 :Init
10390     ld a,#40
10400     ld (#Lin40),a
10410     call #InitTxt
10420     jp #TypeMach
10430  --
10440 ENDCODE

```

```

.: C000 F5 CD A2 00    "...."
.: C004 CD A5 00 F1    "...."
.: C008 C9 CD 56 01    "..V."
.: C00C CD 9F 00 CD    "...."
.: C010 00 C0 FE 0D    "...."
.: C014 C2 21 C0 3E    ".!.>"
.: C018 0A CD 00 C0    "...."
.: C01C F5 CD C0 00    "...."
.: C020 F1 FE 1B C2    "...."
.: C024 0C C0 C9 3E    "...>"
.: C028 28 32 AE F3    "(2.."
.: C02C CD 6C 00 C3    ".1.."
.: C030 09 C0 52 45    ".RE"
.
```

```

.; C000 F5           PUSH AF
.; C001 CD A2 00     CALL $00A2
.; C004 CD A5 00     CALL $00A5
.; C007 F1           POP AF
.; C008 C9           RET
.; C009 CD 56 01     CALL $0156
.; C00C CD 9F 00     CALL $009F
.; C00F CD 00 C0     CALL $C000
.; C012 FE 0D         CP $0D

```

```
.; C014 C2 21 C0      JP NZ,$C021
.; C017 3E 0A      LD A,$0A
.; C019 CD 00 C0      CALL $C000
.; C01C F5      PUSH AF
.; C01D CD C0 00      CALL $00C0
.; C020 F1      POP AF
.; C021 FE 1B      CP $1B
.; C023 C2 0C C0      JP NZ,$C00C
.; C026 C9      RET
.; C027 3E 28      LD A,$28
.; C029 32 AE F3      LD ($F3AE),A
.; C02C CD 6C 00      CALL $006C
.; C02F C3 09 C0      JP $C009
```

### Toelichting

Als u een goede MSX-printer bezit, kan er absoluut niets misgaan. Verder hangt het van uw eigen fantasie af hoe exotisch u de over te sturen codes wilt hebben.

TYPEMACH initialiseert de keyboard-buffer en haalt een karakter met behulp van de BIOS-routine CHGET. Dit karakter wordt vervolgens meteen uitgevoerd met onze CHAROUT-subroutine. We PUSH-en de waarde, sturen het scherm en de printer (BIOS-LPTOUT) aan, waarna we de ingevoerde waarde POP-pen. Vervolgens kijken we of het om een carriage return (13) gaat, om er meteen nog even een linefeed achteraan te jagen (niet voor iedere printer noodzakelijk). Nog even een slinger aan de bel en als het om een escape-code ging, keren we terug naar ons hoofd-programma. Heel erg simpel allemaal!

TIP: u kunt escape-codes toestaan om bijvoorbeeld over te gaan naar een andere karakterset, of om grafisch te printen.

# 11 Spelletje spelen?

De joystick is als homecomputer-accessoire niet meer weg te denken. Tenslotte is iedere huiscomputer ook een veredelde speelkameraad, vandaar deze joystick-routine. Erg handig is het feit dat u altijd meteen over de juiste joystick/cursor/spatie-informatie kunt beschikken. Dit in tegenstelling tot het werken met de CHGET-routine, waar het gebruik afhankelijk is van de grootte van de toetsenbordbuffer en/of de status van dat moment.

TIP: een uitstekende mogelijkheid om uw programma op twee manieren van invoer te voorzien. Aan de ene kant de gewone CHGET-invoer en aan de andere kant een directe controle. Handig voor wachtlussen en/of onderbrekingen!

```
10000 ORIGIN ($c000)
10010 --
10020 $006c CONSTANT Initxt
10030 $00a2 CONSTANT ChPut
10040 $00ba CONSTANT IscNtc
10050 $00d5 CONSTANT GtStck
10060 $00d8 CONSTANT GtTrig
10070 --
10080 CODE
10090 --
10100 :Hoog    .ch "stick omhoog"
10110      .by $0a,$0d,$00
10120 :Laag    .ch "stick omlaag"
10130      .by $0a,$0d,$00
10140 :Links   .ch "stick links"
10150      .by $0a,$0d,$00
10160 :Rechts  .ch "stick rechts"
10170 :        .by $0a,$0d,$00
10180 :Vuur    .ch "*** vuur ***"
10190      .by $0a,$0d,$00
10200 --
10210 :Print
10220   ld a,(hl)
10230   or a
10240   retif z
10250   call #ChPut
10260   inc hl
```

```
10270      jp #Print
10280      --
10290      :Up
10300      ld hl,#Hoog
10310      jp #Print
10320      --
10330      :Down
10340      ld hl,#Laag
10350      jp #Print
10360      --
10370      :Left
10380      ld hl,#Links
10390      jp #Print
10400      --
10410      :Right
10420      ld hl,#Rechts
10430      jp #Print
10440      --
10450      :Fire
10460      ld hl,#Vuur
10470      jp #Print
10480      --
10490      :JoyStick
10500      call #IscNtc
10510      ld a,#0
10520      call #GtStck
10530      ld b,a
10540      cp #1
10550      callif z,#Up
10560      ld a,b
10570      cp #5
10580      callif z,#Down
10590      ld a,b
10600      cp #3
10610      callif z,#Left
10620      ld a,b
10630      cp #7
10640      callif z,#Right
10650      ld a,#0
10660      call #GtTrig
10670      cp #$ff
10680      callif z,#Fire
10690      jp #Joystick
10700      --
10710      :Init
10720      call #IniTxt
10730      jp #Joystick
10740      --
10750 ENDCODE
```

..: C000	53	54	49	43	"STIC"	..: C054	21	00	C0	C3	"!..."
..: C004	4B	20	4F	4D	"K OM"	..: C058	4A	C0	21	0F	"J.!"
..: C008	48	4F	4F	47	"HOOG"	..: C05C	C0	C3	4A	C0	".J."
..: C00C	0A	0D	00	53	".S"	..: C060	21	1E	C0	C3	"!..."
..: C010	54	49	43	4B	"TICK"	..: C064	4A	C0	21	2C	"J.!"
..: C014	20	4F	4D	4C	" OML"	..: C068	C0	C3	4A	C0	".J."
..: C018	41	41	47	0A	"AAG."	..: C06C	21	3B	C0	C3	"!..."
..: C01C	0D	00	53	54	".ST"	..: C070	4A	C0	CD	BA	"J..."
..: C020	49	43	4B	20	"ICK "	..: C074	00	3E	00	CD	".>..."
..: C024	4C	49	4E	4B	"LINK"	..: C078	D5	00	47	FE	".G."
..: C028	53	0A	0D	00	"S..."	..: C07C	01	CC	54	C0	".T."
..: C02C	53	54	49	43	"STIC"	..: C080	78	FE	05	CC	"x..."
..: C030	4B	20	52	45	"K RE"	..: C084	5A	C0	78	FE	"Z.x."
..: C034	43	48	54	53	"CHTS"	..: C088	03	CC	60	C0	".!"
..: C038	0A	0D	00	2A	".**"	..: C08C	78	FE	07	CC	"x..."
..: C03C	2A	2A	20	56	"** V"	..: C090	66	C0	3E	00	"f.>."
..: C040	55	55	52	20	"UUR "	..: C094	CD	D8	00	FE	"...."
..: C044	2A	2A	2A	0A	"***."	..: C098	FF	CC	6C	C0	".1."
..: C048	0D	00	7E	B7	". . ."	..: C09C	C3	72	C0	CD	".r.."
..: C04C	C8	CD	A2	00	". . ."	..: COA0	6C	00	C3	72	"l..r"
..: C050	23	C3	4A	C0	"#.J."	..: COA4	C0	F0	00	21	". . .!"

..: C04A	7E	LD A,(HL)		
..: C04B	B7	OR A		
..: C04C	C8	RET Z		
..: C04D	CD	CALL \$00A2		
..: C050	23	INC HL		
..: C051	C3	4A	C0	JP \$C04A
..: C054	21	00	C0	LD HL,\$C000
..: C057	C3	4A	C0	JP \$C04A
..: C05A	21	0F	C0	LD HL,\$C00F
..: C05D	C3	4A	C0	JP \$C04A
..: C060	21	1E	C0	LD HL,\$C01E
..: C063	C3	4A	C0	JP \$C04A
..: C066	21	2C	C0	LD HL,\$C02C
..: C069	C3	4A	C0	JP \$C04A
..: C06C	21	3B	C0	LD HL,\$C03B
..: C06F	C3	4A	C0	JP \$C04A
..: C072	CD	BA	00	CALL \$00BA
..: C075	3E	00	LD A,\$00	
..: C077	CD	D5	00	CALL \$00D5
..: C07A	47	LD B,A		
..: C07B	FE	01	CP \$01	
..: C07D	CC	54	C0	CALL Z,\$C054
..: C080	78	LD A,B		
..: C081	FE	05	CP \$05	
..: C083	CC	5A	C0	CALL Z,\$C05A
..: C086	78	LD A,B		

```
.; C087 FE 03      CP $03
.; C089 CC 60 C0    CALL Z,$C060
.; C08C 78          LD A,B
.; C08D FE 07      CP $07
.; C08F CC 66 C0    CALL Z,$C066
.; C092 3E 00      LD A,$00
.; C094 CD D8 00    CALL $00D8
.; C097 FE FF      CP $FF
.; C099 CC 6C C0    CALL Z,$C06C
.; C09C C3 72 C0    JP $C072
.; C09F CD 6C 00    CALL $006C
.; COA2 C3 72 C0    JP $C072
```

### Toelichting

We maken gebruik van de BIOS-routine GTSTCK. Deze kunt u aanroepen met 0 (keyboard), 1 (joystick A), of 2 (joystick B) in de accumulator. Voor de vuurknop gebruiken we BIOS-GTTRIG met identieke waarden in de accumulator.

De JOYSTICK-routine kijkt naar CTRL/STOP, om vervolgens de joystick-waarde uit te lezen. Deze waarde reden we door hem in het B-register te zetten. Hierna is het een kwestie van vergelijken en, afhankelijk van de waarde, een passende tekst afdrukken (met de reeds bekende print-routine).

We halen de vuurknopwaarde op (0 = niets / 255 of \$FF = vuur) en drukken af als er inderdaad wordt 'geschoten'.

Het uitlezen van een joystick heeft normaliter alleen maar met het spelen van spelletjes te maken. Spelbesturing is vaak erg complex, dus in dergelijke gevallen hangt het van de grenzeloze fantasie van de programmeur af of er met de invoer ook daadwerkelijk iets wordt gedaan. Het blijft altijd handig om van een test zoals in dit voorbeeld gebruik te maken. U simuleert dan een CASE-structuur en dat is wel zo praktisch.

# 12 Bitbeschouwingen

Het geheugen van uw MSX-machine staat in feite boordevol getallen. In bytes (8 bits) gezien, spreken we over waarden tussen 0 en 255, in words (16 bits) over getallen tussen 0 en 65535. Maar nog altijd decimaal!

Een schrijfwijze die beter aansluit op de flip/flop-situatie in uw MSX-computer is de binaire notatie: allemaal nullen en enen. U kunt dan ook veel sneller aflezen hoe het is afgelopen met die vreemde bitbewerkingen.

De bijbehorende subroutine demonstreert hoe u binaire waarden op uw scherm krijgt afdrukkt. Dit alles kan op twee manieren en met een binair ‘procentje’ ervoor. Een byte-waarde of een word-waarde, u plaatst het getal op een zelfgekozen adres, u springt naar de juiste routine en voilà!

TIP: gebruik deze routine zolang uw programma’s zich nog in hun testfase bevinden. Ten eerste ziet u óf er wat gebeurt en ten tweede weet u hóe het gebeurt.

```
10000 ORIGIN ($c000)
10010 --
10020 $00a2 CONSTANT ChPut
10030 $00d8 CONSTANT GtTrig
10040 --
10050 CODE
10060 --
10070 :Byt    .by $00
10080 :Word   .wo $0000
10090 --
10100 :Procent
10110     ld a,#37
10120     jp #ChPut
10130 --
10140 :NewLin
10150     ld a,#10
10160     call #ChPut
10170     ld a,#13
10180     jp #ChPut
10190 --
10200 :Binair
10210     ld (#Byt),a
10220     ld b,#128
```

```
10230      REPEAT
10240          ld a, (#Byt)
10250          and b
10260          IF z
10270              THEN
10280                  ld a, #48
10290              ELSE
10300                  ld a, #49
10310          FI
10320          call #ChPut
10330          srl b
10340          UNTIL z
10350          ret
10360 --
10370 :DoByt
10380     call #Procent
10390     ld a, (#Byt)
10400     call #Binair
10410     jp #NewLin
10420 --
10430 :DoWord
10440     call #Procent
10450     ld a, (#Word+1)
10460     call #Binair
10470     ld a, (#Word)
10480     call #Binair
10490     jp #NewLin
10500 --
10510 :Pauze
10520     ld a, #0
10530     call #GtTrig
10540     cp #$ff
10550     jpif z, #Pauze
10560     ret
10570 --
10580 :Init
10590     ld a, #0
10600     ld (#Byt), a
10610     REPEAT
10620         call #Pauze
10630         call #DoByt
10640         ld a, (#Byt)
10650         inc a
10660         ld (#Byt), a
10670     UNTIL z
10680     ld a, #0
10690     ld (#Word), a
10700     ld (#Word+1), a
10710     REPEAT
10720         call #Pauze
10730         call #DoWord
10740         ld a, (#Word)
```

```

10750      inc a
10760      ld (#Word),a
10770      ld a,(#Word+1)
10780      dec a
10790      ld (#Word+1),a
10800      UNTIL z
10810      ret
10820      --
10830 ENDCODE

```

.: C000 00 00 00 3E	"...>"	.: C048 C0 C3 08 C0	"...."
.: C004 25 C3 A2 00	"%..."	.: C04C 3E 00 CD D8	">..."
.: C008 3E 0A CD A2	.: C050 00 FE FF CA	"...."	
.: C00C 00 3E 0D C3	".>..."	.: C054 4C C0 C9 3E	"L..>"
.: C010 A2 00 32 00	".2.."	.: C058 00 32 00 C0	".2.."
.: C014 C0 06 80 3A	"...:"	.: C05C CD 4C C0 CD	".L.."
.: C018 00 C0 A0 C2	"...."	.: C060 2E C0 3A 00	"...:."
.: C01C 23 C0 3E 30	"#.>>"	.: C064 C0 3C 32 00	".<2.."
.: C020 C3 25 C0 3E	".%.>"	.: C068 C0 C2 5C C0	".\.."
.: C024 31 CD A2 00	"1..."	.: C06C 3E 00 32 01	">.2.."
.: C028 CB 38 C2 17	".8.."	.: C070 C0 32 02 C0	".2.."
.: C02C C0 C9 CD 03	"...."	.: C074 CD 4C C0 CD	".L.."
.: C030 C0 3A 00 C0	".:..."	.: C078 3A C0 3A 01	"...:."
.: C034 CD 12 C0 C3	"...."	.: C07C C0 3C 32 01	".<2.."
.: C038 08 C0 CD 03	"...."	.: C080 C0 3A 02 C0	"...:."
.: C03C C0 3A 02 C0	".:..."	.: C084 3D 32 02 C0	"=2.."
.: C040 CD 12 C0 3A	".:..."	.: C088 C2 74 C0 C9	".t.."
.: C044 01 C0 CD 12	"...."	.	.

.; C003 3E 25	LD A,\$25
.; C005 C3 A2 00	JP \$00A2
.; C008 3E 0A	LD A,\$0A
.; C00A CD A2 00	CALL \$00A2
.; C00D 3E 0D	LD A,\$0D
.; C00F C3 A2 00	JP \$00A2
.; C012 32 00 C0	LD (\$C000),A
.; C015 06 80	LD B,\$80
.; C017 3A 00 C0	LD A,(\$C000)
.; C01A A0	AND B
.; C01B C2 23 C0	JP NZ,\$C023
.; C01E 3E 30	LD A,\$30
.; C020 C3 25 C0	JP \$C025
.; C023 3E 31	LD A,\$31
.; C025 CD A2 00	CALL \$00A2
.; C028 CB 38	SRL B

```

.; C02A C2 17 CO      JP NZ,$C017
.; C02D C9      RET
.; C02E CD 03 CO      CALL $C003
.; C031 3A 00 CO      LD A,($C000)
.; C034 CD 12 CO      CALL $C012
.; C037 C3 08 CO      JP $C008
.; C03A CD 03 CO      CALL $C003
.; C03D 3A 02 CO      LD A,($C002)
.; C040 CD 12 CO      CALL $C012
.; C043 3A 01 CO      LD A,($C001)
.; C046 CD 12 CO      CALL $C012
.; C049 C3 08 CO      JP $C008
.; C04C 3E 00          LD A,$00
.; C04E CD D8 00      CALL $00D8
.; C051 FE FF          CP $FF
.; C053 CA 4C CO      JP Z,$C04C
.; C056 C9      RET
.; C057 3E 00          LD A,$00
.; C059 32 00 CO      LD ($C000),A
.; C05C CD 4C CO      CALL $C04C
.; C05F CD 2E CO      CALL $C02E
.; C062 3A 00 CO      LD A,($C000)
.; C065 3C          INC A
.; C066 32 00 CO      LD ($C000),A
.; C069 C2 5C CO      JP NZ,$C05C
.; C06C 3E 00          LD A,$00
.; C06E 32 01 CO      LD ($C001),A
.; C071 32 02 CO      LD ($C002),A
.; C074 CD 4C CO      CALL $C04C
.; C077 CD 3A CO      CALL $C03A
.; C07A 3A 01 CO      LD A,($C001)
.; C07D 3C          INC A
.; C07E 32 01 CO      LD ($C001),A
.; C081 3A 02 CO      LD A,($C002)
.; C084 3D          DEC A
.; C085 32 02 CO      LD ($C002),A
.; C088 C2 74 CO      JP NZ,$C074
.; C08B C9      RET
.

```

### Toelichting

Het wordt nu toch wat minder makkelijk, dus laten we de listing eens puntsgewijs van boven naar beneden bekijken. U zult dan zien dat de feitelijke routine niet zoveel in huis heeft.

PROCENT vertelt de gebruiker door middel van een %-teken dat het om een binair getal gaat.

NEWLIN geeft een linefeed en een carriage return (zie appendix voor exacte werking).

BINAIR redt de af te drukken accumulator. In het B-register wordt 128 (%10000000) geplaatst. Vervolgens gaan we dit B-register als masker voor het af te drukken byte gebruiken. Eerst een AND met %10000000, een gesette zero-flag heeft tot gevolg dat we een 1 afdrukken, anders een 0. Het B-register wordt naar rechts geschoven (%10000000 wordt %01000000...) en we herhalen de afdruk-test. Schuiven, afdrukken enz. Dit alles acht keer, waarna we terugkeren.

DOBYT drukt een byte af en verplaats de cursor naar een nieuwe regel (De CALL-return wordt verderop meegenomen).

DOWORD drukt een word af. Eerst het high-byte, dan het low-byte.

PAUZE kijkt of er op de spatiebalk wordt gedrukt. Is dat het geval dan wacht de routine totdat de spatiebalk weer wordt losgelaten.

INIT drukt ter demonstratie 255 bytes en 255 words af.

# 13 Hexadecimale sprongen

De getalsnotatie die in computerland het meest wordt gebruikt, is zonder twijfel de hexadecimale vorm. Niet alleen is deze het best leesbaar (toegegeven, het is even wennen...), het is ook de meest compacte manier om een getal te representeren. Waar een binaire byte-uitvoer acht maal naar de CHPUT zal grijpen, hoeft de hexadecimale variant slechts twee keer te scoren. Ziedaar de ruimte- en tijdwinst! Net als bij de voorgaande binaire routine roept u ook deze routine op twee manieren aan. Een byte-afdruk en een word-afdruk.

TIP: een prima routine voor een hexadecimale memory-dump! In combinatie met een invoermogelijkheid simuleert u de SpeedTyper.

```
10000 ORIGIN ($c000)
10010 --
10020 $00a2 CONSTANT ChPut
10030 $00d8 CONSTANT GtTrig
10040 --
10050 CODE
10060 --
10070 :Byt    .by $00
10080 :Word   .wo $0000
10090 --
10100 :HexCod .ch "0123456789abcdef"
10110 --
10120 :String
10130     ld a,#36
10140     jp #ChPut
10150 --
10160 :NewLin
10170     ld a,#10
10180     call #ChPut
10190     ld a,#13
10200     jp #ChPut
10210 --
10220 :MakNib
10230     ld hl,#HexCod
10240     ld c,a
10250     ld b,#0
10260     add hl,bc
```

```
10270     ld a,(hl)
10280     jp #ChPut
10290 --
10300 :MakByt
10310     push af
10320     srl a
10330     srl a
10340     srl a
10350     srl a
10360     call #MakNib
10370     pop af
10380     and #15
10390     jp #MakNib
10400 --
10410 :DoByt
10420     call #String
10430     ld a,(#Byt)
10440     call #MakByt
10450     jp #NewLin
10460 --
10470 :DoWord
10480     call #String
10490     ld a,(#Word+1)
10500     call #MakByt
10510     ld a,(#Word)
10520     call #MakByt
10530     jp #NewLin
10540 --
10550 :Pauze
10560     ld a,#0
10570     call #GtTrig
10580     cp #$ff
10590     jpif z,#Pauze
10600     ret
10610 --
10620 :Init
10630     ld a,#0
10640     ld (#Byt),a
10650     REPEAT
10660         call #Pauze
10670         call #DoByt
10680         ld a,(#Byt)
10690         inc a
10700         ld (#Byt),a
10710     UNTIL z
10720     ld a,#0
10730     ld (#Word),a
10740     ld (#Word+1),a
10750     REPEAT
10760         call #Pauze
10770         call #DoWord
```

```

10780      ld a,(#Word)
10790      inc a
10800      ld (#Word),a
10810      ld a,(#Word+1)
10820      dec a
10830      ld (#Word+1),a
10840      UNTIL z
10850      ret
10860  --
10870 ENDCODE

```

.: C000 00 00 00 30	"...0"	.: C050 C0 CD 2D C0	"...-."
.: C004 31 32 33 34	"1234"	.: C054 3A 01 C0 CD	"...-."
.: C008 35 36 37 38	"5678"	.: C058 2D C0 C3 18	"-..."
.: C00C 39 41 42 43	"9ABC"	.: C05C C0 3E 00 CD	".>.."
.: C010 44 45 46 3E	"DEF>"	.: C060 D8 00 FE FF	"...-."
.: C014 24 C3 A2 00	"\$..."	.: C064 CA 5D C0 C9	".].."
.: C018 3E 0A CD A2	.: C068 3E 00 32 00		
.: C01C 00 3E 0D C3	".>.."	.: C06C C0 CD 5D C0	".].."
.: C020 A2 00 21 03	".!.."	.: C070 CD 3F C0 3A	".?.:"
.: C024 C0 4F 06 00	".O.."	.: C074 00 C0 3C 32	".<2"
.: C028 09 7E C3 A2	"...."	.: C078 00 C0 C2 6D	"....m"
.: C02C 00 F5 CB 3F	".?.?"	.: C07C C0 3E 00 32	".>.2"
.: C030 CB 3F CB 3F	".?.?"	.: C080 01 C0 32 02	".2."
.: C034 CB 3F CD 22	".?.?"	.: C084 C0 CD 5D C0	".].."
.: C038 C0 F1 E6 0F	"...."	.: C088 CD 4B C0 3A	".K.:"
.: C03C C3 22 C0 CD	"...."	.: C08C 01 C0 3C 32	".<2"
.: C040 13 C0 3A 00	".:.."	.: C090 01 C0 3A 02	".?.."
.: C044 C0 CD 2D C0	".-.."	.: C094 C0 3D 32 02	".=2."
.: C048 C3 18 C0 CD	"...."	.: C098 C0 C2 85 C0	"...."
.: C04C 13 C0 3A 02	".:.."	.: C09C C9 00 FE OD	"...."

.; C013 3E 24	LD A,\$24
.; C015 C3 A2 00	JP \$00A2
.; C018 3E 0A	LD A,\$0A
.; C01A CD A2 00	CALL \$00A2
.; C01D 3E 0D	LD A,\$0D
.; C01F C3 A2 00	JP \$00A2
.; C022 21 03 C0	LD HL,\$C003
.; C025 4F	LD C,A
.; C026 06 00	LD B,\$00
.; C028 09	ADD HL,BC
.; C029 7E	LD A,(HL)
.; C02A C3 A2 00	JP \$00A2
.; C02D F5	PUSH AF
.; C02E CB 3F	SRL A

```

.; C030 CB 3F      SRL A
.; C032 CB 3F      SRL A
.; C034 CB 3F      SRL A
.; C036 CD 22 CO   CALL $C022
.; C039 F1         POP AF
.; C03A E6 0F       AND $0F
.; C03C C3 22 CO   JP $C022
.; C03F CD 13 CO   CALL $C013
.; C042 3A 00 CO   LD A,($C000)
.; C045 CD 2D CO   CALL $C02D
.; C048 C3 18 CO   JP $C018
.; C04B CD 13 CO   CALL $C013
.; C04E 3A 02 CO   LD A,($C002)
.; C051 CD 2D CO   CALL $C02D
.; C054 3A 01 CO   LD A,($C001)
.; C057 CD 2D CO   CALL $C02D
.; C05A C3 18 CO   JP $C018
.; C05D 3E 00       LD A,$00
.; C05F CD D8 00   CALL $00D8
.; C062 FE FF       CP $FF
.; C064 CA 5D CO   JP Z,$C05D
.; C067 C9         RET
.; C068 3E 00       LD A,$00
.; C06A 32 00 CO   LD ($C000),A
.; C06D CD 5D CO   CALL $C05D
.; C070 CD 3F CO   CALL $C03F
.; C073 3A 00 CO   LD A,($C000)
.; C076 3C         INC A
.; C077 32 00 CO   LD ($C000),A
.; C07A C2 6D CO   JP NZ,$C06D
.; C07D 3E 00       LD A,$00
.; C07F 32 01 CO   LD ($C001),A
.; C082 32 02 CO   LD ($C002),A
.; C085 CD 5D CO   CALL $C05D
.; C088 CD 4B CO   CALL $C04B
.; C08B 3A 01 CO   LD A,($C001)
.; C08E 3C         INC A
.; C08F 32 01 CO   LD ($C001),A
.; C092 3A 02 CO   LD A,($C002)
.; C095 3D         DEC A
.; C096 32 02 CO   LD ($C002),A
.; C099 C2 85 CO   JP NZ,$C085
.; C09C C9         RET
.

```

### Toelichting

Deze routine is nog iets ingewikkelder dan de vorige, aangezien we ieder byte in tweeën moeten splitsen (vier bits leveren een waarde tussen %0000 en %1111,

tussen \$0 en \$F, of tussen #0 en #15 op) en via een tabel moeten afdrukken. We bekijken de listing wederom puntsgewijs. Vooral MAKNIB en MAKBYT verdienen uw gewaardeerde aandacht.

STRING vertelt de gebruiker door middel van een \$-teken dat hij met een hexadecimaal getal te maken heeft.

NEWLIN geeft een linefeed en een carriage return.

MAKNIB plaatst het adres van de string HEXCOD in het HL-registerpaar. de inhoud van de accumulator (een waarde tussen 0 en 15) komt in het C-register en er komt een nul in het B-register. Vervolgens tellen we de registerparen HL en BC bij elkaar op, waarna we de HL-gadresseerde waarde afdrukken.

MAKBYT splitst het af te drukken byte in tweeën. Eerst het high-nybble. Daartoe schuiven we het byte vier maal naar rechts met als gevolg dat de zaak verschuift van 'aaaabbbb' naar '0000aaaa'. Met deze laatste waarde duiken we de MAKNIB-routine in.

Het low-nybble verkrijgen we met een AND-instructie. 'aaaabbbb' is na deze instructie omgetoverd in '0000bbbb'. Ook nu weer naar MAKNIB.

DOBYT drukt een byte af en verplaatst de cursor naar een nieuwe regel.

DOWORD drukt een word af. Eerst het high-byte, dan het low-byte.

# 14 Ongewoon gewoon!

Decimale getallen zien er zo bekend uit! Omdat 'gewone' computergebruikers zich niet willen pijnigen over het hoe en waarom van hexadecimale getallen, wordt van de programmeur verwacht dat hij of zij zich omschakelt. Decimale getallen op een computer is zoiets als eieren in een magnetron: het gaat wel, maar niet écht lekker! Geen nood, want een beetje programmeur serveert iedereen op maat, zelfs de leek. Evenals bij de twee voorgaande gevallen duiken we een subroutine in, maar er moet vooraf nog even wat gebeuren! U moet namelijk aangeven hoe 'breed' het getal gaat worden. 00000, 0000, 000, 00, of 0? Enfin, kijkt u maar naar het voorbeeld.

TIP: als u de werking van deze routine bijna letterlijk omdraait, kunt u in combinatie met de invoeroptie ook decimale getallen inlezen.

```
10000 ORIGIN ($c000)
10010 --
10020 $00a2 CONSTANT ChPut
10030 $00d8 CONSTANT GtTrig
10040 --
10050 CODE
10060 --
10070 :Byt    .by $00
10080 :Word   .wo $0000
10090 --
10100 :DecTab .wo 10000
10110           .wo 1000
10120           .wo 100
10130           .wo 10
10140           .wo 1
10150 --
10160 :Hekje
10170     ld a,#35
10180     jp #ChPut
10190 --
10200 :NewLin
10210     ld a,#10
10220     call #ChPut
```

```

10230    ld a,#13
10240    jp #ChPut
10250    --
10260    :Decimaal
10270    REPEAT
10280        push af
10290        ld a,(bc)
10300        ld e,a
10310        inc bc
10320        ld a,(bc)
10330        ld d,a
10340        inc bc
10350        ld a,#$ff
10360        or a
10370        push bc
10380    REPEAT
10390        ld b,h
10400        ld c,l
10410        sbc hl,de
10420        inc a
10430    UNTIL cs
10440        ld h,b
10450        ld l,c
10460        pop bc
10470        add a,#48
10480        call #ChPut
10490        pop af
10500        dec a
10510    UNTIL z
10520    ret
10530    --
10540    :DoByt
10550        call #Hekje
10560        ld h,#0
10570        ld a,(#Byt)
10580        ld l,a
10590        ld bc,#DecTab+4
10600        ld a,#3
10610        call #Decimaal
10620        jp #NewLin
10630    --
10640    :DoWord
10650        call #Hekje
10660        ld a,(#Word+1)
10670        ld h,a
10680        ld a,(#Word)
10690        ld l,a
10700        ld bc,#DecTab
10710        ld a,#5
10720        call #Decimaal
10730        jp #NewLin
10740    --

```

```

10750 :Pauze
10760     ld a,#0
10770     call #GtTrig
10780     cp #$ff
10790     jpif z,#Pauze
10800     ret
10810 --
10820 :Init
10830     ld a,#0
10840     ld (#Byt),a
10850     REPEAT
10860         call #Pauze
10870         call #DoByt
10880         ld a,(#Byt)
10890         inc a
10900         ld (#Byt),a
10910     UNTIL z
10920     ld a,#0
10930     ld (#Word),a
10940     ld (#Word+1),a
10950     REPEAT
10960         call #Pauze
10970         call #DoWord
10980         ld a,(#Word)
10990         inc a
11000         ld (#Word),a
11010         ld a,(#Word+1)
11020         dec a
11030         ld (#Word+1),a
11040     UNTIL z
11050     ret
11060 --
11070 ENDCODE

```

..: C000 00 00 00 10	"...."	..: C038 3D C2 1C C0	"=..."
..: C004 27 E8 03 64	".d"	..: C03C C9 CD 0D C0	"...."
..: C008 00 0A 00 01	"...."	..: C040 26 00 3A 00	"&::."
..: C00C 00 3E 23 C3	".>#."	..: C044 C0 6F 01 07	".o.."
..: C010 A2 00 3E 0A	"..>."	..: C048 C0 3E 03 CD	".>.."
..: C014 CD A2 00 3E	"...>"	..: C04C 1C C0 G3 12	"...."
..: C018 0D C3 A2 00	"...."	..: C050 C0 CD 0D C0	"...."
..: C01C F5 0A 5F 03	"...-."	..: C054 3A 02 C0 67	"...g"
..: C020 0A 57 03 3E	".W.>"	..: C058 3A 01 C0 6F	"...o"
..: C024 FF B7 C5 44	"...D"	..: C05C 01 03 C0 3E	"...>"
..: C028 4D ED 52 3C	"M.R<"	..: C060 05 CD 1C C0	"...."
..: C02C D2 27 C0 60	"..`"	..: C064 C3 12 C0 3E	"...>"
..: C030 69 C1 C6 30	"i..0"	..: C068 00 CD D8 00	"...."
..: C034 CD A2 00 F1	"...."	..: C06C FE FF CA 67	"...g"

```

. : C070 C0 C9 3E 00    "...>."    . : C08C 32 02 C0 CD "2...""
. : C074 32 00 C0 CD "2..."    . : C090 67 C0 CD 51 "g..Q"
. : C078 67 C0 CD 3D "g..="    . : C094 C0 3A 01 C0 ".::."
. : C07C C0 3A 00 C0 ".::."    . : C098 3C 32 01 C0 "<2.."
. : C080 3C 32 00 C0 "<2.."    . : C09C 3A 02 C0 3D ".::="
. : C084 C2 77 C0 3E ".w.>"    . : COA0 32 02 C0 C2 "2..."
. : C088 00 32 01 C0 ".2.."    . : COA4 8F C0 C9 49 "...I"

```

```

. ; C000 00          NOP
. ; C001 00          NOP
. ; C002 00          NOP
. ; C003 10 27        DJNZ @C02C
. ; C005 E8          RET PE
. ; C006 03          INC BC
. ; C007 64          LD H,H
. ; C008 00          NOP
. ; C009 0A          LD A,(BC)
. ; C00A 00          NOP
. ; C00B 01 00 3E      LD BC,$3E00
. ; C00E 23          INC HL
. ; C00F C3 A2 00      JP $00A2
. ; C012 3E 0A          LD A,$0A
. ; C014 CD A2 00      CALL $00A2
. ; C017 3E 0D          LD A,$0D
. ; C019 C3 A2 00      JP $00A2
. ; C01C F5          PUSH AF
. ; C01D 0A          LD A,(BC)
. ; C01E 5F          LD E,A
. ; C01F 03          INC BC
. ; C020 0A          LD A,(BC)
. ; C021 57          LD D,A
. ; C022 03          INC BC
. ; C023 3E FF          LD A,$FF
. ; C025 B7          OR A
. ; C026 C5          PUSH BC
. ; C027 44          LD B,H
. ; C028 4D          LD C,L
. ; C029 ED 52          SBC HL,DE
. ; C02B 3C          INC A
. ; C02C D2 27 C0      JP NC,$C027
. ; C02F 60          LD H,B
. ; C030 69          LD L,C
. ; C031 C1          POP BC
. ; C032 C6 30          ADD A,$30
. ; C034 CD A2 00      CALL $00A2
. ; C037 F1          POP AF
. ; C038 3D          DEC A
. ; C039 C2 1C C0      JP NZ,$C01C
. ; C03C C9          RET

```

```

.; C03D CD 0D CO      CALL $C00D
.; C040 26 00          LD H,$00
.; C042 3A 00 CO      LD A,($C000)
.; C045 6F             LD L,A
.; C046 01 07 CO      LD BC,$C007
.; C049 3E 03          LD A,$03
.; C04B CD 1C CO      CALL $C01C
.; C04E C3 12 CO      JP $C012
.; C051 CD 0D CO      CALL $C00D
.; C054 3A 02 CO      LD A,($C002)
.; C057 67             LD H,A
.; C058 3A 01 CO      LD A,($C001)
.; C05B 6F             LD L,A
.; C05C 01 03 CO      LD BC,$C003
.; C05F 3E 05          LD A,$05
.; C061 CD 1C CO      CALL $C01C
.; C064 C3 12 CO      JP $C012
.; C067 3E 00          LD A,$00
.; C069 CD D8 00      CALL $00D8
.; C06C FE FF          CP $FF
.; C06E CA 67 CO      JP Z,$C067
.; C071 C9             RET
.; C072 3E 00          LD A,$00
.; C074 32 00 CO      LD ($C000),A
.; C077 CD 67 CO      CALL $C067
.; C07A CD 3D CO      CALL $C03D
.; C07D 3A 00 CO      LD A,($C000)
.; C080 3C             INC A
.; C081 32 00 CO      LD ($C000),A
.; C084 C2 77 CO      JP NZ,$C077
.; C087 3E 00          LD A,$00
.; C089 32 01 CO      LD ($C001),A
.; C08C 32 02 CO      LD ($C002),A
.; C08F CD 67 CO      CALL $C067
.; C092 CD 51 CO      CALL $C051
.; C095 3A 01 CO      LD A,($C001)
.; C098 3C             INC A
.; C099 32 01 CO      LD ($C001),A
.; C09C 3A 02 CO      LD A,($C002)
.; C09F 3D             DEC A
.; COA0 32 02 CO      LD ($C002),A
.; COA3 C2 8F CO      JP NZ,$C08F
.; COA6 C9             RET
.
```

### Toelichting

Decimaal afdrukken werkt, vanuit de computer gezien, nogal ongelukkig zoals u inmiddels heeft vernomen. We moeten met machten van tien rekenen, met als

gevolg dat we in een minder gecontroleerde lus komen. In dit geval kost het dan ook MEER tijd om een GROTER getal af te drukken. Waarschijnlijk is dit voor u geen enkel probleem, maar in kritische gevallen is het toch erg onhandig. We bekijken de listing weer eens puntsgewijs. DECIMAAL, DOBYT en DOWORD verdienen uw aandacht! Let trouwens ook op het werken met die DECTAB-word-tabel, een dergelijke adressering heeft u ook nodig als u externe invoer wilt omzetten naar een later te verwerken waarde.

HEKJE vertelt ons door middel van een # -teken dat we met een decimaal getal te maken hebben.

NEWLIN geeft een linefeed en een carriage return.

DECIMAAL zet een waarde (byte of word) om in een leesbare output. Eerst halen we via via een DECTAB-waarde in het DE-registerpaar. Met die OR A zorgen we voor een juiste SBC-ingang (Laat die OR A maar eens weg! Ziet u wel...) Nu verminderen we onze waarde net zolang met de inhoud van het DE-registerpaar totdat het fout dreigt te gaan. In dat geval hebben we een nog net juiste waarde in het BC-registerpaar staan en die bewaren we dan ook voor het volgende DECTAB-getal.

Om te zorgen dat we een cijfer op het scherm krijgen, tellen we 48 bij de accumulator op (zie appendix B voor het hoe en waarom.) en drukken we die waarde af.

Dit alles opnieuw, totdat de teller op nul staat. Dit heeft als gevolg dat we weer terugkeren.

DOBYT drukt een byte af via het HL-registerpaar en verplaatst de cursor naar nieuwe regel. Let op het instellen van de DECIMAAL-routine (Lengte is drie: 100, 10 en 1).

DOWORD drukt een word af via het HL-registerpaar en verplaatst de cursor naar een nieuwe regel. Let ook hier op het instellen van de DECIMAAL-routine (Lengte is vijf: 10000, 1000, 100, 10 en 1).

PAUZE kijkt of er op de spatiebalk wordt gedrukt. Als dat het geval is, wacht de routine totdat de spatiebalk wordt losgelaten.

INIT drukt ter demonstratie 255 bytes en 255 words af.

## 15 Tot de volgende keer

De meeste programma's die u op uw computer maakt, wilt u de volgende keer weer gebruiken en dan nog het liefst zonder alle poespas van de vorige keer. Een nadeel van de computer is in zo'n geval de aan/uit-knop, want als u het apparaat uitzet, bent u meteen alle ingevoerde gegevens kwijt. Laat u het ding aanstaan dan wordt de stroomrekening als het ware gelanceerd.

Wat moet u doen? Gewoon alles op een cassettebandje proppen. Saven en loaden dus!

Welnu, dat loaden en saven is een heel gedoe met een MSX-computer. Een echt goed cassetteprotocol ontbreekt en daarom moet u dat zelf in elkaar zetten. Denkt u even mee, wat hebben we nodig om te kunnen saven? Een bestandsnaam, een startadres en een eindadres (aangenomen dat u voor de recorder en het bandje zorgt).

Om te beginnen starten we de band, we plaatsen een vast aantal keren een bepaalde waarde op de tape zodat de computer tijdens het loaden het begin van een file kan vinden. Vervolgens schrijven we de bestandsnaam weg, hierna (ook al weer voor het loaden) het beginadres en het aantal bytes dat ge-saved worden. En ten slotte zet u de inhoud van het geheugengebied op de band.

Het loaden vervolgens: starten, het begin van een file opzoeken (een vast aantal keren diezelfde waarde), de bestandsnaam inlezen en vergelijken met de door de gebruiker opgegeven bestandsnaam. (Zo nee, doorzoeken. Zo ja, inlezen!) Vervolgens het startadres opzoeken en opslaan, het aantal te loaden bytes inlezen en deze ten slotte loaden en in het geheugen zetten.

TIP: tijdens het loaden is het zeer eenvoudig om vanaf een ander startadres te schrijven. Bedenkt u zelf maar hoe!

```

10000 ORIGIN ($c000)
10010 --
10020 $00a2 CONSTANT ChPut
10030 $00d8 CONSTANT GtTrig
10040 $00e1 CONSTANT TapIon
10050 $00e4 CONSTANT TapIn
10060 $00e7 CONSTANT TapIof
10070 $00ea CONSTANT TapOon
10080 $00ed CONSTANT TapOut
10090 $00fo CONSTANT TapOof
10100 --
10110 CODE
10120 --
10130 :Stapel .wo $0000
10140 :Begin .wo $e000
10150 :Eind .wo $e400
10160 --
10170 :Name .ch "testje"
10180 .by $0a,$0d,$00
10190 :Deze .ch "....."
10200 .by $0a,$0d,$00
10210 --
10220 :RecPly .ch "press record and play on tape"
10230 .by $0a,$0d,$00
10240 :RewPly .ch "rewind - press play on tape"
10250 .by $0a,$0d,$00
10260 :IOTxt .ch "io-error!"
10270 .by $0a,$0d,$00
10280 :SavTxt .ch "saving: "
10290 .by $00
10300 :ZkTxt .ch "searching: "
10310 .by $00
10320 :VndTxt .ch "found: "
10330 .by $00
10340 :LdTxt .ch "loading..."
10350 .by $0a,$0d,$00
10360 :Ready .ch "ready!"
10370 .by $0a,$0a,$0d,$00
10380 --
10390 :Print
10400   ld a,(hl)
10410   or a
10420   retif z
10430   call #ChPut
10440   inc hl
10450   jp #Print
10460 --
10470 :Error
10480   ld sp,(#Stapel)
10490   call #TapOof
10500   ld hl,#IOTxt
10510   jp #Print

```

```
10520 --
10530 :Won
10540     push bc
10550     push de
10560     push hl
10570     ld a,#0
10580     call #TapOn
10590     jpif cs,#Error
10600 :Pull
10610     pop hl
10620     pop de
10630     pop bc
10640     ret
10650 --
10660 :WChr
10670     push bc
10680     push de
10690     push hl
10700     call #TapOut
10710     jpif cs,#Error
10720     jp #Pull
10730 --
10740 :RChr
10750     push bc
10760     push de
10770     push hl
10780     call #TapIn
10790     jpif cs,#Error
10800     jp #Pull
10810 --
10820 :Ron
10830     push bc
10840     push de
10850     push hl
10860     call #TapIon
10870     jpif cs,#Error
10880     jp #Pull
10890 --
10900 :Save
10910     ld (#Stapel),sp
10920     ld hl,#SavTxt
10930     call #Print
10940     ld hl,#Name
10950     call #Print
10960     call #Won
10970     ld b,#16
10980 :Save0
10990     ld a,$$80
11000     call #WChr
11010     djnz #Save0
11020     ld hl,#Name
11030     ld b,#6
```

```
11040 :Save1
11050     ld a,(hl)
11060     call #WChr
11070     inc hl
11080     djnz #Save1
11090     call #TapOof
11100     ld hl,(#Begin)
11110     push hl
11120     ld de,(#Eind)
11130     ex hl,de
11140     or a
11150     sbc hl,de
11160     push hl
11170     ld (#Eind),hl
11180     call #Won
11190     ld b,#2
11200 :Save2
11210     pop hl
11220     ld a,l
11230     call #WChr
11240     ld a,h
11250     call #WChr
11260     djnz #Save2
11270     ld de,(#Begin)
11280     ld hl,(#Eind)
11290 :Save3
11300     ld a,(de)
11310     call #WChr
11320     inc de
11330     dec hl
11340     ld a,h
11350     or l
11360     jpif nz,#Save3
11370     call #TapOof
11380     ld hl,#Ready
11390     jp #Print
11400 --
11410 :Load
11420     ld (#Stapel),sp
11430 :Load0
11440     ld hl,#ZkTxt
11450     call #Print
11460     ld hl,#Name
11470     call #Print
11480     call #Ron
11490 :Load1
11500     ld b,#16
11510 :Load2
11520     call #RChr
11530     cp #$80
11540     jpif nz,#Load1
11550     djnz #Load2
```

```
11560      ld hl,#Deze
11570      ld b,#6
11580 :Load3
11590      call #Rchr
11600      ld (hl),a
11610      inc hl
11620      djnz #Load3
11630      ld (hl),#0
11640      call #TapIof
11650      ld hl,#VndTxt
11660      ld a,#$0a
11670      ld (#Deze+6),a
11680      call #Print
11690      ld hl,#Deze
11700      call #Print
11710      ld de,#Name
11720      ld hl,#Deze
11730      ld b,#6
11740 :Load4
11750      ld a,(de)
11760      cp (hl)
11770      jpif nz,#Load0
11780      inc de
11790      inc hl
11800      djnz #Load4
11810      ld hl,#LdTxt
11820      call #Print
11830      call #Ron
11840      ld b,#2
11850 :Load5
11860      call #RChr
11870      ld l,a
11880      call #RChr
11890      ld h,a
11900      push hl
11910      djnz #Load5
11920      pop de
11930      ld (#Begin),de
11940      pop hl
11950 :Load6
11960      call #RChr
11970      ld (de),a
11980      inc de
11990      dec hl
12000      ld a,h
12010      or l
12020      jpif nz,#Load6
12030      call #TapIof
12040      ld hl,#Ready
12050      jp #Print
12060 --
12070 :Wacht
```

```

12080    ld a,#0
12090    call #GtTrig
12100    cp #$ff
12110    jpif nz,#Wacht
12120    ret
12130    --
12140    :Init
12150    ld hl,#RecPly
12160    call #Print
12170    call #Wacht
12180    call #Save
12190    ld hl,#RewPly
12200    call #Print
12210    call #Wacht
12220    call #Load
12230    jp #Wacht
12240    --
12250 ENDCODE

```

..: C000 00 00 00 E0	"...."	..: C074 3A 20 00 46	" : .F"
..: C004 00 E4 54 45	".TE"	..: C078 4F 55 4E 44	"OUND"
..: C008 53 54 4A 45	"STJE"	..: C07C 3A 20 00 4C	" : .L"
..: C00C 0A 0D 00 2E	"...."	..: C080 4F 41 44 49	"OADI"
..: C010 2E 2E 2E 2E	"...."	..: C084 4E 47 2E 2E	"NG.."
..: C014 2E 0A 0D 00	"...."	..: C088 2E 0A 0D 00	"...."
..: C018 50 52 45 53	"PRES"	..: C08C 52 45 41 44	"READ"
..: C01C 53 20 52 45	"S RE"	..: C090 59 21 0A 0A	"Y!.."
..: C020 43 4F 52 44	"CORD"	..: C094 0D 00 7E B7	"...."
..: C024 20 41 4E 44	" AND"	..: C098 C8 CD A2 00	"...."
..: C028 20 50 4C 41	" PLA"	..: C09C 23 C3 96 CO	"#...."
..: C02C 59 20 4F 4E	"Y ON"	..: COA0 ED 7B 00 CO	".{.."
..: C030 20 54 41 50	" TAP"	..: COA4 CD F0 00 21	"....!"
..: C034 45 0A 0D 00	"E..."	..: COA8 56 CO C3 96	"V..."
..: C038 52 45 57 49	"REWI"	..: COAC CO C5 D5 E5	"...."
..: C03C 4E 44 20 2D	"ND -"	..: COBO 3E 00 CD EA	
..: C040 20 50 52 45	" PRE"	..: COB4 00 DA A0 CO	"...."
..: C044 53 53 20 50	"SS P"	..: COB8 E1 D1 C1 C9	"...."
..: C048 4C 41 59 20	"LAY "	..: COBC C5 D5 E5 CD	"...."
..: C04C 4F 4E 20 54	"ON T"	..: COCO ED 00 DA A0	"...."
..: C050 41 50 45 0A	"APE."	..: COC4 CO C3 B8 CO	"...."
..: C054 0D 00 49 4F	".IO"	..: COC8 C5 D5 E5 CD	"...."
..: C058 2D 45 52 52	"-ERR"	..: COCC E4 00 DA A0	"...."
..: C05C 4F 52 21 0A	"OR!.."	..: COD0 CO C3 B8 CO	"...."
..: C060 0D 00 53 41	". SA"	..: COD4 C5 D5 E5 CD	"...."
..: C064 56 49 4E 47	"VING"	..: COD8 E1 00 DA A0	"...."
..: C068 3A 20 00 53	" : .S"	..: CODC CO C3 B8 CO	"...."
..: C06C 45 41 52 43	"EARC"	..: COEO ED 73 00 CO	".s..."
..: C070 48 49 4E 47	"HING"	..: COE4 21 62 CO CD	"!b..."

.. COE8 96 CO 21 06	"...!"	.. C16C CO 77 23 10	".w#."
.. COEC CO CD 96 CO	"...."	.. C170 F9 36 00 CD	".6.."
.. COFO CD AD CO 06	"...."	.. C174 E7 00 21 77	".!w."
.. COF4 10 3E 80 CD	".>.."	.. C178 CO 3E OA 32	".>.2"
.. COF8 BC CO 10 F9	"...."	.. C17C 15 CO CD 96	"...."
.. COFC 21 06 CO 06	"!...."	.. C180 CO 21 OF CO	"!...."
.. C100 06 7E CD BC	"...."	.. C184 CD 96 CO 11	"...."
.. C104 CO 23 10 F9	".#.."	.. C188 06 CO 21 OF	"!...."
.. C108 CD FO 00 2A	"....*"	.. C18C CO 06 06 1A	"...."
.. C10C 02 CO E5 ED	"...."	.. C190 BE C2 4A C1	".J."
.. C110 5B 04 CO EB	"[...]"	.. C194 13 23 10 F7	".#.."
.. C114 B7 ED 52 E5	".R."	.. C198 21 7F CO CD	"!...."
.. C118 22 04 CO CD	"...."	.. C19C 96 CO CD D4	"...."
.. C11C AD CO 06 02	"...."	.. C1A0 CO 06 02 CD	"...."
.. C120 E1 7D CD BC	".}.."	.. C1A4 C8 CO 6F CD	".o."
.. C124 CO 7C CD BC	".!.."	.. C1A8 C8 CO 67 E5	".g."
.. C128 CO 10 F5 ED	"...."	.. C1AC 10 F5 D1 ED	"...."
.. C12C 5B 02 CO 2A	"[...*]"	.. C1B0 53 02 CO E1	"S..."
.. C130 04 CO 1A CD	"...."	.. C1B4 CD C8 CO 12	"...."
.. C134 BC CO 13 2B	"....+"	.. C1B8 13 2B 7C B5	".+!.."
.. C138 7C B5 C2 32	".!2"	.. C1BC C2 B4 C1 CD	"...."
.. C13C C1 CD FO 00	"...."	.. C1C0 E7 00 21 8C	".!..."
.. C140 21 8C CO C3	"!...."	.. C1C4 CO C3 96 CO	"...."
.. C144 96 CO ED 73	"...s"	.. C1C8 3E 00 CD D8	
.. C148 00 CO 21 6B	".!k"	.. C1CC 00 FE FF C2	"...."
.. C14C CO CD 96 CO	"...."	.. C1D0 C8 C1 C9 21	".!..."
.. C150 21 06 CO CD	"!...."	.. C1D4 18 CO CD 96	"...."
.. C154 96 CO CD D4	"...."	.. C1D8 CO CD C8 C1	"...."
.. C158 CO 06 10 CD	"...."	.. C1DC CD EO CO 21	".!..."
.. C15C C8 CO FE 80	"...."	.. C1E0 38 CO CD 96	"8..."
.. C160 C2 59 C1 10	".Y.."	.. C1E4 CO CD C8 C1	"...."
.. C164 F6 21 OF CO	".!.."	.. C1E8 CD 46 C1 C3	".F.."
.. C168 06 06 CD C8	"...."	.. C1EC C8 C1 C8 FC	"...."

.. C096 7E	LD A, (HL)
.. C097 B7	OR A
.. C098 C8	RET Z
.. C099 CD A2 00	CALL \$00A2
.. C09C 23	INC HL
.. C09D C3 96 CO	JP \$C096
.. COA0 ED 7B 00 CO	LD SP, (\$C000)
.. COA4 CD F0 00	CALL \$00F0
.. COA7 21 56 CO	LD HL, \$C056
.. COAA C3 96 CO	JP \$C096
.. COAD C5	PUSH BC
.. COAE D5	PUSH DE
.. COAF E5	PUSH HL
.. COBO 3E 00	LD A, \$00
.. COB2 CD EA 00	CALL \$00EA

```

.; COB5 DA A0 C0      JP C,$COA0
.; COB8 E1            POP HL
.; COB9 D1            POP DE
.; COBA C1            POP BC
.; COBB C9            RET
.; COBC C5            PUSH BC
.; COBD D5            PUSH DE
.; COBE E5            PUSH HL
.; COBF CD ED 00      CALL $00ED
.; COC2 DA A0 C0      JP C,$COA0
.; COC5 C3 B8 C0      JP $COB8
.; COC8 C5            PUSH BC
.; COC9 D5            PUSH DE
.; COCA E5            PUSH HL
.; COCB CD E4 00      CALL $00E4
.; COCE DA A0 C0      JP C,$COA0
.; COD1 C3 B8 C0      JP $COB8
.; COD4 C5            PUSH BC
.; COD5 D5            PUSH DE
.; COD6 E5            PUSH HL
.; COD7 CD E1 00      CALL $00E1
.; CODA DA A0 C0      JP C,$COA0
.; CODD C3 B8 C0      JP $COB8
.; COEO ED 73 00 C0   LD ($C000),SP
.; COE4 21 62 C0      LD HL,$C062
.; COE7 CD 96 C0      CALL $C096
.; COEA 21 06 C0      LD HL,$C006
.; COED CD 96 C0      CALL $C096
.; COFO CD AD C0      CALL $COAD
.; COF3 06 10          LD B,$10
.; COF5 3E 80          LD A,$80
.; COF7 CD BC C0      CALL $COBC
.; COFA 10 F9          DJNZ @COF5
.; COFC 21 06 C0      LD HL,$C006
.; COFF 06 06          LD B,$06
.; C101 7E              LD A,(HL)
.; C102 CD BC C0      CALL $COBC
.; C105 23              INC HL
.; C106 10 F9          DJNZ @C101
.; C108 CD F0 00      CALL $00F0
.; C10B 2A 02 C0      LD HL,($C002)
.; C10E E5              PUSH HL
.; C10F ED 5B 04 C0   LD DE,($C004)
.; C113 EB              EX DE,HL
.; C114 B7              OR A
.; C115 ED 52          SBC HL,DE
.; C117 E5              PUSH HL
.; C118 22 04 C0      LD ($C004),HL
.; C11B CD AD C0      CALL $COAD
.; C11E 06 02          LD B,$02
.; C120 E1              POP HL

```

```
; C121 7D LD A,L
; C122 CD BC CO CALL $C0BC
; C125 7C LD A,H
; C126 CD BC CO CALL $C0BC
; C129 10 F5 DJNZ @C120
; C12B ED 5B 02 CO LD DE,($C002)
; C12F 2A 04 CO LD HL,($C004)
; C132 1A LD A,(DE)
; C133 CD BC CO CALL $C0BC
; C136 13 INC DE
; C137 2B DEC HL
; C138 7C LD A,H
; C139 B5 OR L
; C13A C2 32 C1 JP NZ,$C132
; C13D CD F0 00 CALL $00F0
; C140 21 8C CO LD HL,$C08C
; C143 C3 96 CO JP $C096
; C146 ED 73 00 CO LD ($C000),SP
; C14A 21 6B CO LD HL,$C06B
; C14D CD 96 CO CALL $C096
; C150 21 06 CO LD HL,$C006
; C153 CD 96 CO CALL $C096
; C156 CD D4 CO CALL $C0D4
; C159 06 10 LD B,$10
; C15B CD C8 CO CALL $C0C8
; C15E FE 80 CP $80
; C160 C2 59 C1 JP NZ,$C159
; C163 10 F6 DJNZ @C15B
; C165 21 0F CO LD HL,$C00F
; C168 06 06 LD B,$06
; C16A CD C8 CO CALL $C0C8
; C16D 77 LD (HL),A
; C16E 23 INC HL
; C16F 10 F9 DJNZ @C16A
; C171 36 00 LD (HL),$00
; C173 CD E7 00 CALL $00E7
; C176 21 77 CO LD HL,$C077
; C179 3E 0A LD A,$0A
; C17B 32 15 CO LD ($C015),A
; C17E CD 96 CO CALL $C096
; C181 21 0F CO LD HL,$C00F
; C184 CD 96 CO CALL $C096
; C187 11 06 CO LD DE,$C006
; C18A 21 0F CO LD HL,$C00F
; C18D 06 06 LD B,$06
; C18F 1A LD A,(DE)
; C190 BE CP (HL)
; C191 C2 4A C1 JP NZ,$C14A
; C194 13 INC DE
; C195 23 INC HL
; C196 10 F7 DJNZ @C18F
```

```

.; C198 21 7F C0      LD HL,$C07F
.; C19B CD 96 C0      CALL $C096
.; C19E CD D4 C0      CALL $C0D4
.; C1A1 06 02          LD B,$02
.; C1A3 CD C8 C0      CALL $C0C8
.; C1A6 6F             LD L,A
.; C1A7 CD C8 C0      CALL $C0C8
.; C1AA 67             LD H,A
.; C1AB E5             PUSH HL
.; C1AC 10 F5          DJNZ @C1A3
.; C1AE D1             POP DE
.; C1AF ED 53 02 C0    LD ($C002),DE
.; C1B3 E1             POP HL
.; C1B4 CD C8 C0      CALL $C0C8
.; C1B7 12             LD (DE),A
.; C1B8 13             INC DE
.; C1B9 2B             DEC HL
.; C1BA 7C             LD A,H
.; C1BB B5             OR L
.; C1BC C2 B4 C1      JP NZ,$C1B4
.; C1BF CD E7 00      CALL $00E7
.; C1C2 21 8C C0      LD HL,$C08C
.; C1C5 C3 96 C0      JP $C096
.; C1C8 3E 00          LD A,$00
.; C1CA CD D8 00      CALL $00D8
.; C1CD FE FF          CP $FF
.; C1CF C2 C8 C1      JP NZ,$C1C8
.; C1D2 C9             RET
.; C1D3 21 18 C0      LD HL,$C018
.; C1D6 CD 96 C0      CALL $C096
.; C1D9 CD C8 C1      CALL $C1C8
.; C1DC CD E0 C0      CALL $COEO
.; C1DF 21 38 C0      LD HL,$C038
.; C1E2 CD 96 C0      CALL $C096
.; C1E5 CD C8 C1      CALL $C1C8
.; C1E8 CD 46 C1      CALL $C146
.; C1EB C3 C8 C1      JP $C1C8
.

```

### Toelichting

Het werken met de cassettereorder houdt veel meer in dan het starten van de motor. Denkt u maar eens aan al die meldingen, gaat het goed, fout of goed fout? Er zijn van die mensen die onderscheid kunnen maken tussen de diverse bliep-bliep-geluiden. Een handige eigenschap, maar het is natuurlijk veel handiger als u weet wat er precies met het bandje gebeurt. Het zou vervelend zijn als u 'ZZ-Top'

in de recorder plaatste, in de waan dat u een hardcopy-routine probeerde binnen te halen...

Voor onze tape-routines denken we aan slimme terugkoppelingen en een eigen opslagmethode. In de listing ziet u al diverse systeemboodschappen en we gaan ze allemaal serieus gebruiken.

Cassetteroutines op een MSX-computer zijn niet bepaald eenvoudig te noemen, maar als u uiteindelijk over een geolied systeem beschikt, is fouten maken verder onmogelijk. We gaan het weer label voor label bekijken. (Houdt u de BIOS-appendix er even bij, dank u!)

PRINT is u inmiddels welbekend.

ERROR wordt aangeroepen als er een I/O-error werd gesignaleerd. Het kan om een CTRL/STOP-combinatie gaan, maar ook is het mogelijk dat de cassetterecoder 'hangt'. We halen (om een juiste terugkeer te garanderen) de oorspronkelijke stapelwijzer voor de dag en we keren met een error-melding weer terug in ons hoofdprogramma.

WON zet de belangrijkste registerparen op de stapel. We plaatsen een nul in de accumulator om een korte header te garanderen en we duiken BIOS-TAPOON in. Als er een error mocht optreden, verwerken we deze. Het ontbreken van een error heeft tot gevolg dat we de registerparen weer van de stapel halen om vervolgens terug te keren.

WCHR schrijft een byte naar de cassette via de BIOS-TAPOUT, en houdt rekening met een eventueel opgetreden error (Met stapelbezoek).

RCHR leest een byte vanaf de cassette via de BIOS-TAPIN. Ook hier denken we weer om die error (dus met eventueel stapelbezoek).

RON start de cassettemotor en garandeert een aanloopje richting header via de BIOS-TAPION. Opgelet voor die error, ook hier wederom met stapelbezoek.

SAVE zet om te beginnen de stack pointer opzij voor de eventueel aan te roepen error-routine. Vervolgens drukken we (gebruikersvriendelijk als altijd) een boodschap af.

De motor wordt gestart. Eerst zetten we 16 (deze waarde kunt u veranderen, maar dan ook bij de LOAD-routine!) maal dezelfde waarde (nooit \$00 of \$FF!) op de band, om later het begin van een programma te herkennen.

Nu de bestandsnaam (lengte 6 tekens) naar de cassetteband. Hierna zetten we de motor voor de zekerheid even uit (kritiek punt bij het weer binnenthalen van een file!) en we berekenen het begin-adres en het aantal te plaatsen bytes. Deze waarden gaan met een inmiddels weer gestarte motor naar de tape.

Eenmaal alle administratie achter de rug? Dan gaat het eigenlijke geheugengebied naar de band, waarna we de boel afsluiten met een READY-melding.

LOAD zet eveneens de stapelwijzer apart. Nu drukken we de gezochte bestandsnaam af waarna we op zoek gaan.

Motor aangezet, vervolgens op jacht naar die zestien speciale waarden... Een afwijking en we beginnen weer opnieuw! Klopt het patroon daarentegen dan moet er een bestandsnaam volgen.

We hebben dus iets gevonden, we kijken wat het is, we maken melding en we vergelijken. Als het patroon opnieuw afwijkt, springen we weer terug. Klopt het dan beginnen we met het werkelijke laden.

De save-routine heeft twee waarden op de band gezet: het startadres en de hoeveelheid binnen te laden materiaal. We lezen beide waarden in. (Op dit moment kunt u het startadres vervangen door een nieuw startadres, maar opgepast!)

Het hart van de routine is eindelijk bereikt. Nu begint het feitelijke laden: een waarde halen, schrijven, nog eentje enz. tot het einde van het bestand is bereikt. Hierna meldt de computer dit aan de gebruiker door middel van een READY-boodschap.

De routines:

WACHT is een oude bekende.

INIT verzorgt ook een stukje tape-OS met behulp van een tweetal mededelingen, en springt naar de save- en load-routines.

TIP: een verify-routine lijkt verschrikkelijk veel op de load-routine. Alleen moet u nu vergelijken in plaats van schrijven. Op het moment dat een waarde afwijkt, springt u naar een speciale VERIFY ERROR-routine.

## **Nog even dit**

Met de acht omschreven routines kunt u nu iedere willekeurige invoer en/of uitvoer controleren. Het spreekt voor zich dat er nog iets dient te gebeuren met al die data. Misschien iets in de richting van tekstverwerken, grafisch printen, of u zoekt het in de recreatieve sfeer? Het maakt allemaal niet uit, als alle gebruikte subroutines maar werken, want daar draait het immers om.

Het kan intussen geen kwaad om de routines in te typen en uit te testen. Als u een aantal routines tegelijk wilt gebruiken, kunt u ze met de A-instructie van HERBY-MON op een ander adres assembleren, u kunt ze verplaatsen met de T-instructie, of binnen laden op een nieuw adres met de L-instructie.

# 16 MSX à la carte...

In de volgende hoofdstukken gaan we een aantal MSX-mogelijkheden eens op een alternatieve manier bekijken. Videomogelijkheden wel te verstaan!

## 16.1 Achter de schermen van een MSX-computer

Zoals u weet, heeft een MSX-computer (MSX1) vier verschillende screen modes (schermconfiguraties: de mogelijkheden van een specifiek videoscherm) die alle vier over specifieke eigenschappen beschikken.

- De  $40 \times 24$ -mode (0) is eigenlijk alleen geschikt voor brede tekst-editing.
- De  $32 \times 24$ -tekst-mode (1) is een smallere versie van de eerste, maar ook sprites zijn nu toegestaan.
- De hires mode (2) werkt nogal bedriegelijk. We beschikken namelijk niet over een echt hires-scherm maar meer over een drietal programmeerbare karaktersets. Dit biedt echter legio mogelijkheden voor onder andere flitsende spelprogrammering.
- De multi color mode (3) ten slotte lijkt een zeer ingewikkelde toestand, maar ook hierin kunnen we in machinetaal leuke grappen uithalen.

Het toverwoord sprites is reeds gevallen. Wat zijn sprites? Sprites zijn speciale, onafhankelijk te manipuleren stukjes scherm... Met andere woorden: handige poppetjes.

De MSX voorziet in maar liefst 32 van dergelijke karakter-achtige wezens die, onafhankelijk van wat dan ook, overal op het scherm te voorschijn zijn te toveren. Zonder veel moeite heeft u nu een mogelijkheid gekregen tot verfraaiing van al uw programma's. Een spectaculaire 'titelpagina', tijdaanduiding, invaders enz. Ook op deze sprites komen we nog terug.

## **16.2 De videochip**

De videochip of VDP-processor is een aparte processor die een speciaal stuk programmeerbaar video-RAM beheert. Deze processor beschikt over een speciaal statusregister en acht werkregisters (zie appendix G). Het VDP-statusregister informeert u over de toestand op dat moment. De acht werkregisters zijn alle te manipuleren met behulp van een speciale BIOS-routine, WRTVDP geheten. Een klein probleem: u kunt deze VDP-registers alleen maar beschrijven (write only). Gelukkig zijn er in het speciale BIOS-werkgebied nog een aantal schaduwregisters aanwezig: RG0SAV, RG1SAV,...en RG7SAV.

Nu moet u ook nog met het videogeheugen (16K groot) kunnen communiceren. Daarvoor zijn er een RDVRM- of 'lees videogeheugen'-commando en een WRTVRM- of 'schrijf videogeheugen'-commando aanwezig. In appendix G leest u hoe u deze routines op de juiste wijze aanspreekt.

Dit drietal (WRTVDP, RDVRM, WRTVRM) is in principe het enige pakket routines dat u nodig hebt om de hele videochip te besturen, ware het niet dat het MSX-systeem nog een aantal extra routines in huis heeft. Het is niet per se noodzakelijk om van deze routines gebruikte te maken, maar ze zitten toch al in het BIOS verwerkt. Dus waarom dan ook niet?

Een paar voorbeelden van deze routines: DISSCR om het scherm uit te zetten, ENASCR zet het scherm weer aan, FILVRM vult een stukvideogeheugen met een door u bepaalde waarde, CLRSPPR wist alle sprites enz.

Alle routines worden wederom in de appendix besproken.

## **16.3 Wat gaan we doen?**

We gaan de vijf bovenstaande principes behandelen maar zoals gezegd op een enigszins afwijkende wijze. Afwijkend van wat? Afwijkend van de MSX-systeem-principes.

We werken tenslotte toch al in machinetaal en in machinetaal hoeft u zich niet aan de algemene richtlijnen te houden. Sterker nog, U bepaalt zelf wat de algemene richtlijnen zullen worden. In machinetaal kunt u het BASIC maar het beste even vergeten en dat geldt dus eveneens voor elk vastpinnend BASIC-protocol.

Een voorbeeld. BASIC beschouwt de hires mode als bitmap, wij beschouwen de hires mode als drie karaktersets. U krijgt in hoofdstuk 19 een treffende illustratie.

## 17 Tekst 40 × 24

In de  $40 \times 24$ -tekst-mode (zie afb. 17.1) neemt de schermtabel 960 (=  $40 \times 24$ ) bytes in beslag. Het scherm bestaat uit een rooster van  $40 \times 24$  karakters. Dit rooster begint in het videogeheugen op geheugenlocatie \$0000.

	111111111111111122222222		
\$0000	0123456789ABCDEF0123456789ABCDEF01234567		\$0800
\$0028			\$0808
\$0050		.....	
\$0078		.....	
\$00A0		.....	
\$00C8		.....	
\$00F0		.....	
\$0118		.....	
\$0140		.....	
\$0168		.....	
\$0190		.....	
\$01B8		.....	
\$01E0		.....	
\$0208		.....	
\$0230		.....	
\$0258		.....	
\$0280		.....	
\$02A8		.....	
\$02D0		.....	
\$02F8		.....	
\$0320		.....	
\$0348		.....	
\$0370		.....	
\$0398		.....	\$1000

Afb. 17.1. Screen mode 0

Iedere waarde (0-255) in dit stukje videogeheugen wijst naar een  $6 \times 8$  pixel-patroon uit de karakterpatroontabel die begint op \$0800.

De kleuren worden in het zevende VDP-register geplaatst. Sprites zijn helaas niet toegestaan.

## 17.1 Scrollen

```
10010 ORIGIN ($c000)
10020 --
10030 $004a CONSTANT RdVrm
10040 $004d CONSTANT WrtVrm
10050 $0056 CONSTANT FilVrm
10060 $00ba CONSTANT IscNtc
10070 $00d5 CONSTANT GtStck
10080 --
10090 CODE
10100 --
10110 :Teller .by $00
10120 :Read .wo $0000
10130 :Write .wo $0000
10140 --
10150 :VPeek
10160     di
10170     call #RdVrm
10180     ei
10190     ret
10200 --
10210 :VPoke
10220     di
10230     call #WrtVrm
10240     ei
10250     ret
10260 --
10270 :VFill
10280     di
10290     call #FilVrm
10300     ei
10310     ret
10320 --
10330 :ScrolUp
10340     ld a,#23
10350     ld (#Teller),a
10360     ld hl,$0000
10370     ld (#Write),hl
10380     ld hl,$0028
10390     ld (#Read),hl
10400 :Up0
10410     ld b,#40
10420 :Up1
10430     ld hl,(#Read)
10440     call #Vpeek
10450     inc hl
```

```

10460 ld (#Read),hl
10470 ld hl,(#Write)
10480 call #Vpoke
10490 inc hl
10500 ld (#Write),hl
10510 djnz #Up1
10520 ld hl,#Teller
10530 dec (hl)
10540 jpif nz,#Up0
10550 ld a,#$20
10560 ld bc,#$0028
10570 ld hl,#$0398
10580 call #VFill
10590 ret
10600 --
10610 :ScrolDown
10620 ld a,#23
10630 ld (#Teller),a
10640 ld hl,#$03bf
10650 ld (#Write),hl
10660 ld hl,#$0397
10670 ld (#Read),hl
10680 :Down0
10690 ld b,#40
10700 :Down1
10710 ld hl,(#Read)
10720 call #Vpeek
10730 dec hl
10740 ld (#Read),hl
10750 ld hl,(#Write)
10760 call #Vpoke
10770 dec hl
10780 ld (#Write),hl
10790 djnz #Down1
10800 ld hl,#Teller
10810 dec (hl)
10820 jpif nz,#Down0
10830 ld a,#$20
10840 ld bc,#$0028
10850 ld hl,#$0000
10860 call #VFill
10870 ret
10880 --
10890 :Init
10900 call #IscNtc
10910 ld a,#0
10920 call #GtStck
10930 cp #1
10940 IF z
10950     THEN
10960         call #ScrolDown
10970     ELSE

```

```

10980      cp #5
10990      IF z THEN
11000          call #ScrolUp
11010      FI
11020      FI
11030      jp #Init
11040  --
11050 ENDCODE

```

..: C000 00 00 00 00	"...."	..: C058 BF 03 22 03	"...."
..: C004 00 F3 CD 4A	".J."	..: C05C C0 21 97 03	".!.."
..: C008 00 FB C9 F3	"...."	..: C060 22 01 C0 06	"...."
..: COOC CD 4D 00 FB	".M.."	..: C064 28 2A 01 C0	
..: C010 C9 F3 CD 56	".V"	..: C068 CD 05 C0 2B	"...+."
..: C014 00 FB C9 3E	".>"	..: C06C 22 01 C0 2A	"...*"
..: C018 17 32 00 C0	".2.."	..: C070 03 C0 CD 0B	"...."
..: C01C 21 00 00 22	".!.."	..: C074 C0 2B 22 03	".+.."
..: C020 03 C0 21 28	".!()."	..: C078 C0 10 EA 21	"...!"
..: C024 00 22 01 C0	"...."	..: C07C 00 C0 35 C2	"..5."
..: C028 06 28 2A 01	".(*."	..: C080 63 C0 3E 20	"c.> "
..: C02C C0 CD 05 C0	"...."	..: C084 01 28 00 21	".(.!"
..: C030 23 22 01 C0	"#..."	..: C088 00 00 C3 11	"...."
..: C034 2A 03 C0 CD	"*..."	..: C08C C0 CD BA 00	"...."
..: C038 0B C0 23 22	".#""	..: C090 3E 00 CD D5	
..: C03C 03 C0 10 EA	"...."	..: C094 00 FE 01 C2	"...."
..: C040 21 00 C0 35	".!.5"	..: C098 A0 C0 CD 52	"...R"
..: C044 C2 28 C0 3E	".(>"	..: C09C C0 C3 A8 C0	"...."
..: C048 20 01 28 00	".(.."	..: COA0 FE 05 C2 A8	"...."
..: C04C 21 98 03 C3	".!.."	..: COA4 C0 CD 17 C0	"...."
..: C050 11 C0 3E 17	".>."	..: COA8 C3 8D C0 00	"...."
..: C054 32 00 C0 21	"2.!"		

..: C005 F3	DI
..: C006 CD 4A 00	CALL \$004A
..: C009 FB	EI
..: C00A C9	RET
..: C00B F3	DI
..: COOC CD 4D 00	CALL \$004D
..: C00F FB	EI
..: C010 C9	RET
..: C011 F3	DI
..: C012 CD 56 00	CALL \$0056
..: C015 FB	EI
..: C016 C9	RET
..: C017 3E 17	LD A,\$17

; C019 32 00 CO LD (\$C000),A  
; C01C 21 00 00 LD HL,\$0000  
; C01F 22 03 CO LD (\$C003),HL  
; C022 21 28 00 LD HL,\$0028  
; C025 22 01 CO LD (\$C001),HL  
; C028 06 28 LD B,\$28  
; C02A 2A 01 CO LD HL,(\$C001)  
; C02D CD 05 CO CALL \$C005  
; C030 23 INC HL  
; C031 22 01 CO LD (\$C001),HL  
; C034 2A 03 CO LD HL,(\$C003)  
; C037 CD 0B CO CALL \$C00B  
; C03A 23 INC HL  
; C03B 22 03 CO LD (\$C003),HL  
; C03E 10 EA DJNZ @C02A  
; C040 21 00 CO LD HL,\$C000  
; C043 35 DEC (HL)  
; C044 C2 28 CO JP NZ,\$C028  
; C047 3E 20 LD A,\$20  
; C049 01 28 00 LD BC,\$0028  
; C04C 21 98 03 LD HL,\$0398  
; C04F C3 11 CO JP \$C011  
; C052 3E 17 LD A,\$17  
; C054 32 00 CO LD (\$C000),A  
; C057 21 BF 03 LD HL,\$03BF  
; C05A 22 03 CO LD (\$C003),HL  
; C05D 21 97 03 LD HL,\$0397  
; C060 22 01 CO LD (\$C001),HL  
; C063 06 28 LD B,\$28  
; C065 2A 01 CO LD HL,(\$C001)  
; C068 CD 05 CO CALL \$C005  
; C06B 2B DEC HL  
; C06C 22 01 CO LD (\$C001),HL  
; C06F 2A 03 CO LD HL,(\$C003)  
; C072 CD 0B CO CALL \$C00B  
; C075 2B DEC HL  
; C076 22 03 CO LD (\$C003),HL  
; C079 10 EA DJNZ @C065  
; C07B 21 00 CO LD HL,\$C000  
; C07E 35 DEC (HL)  
; C07F C2 63 CO JP NZ,\$C063  
; C082 3E 20 LD A,\$20  
; C084 01 28 00 LD BC,\$0028  
; C087 21 00 00 LD HL,\$0000  
; C08A C3 11 CO JP \$C011  
; C08D CD BA 00 CALL \$00BA  
; C090 3E 00 LD A,\$00  
; C092 CD D5 00 CALL \$00D5  
; C095 FE 01 CP \$01  
; C097 C2 A0 CO JP NZ,\$COA0  
; C09A CD 52 CO CALL \$C052

```
.; C09D C3 A8 C0      JP $C0A8
.; COAO FE 05      CP $05
.; COA2 C2 A8 C0      JP NZ,$C0A8
.; COA5 CD 17 C0      CALL $C017
.; COA8 C3 8D C0      JP $C08D
.; COAB 00      NOP
```

In deze screen mode gaan we ‘scrollen’ in machinecode. U weet hoe dat gaat? Tijdens het listen van programmaregels wordt de tekst omhoog gezet en wordt een nieuwe regel onderaan het beeld geplaatst. Dit proces, ‘scrolling’, programmeren wij nu ook; niet alleen omhoog maar ook omlaag.

We gebruiken een niet-cyclische scroll over het gehele scherm. Naar boven bewegend komt dit neer op het volgende:

- tweede regel op de eerste;
- derde regel op de tweede;
- laatste regel op de één na laatste;
- aanvullen met spaties.

Naar beneden scrollen gaat natuurlijk precies andersom.  
Dit alles gebeurt als u op de cursortoetsen drukt.

## 17.2 De routines op een rij

VPEEK haalt een geadresseerde waarde uit het videogeheugen. Het adres staat in HL en de waarde komt in het A-register te staan.

VPOKE plaatst de inhoud van het A-register in het door middel van HL geadresseerde video-adres.

VFILL vult een stuk videogeheugen met een bepaalde waarde. Die waarde komt in de accumulator. De lengte van het te vullen gebied komt in het BC-registerpaar en het beginadres (We vullen naar boven!) komt in het HL-registerpaar.

SCROLUP scrollt natuurlijk omhoog. Eerst wordt een teller (op 23) geïnitialiseerd, evenals het eerste schermadres (\$0000) en het schermadres van de volgende regel (\$0028).

Een regel is 40 tekens breed dus we maken een binnenlus met lengte 40. Vervolgens gaan we regels verplaatsen, van beneden naar boven. Dit alles 23 keer. Na het scrollen maken we gebruik van de VFILL-routine om de onderste regel snel van spaties te voorzien.

SCROLDOWN scrollt omlaag. Wederom wordt een teller (op 23) geïnitialiseerd, evenals het laatste schermadres (\$03BF) en het schermadres van de vorige regel (\$0397).

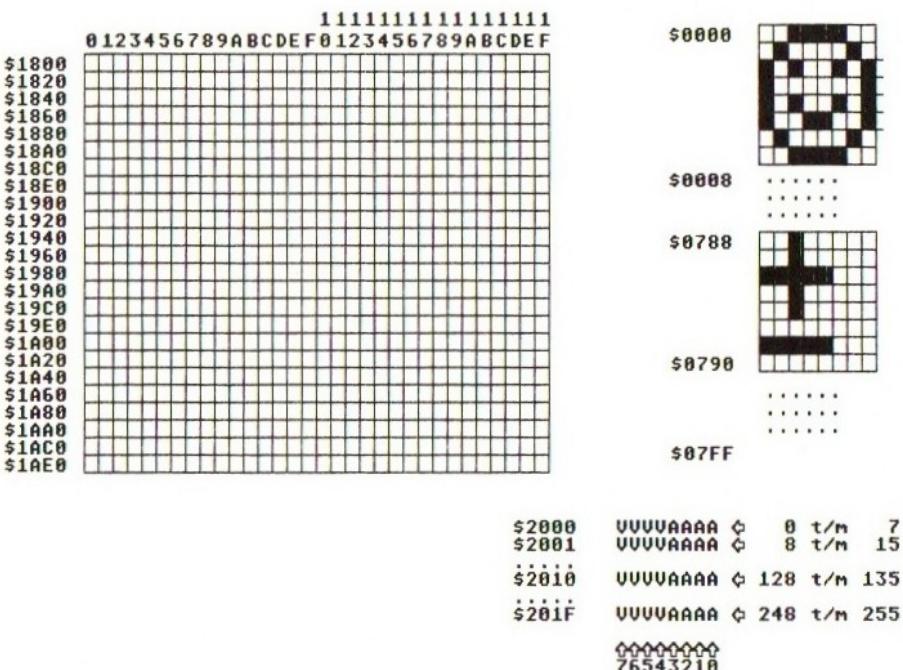
Een regel is 40 breed, dus we maken een binnenlus met lengte 40. Vervolgens gaan we regels verplaatsen, van boven naar beneden. Dit alles natuurlijk 23 keer. Na het scrollen maken we gebruik van de VFILL-routine om de bovenste regel snel van spaties te voorzien.

INIT gaat ervan uit dat het scherm al in de  $40 \times 24$ -mode staat (u komt tenslotte uit de HERBYMON) en kijkt naar de CTRL/STOP-toestand. Niets aan de hand? Dan worden de cursortoetsen uitgelezen en aan de hand daarvan wordt actie ondernomen.

# 18 Tekst 32 × 24

In de 32 × 24-tekst-mode (zie afb. 18.1) neemt de schermtabel 768 (= 32 × 24) bytes in beslag. Het scherm bestaat uit een rooster van 32 × 24 tekens, in het videogeheugen beginnend op \$1800.

Elke waarde (0-255) in dit stukje videogeheugen wijst naar een patroon van 8 × 8 pixels uit de karakterpatroontabel die op \$0000 begint.



Afb. 18.1. Screen mode 1

De karakterkleuren (voorgrond en achtergrond) worden per groep van 8 in het videogeheugen vanaf adres \$2000 geplaatst. Deze kleurentabel is dus 32 bytes groot.

Sprites zijn toegestaan; de attributentabel begint op \$1B00 en de patroontabel op \$3800.

## 18.1 Karakters ontwerpen

```
10000 ORIGIN ($c000)
10010 --
10020 $004a CONSTANT RdVrm
10030 $004d CONSTANT WrtVrm
10040 $006f CONSTANT Init32
10050 $009f CONSTANT ChGet
10060 $00a2 CONSTANT ChPut
10070 $00ba CONSTANT IscNtc
10080 $0156 CONSTANT KilBuf
10090 --
10100 $f3af CONSTANT Lin32
10110 $f3dd CONSTANT CrsX
10120 $f3dc CONSTANT CrsY
10130 $fcfa7 CONSTANT EscCnt
10140 --
10150 CODE
10160 --
10170 :Help      .by $00
10180 :Teller    .by $00
10190 :Positie   .wo $0000
10200 --
10210 :VPeek
10220     di
10230     call #RdVrm
10240     ei
10250     ret
10260 --
10270 :VPoke
10280     di
10290     call #WrtVrm
10300     ei
10310     ret
10320 --
10330 :GetPos
10340     ld a,#0
10350     ld b,a
10360     ld a,(#Teller)
10370     sla a
10380     rl b
10390     sla a
```

```
10400    rl b
10410    sla a
10420    rl b
10430    ld (#Positie),a
10440    ld a,b
10450    ld (#Positie+1),a
10460    ret
10470 --
10480 :GetLin
10490    ld a,(#CrsY)
10500 :GetLin0
10510    cp #11
10520    retif z
10530    ld de,(#Positie)
10540    inc de
10550    ld (#Positie),de
10560    dec a
10570    jp #GetLin0
10580 --
10590 :GetCol
10600    ld b,#128
10610    ld a,(#CrsX)
10620 :GetCol0
10630    cp #13
10640    retif z
10650    srl b
10660    dec a
10670    jp #GetCol0
10680 --
10690 :Setup
10700    ld hl,$1800
10710    ld a,#0
10720 :Setup0
10730    call #VPoke
10740    inc hl
10750    inc a
10760    jpif nz,#Setup0
10770    call #GetPos
10780    ld d,#8
10790    ld hl,$194c
10800 :Setup1
10810    ld e,#8
10820    push hl
10830    ld hl,(#Positie)
10840    call #VPeek
10850    inc hl
10860    ld (#Positie),hl
10870    pop hl
10880    ld b,#128
10890 :Setup2
10900    ld (#Help),a
10910    and b
```

```
10920    IF z
10930        THEN
10940            ld a,#196
10950        ELSE
10960            ld a,#219
10970    FI
10980    call #VPoke
10990    inc hl
11000    srl b
11010    ld a,(#Help)
11020    dec e
11030    jpif nz,#Setup2
11040    ld bc,#$0018
11050    add hl,bc
11060    dec d
11070    jpif nz,#Setup1
11080    ret
11090 --
11100 :Plus
11110    ld hl,#Teller
11120    inc (hl)
11130    ret
11140 --
11150 :Min
11160    ld hl,#Teller
11170    dec (hl)
11180    ret
11190 --
11200 :Up
11210    ld a,(#CrsY)
11220    cp #11
11230    retif z
11240    ld a,#$1e
11250    jp #ChPut
11260 --
11270 :Down
11280    ld a,(#CrsY)
11290    cp #18
11300    retif z
11310    ld a,#$1f
11320    jp #ChPut
11330 --
11340 :Left
11350    ld a,(#CrsX)
11360    cp #13
11370    retif z
11380    ld a,#$1d
11390    jp #ChPut
11400 --
11410 :Right
11420    ld a,(#CrsX)
```

```
11430      cp #20
11440      retif z
11450      ld a,#$1c
11460      jp #ChPut
11470  --
11480  :Pixel
11490      call #GetPos
11500      call #GetLin
11510      call #GetCol
11520      ld hl,(#Positie)
11530      call #VPeek
11540      xor b
11550      call #VPoke
11560      jp #Right
11570  --
11580  :JmpTab    .wo #Plus
11590          .wo #Min
11600          .wo #Up
11610          .wo #Down
11620          .wo #Left
11630          .wo #Right
11640          .wo #Pixel
11650  --
11660  :KeyTab    .by 43
11670          .by 45
11680          .by $1e
11690          .by $1f
11700          .by $1d
11710          .by $1c
11720          .by 32
11730          .by 0
11740  --
11750  :SelfMod
11760      jp #SelfMod
11770  --
11780  :Init
11790      ld a,#32
11800      ld (#Lin32),a
11810      call Init32
11820      ld a,#13
11830      ld (#CrsX),a
11840      ld a,#11
11850      ld (#CrsY),a
11860  :Init0
11870      call #Setup
11880      call #KilBuf
11890  :Init1
11900      call #IscNtc
11910      ld a,#0
11920      ld (#EscCnt),a
11930      call #ChGet
```

```

11940    ld (#Help),a
11950    ld hl,#KeyTab
11960    ld de,#JmpTab
11970 :Init2
11980    ld a,(hl)
11990    cp #0
12000    jpif z,#Init1
12010    ld a,(#Help)
12020    cp (hl)
12030    IF z THEN
12040            ld a,(de)
12050            ld (#SelfMod+1),a
12060            inc de
12070            ld a,(de)
12080            ld (#SelfMod+2),a
12090            call #SelfMod
12100            jp #Init0
12110    FI
12120    inc hl
12130    inc de
12140    inc de
12150    jp #Init2
12160 --
12170 ENDCODE

```

.. C000 00 00 00 00	"...."	.. C054 3C C2 50 C0	"<.P."
.. C004 F3 CD 4A 00	".J."	.. C058 CD 10 C0 16	"...."
.. C008 FB C9 F3 CD	"...."	.. C05C 08 21 4C 19	".!L."
.. C00C 4D 00 FB C9	"M..."	.. C060 1E 08 E5 2A	"...*"
.. C010 3E 00 47 3A	.. C064 02 C0 CD 04	"...."	
.. C014 01 C0 CB 27	"...,"	.. C068 C0 23 22 02	".#"."
.. C018 CB 10 CB 27	"...,"	.. C06C C0 E1 06 80	"...."
.. C01C CB 10 CB 27	"...,"	.. C070 32 00 C0 A0	"2..."
.. C020 CB 10 32 02	"..2."	.. C074 C2 7C C0 3E	". .>"
.. C024 C0 78 32 03	".x2."	.. C078 C4 C3 7E C0	"...."
.. C028 C0 C9 3A DC	"...:."	.. C07C 3E DB CD 0A	
.. C02C F3 FE 0B C8	"...."	.. C080 C0 23 CB 38	".#.8"
.. C030 ED 5B 02 C0	".[..]"	.. C084 3A 00 C0 1D	
.. C034 13 ED 53 02	".S."	.. C088 C2 70 C0 01	".p."
.. C038 C0 3D C3 2D	".=.-"	.. C08C 18 00 09 15	"...."
.. C03C C0 06 80 3A	"...:."	.. C090 C2 60 C0 C9	".'.."
.. C040 DD F3 FE 0D	"...."	.. C094 21 01 C0 34	".!..4"
.. C044 C8 CB 38 3D	".8=."	.. C098 C9 21 01 C0	".!.."
.. C048 C3 42 C0 21	".B.!"	.. C09C 35 C9 3A DC	"5..."
.. C04C 00 18 3E 00	".>."	.. COA0 F3 FE 0B C8	"...."
.. C050 CD 0A C0 23	".#"	.. COA4 3E 1E C3 A2	

```

.: COA8 00 3A DC F3 ". . ." .: C0F8 C0 3E 20 32 ".> 2"
.: COAC FE 12 C8 3E ". . .>" .: COFC AF F3 CD 6F ". . .o"
.: COB0 1F C3 A2 00 ". . ." .: C100 00 3E 0D 32 ".>.2"
.: COB4 3A DD F3 FE ". . ." .: C104 DD F3 3E 0B ". . .>."
.: COB8 0D C8 3E 1D ". . .>." .: C108 32 DC F3 CD "2... "
.: COBC C3 A2 00 3A ". . ." .: C10C 4B C0 CD 56 "K..V"
.: COCO DD F3 FE 14 ". . ." .: C110 01 CD BA 00 ". . ."
.: COC4 C8 3E 1C C3 ". . .>." .: C114 3E 00 32 A7 ">.2."
.: COC8 A2 00 CD 10 ". . ." .: C118 FC CD 9F 00 ". . ."
.: COCC CO CD 2A CO ". . *." .: C11C 32 00 C0 21 "2..! "
.: COD0 CD 3D C0 2A ". . *." .: C120 EE C0 11 E0 ". . ."
.: COD4 02 C0 CD 04 ". . ." .: C124 C0 7E FE 00 ". . ."
.: COD8 CO A8 CD 0A ". . ." .: C128 CA 11 C1 3A ". . ."
.: CODC CO C3 BF CO ". . ." .: C12C 00 C0 BE C2 ". . ."
.: COE0 94 C0 99 CO ". . ." .: C130 41 C1 1A 32 "A..2"
.: COE4 9E C0 A9 CO ". . ." .: C134 F7 C0 13 1A ". . ."
.: COE8 B4 C0 BF CO ". . ." .: C138 32 F8 C0 CD "2... "
.: COEC CA C0 2B 2D ". . .+." .: C13C F6 C0 C3 0B ". . ."
.: COFO 1E 1F 1D 1C ". . ." .: C140 C1 23 13 13 ". #.. "
.: COF4 20 00 C3 F6 ". . ." .: C144 C3 25 C1 3E ".%.>"

```

```

.; C004 F3           DI
.; C005 CD 4A 00     CALL $004A
.; C008 FB           EI
.; C009 C9           RET
.; C00A F3           DI
.; C00B CD 4D 00     CALL $004D
.; C00E FB           EI
.; C00F C9           RET
.; C010 3E 00         LD A,$00
.; C012 47           LD B,A
.; C013 3A 01 C0     LD A,($C001)
.; C016 CB 27         SLA A
.; C018 CB 10         RL B
.; C01A CB 27         SLA A
.; C01C CB 10         RL B
.; C01E CB 27         SLA A
.; C020 CB 10         RL B
.; C022 32 02 C0     LD ($C002),A
.; C025 78           LD A,B
.; C026 32 03 C0     LD ($C003),A
.; C029 C9           RET
.; C02A 3A DC F3     LD A,($F3DC)
.; C02D FE 0B         CP $0B
.; C02F C8           RET Z
.; C030 ED 5B 02 C0   LD DE,($C002)
.; C034 13           INC DE
.; C035 ED 53 02 C0   LD ($C002),DE

```

; C039 3D DEC A  
; C03A C3 2D CO JP \$C02D  
; C03D 06 80 LD B,\$80  
; C03F 3A DD F3 LD A,(\$F3DD)  
; C042 FE 0D CP \$0D  
; C044 C8 RET Z  
; C045 CB 38 SRL B  
; C047 3D DEC A  
; C048 C3 42 CO JP \$C042  
; C04B 21 00 18 LD HL,\$1800  
; C04E 3E 00 LD A,\$00  
; C050 CD 0A CO CALL \$C00A  
; C053 23 INC HL  
; C054 3C INC A  
; C055 C2 50 CO JP NZ,\$C050  
; C058 CD 10 CO CALL \$C010  
; C05B 16 08 LD D,\$08  
; C05D 21 4C 19 LD HL,\$194C  
; C060 1E 08 LD E,\$08  
; C062 E5 PUSH HL  
; C063 2A 02 CO LD HL,(\$C002)  
; C066 CD 04 CO CALL \$C004  
; C069 23 INC HL  
; C06A 22 02 CO LD (\$C002),HL  
; C06D E1 POP HL  
; C06E 06 80 LD B,\$80  
; C070 32 00 CO LD (\$C000),A  
; C073 A0 AND B  
; C074 C2 7C CO JP NZ,\$C07C  
; C077 3E C4 LD A,\$C4  
; C079 C3 7E CO JP \$C07E  
; C07C 3E DB LD A,\$DB  
; C07E CD 0A CO CALL \$C00A  
; C081 23 INC HL  
; C082 CB 38 SRL B  
; C084 3A 00 CO LD A,(\$C000)  
; C087 1D DEC E  
; C088 C2 70 CO JP NZ,\$C070  
; C08B 01 18 00 LD BC,\$0018  
; C08E 09 ADD HL,BC  
; C08F 15 DEC D  
; C090 C2 60 CO JP NZ,\$C060  
; C093 C9 RET  
; C094 21 01 CO LD HL,\$C001  
; C097 34 INC (HL)  
; C098 C9 RET  
; C099 21 01 CO LD HL,\$C001  
; C09C 35 DEC (HL)  
; C09D C9 RET  
; C09E 3A DC F3 LD A,(\$F3DC)  
; COA1 FE 0B CP \$0B  
; COA3 C8 RET Z

```

.; COA4 3E 1E LD A,$1E
.; COA6 C3 A2 00 JP $00A2
.; COA9 3A DC F3 LD A,($F3DC)
.; COAC FE 12 CP $12
.; COAE C8 RET Z
.; COAF 3E 1F LD A,$1F
.; COB1 C3 A2 00 JP $00A2
.; COB4 3A DD F3 LD A,($F3DD)
.; COB7 FE 0D CP $0D
.; COB9 C8 RET Z
.; COBA 3E 1D LD A,$1D
.; COBC C3 A2 00 JP $00A2
.; COBF 3A DD F3 LD A,($F3DD)
.; COC2 FE 14 CP $14
.; COC4 C8 RET Z
.; COC5 3E 1C LD A,$1C
.; COC7 C3 A2 00 JP $00A2
.; COCA CD 10 CO CALL $C010
.; COCD CD 2A CO CALL $C02A
.; COD0 CD 3D CO CALL $C03D
.; COD3 2A 02 CO LD HL,($C002)
.; COD6 CD 04 CO CALL $C004
.; COD9 A8 XOR B
.; CODA CD 0A CO CALL $C00A
.; CODD C3 BF CO JP $COBF
.; COEO 94 SUB H
.; COE1 CO RET NZ
.; COE2 99 SBC A,C
.; COE3 CO RET NZ
.; COE4 9E SBC A,(HL)
.; COE5 CO RET NZ
.; COE6 A9 XOR C
.; COE7 CO RET NZ
.; COE8 B4 OR H
.; COE9 CO RET NZ
.; COEA BF CP A
.; COEB CO RET NZ
.; COEC CA CO 2B JP Z,$2BC0
.; COEF 2D DEC L
.; COFO 1E 1F LD E,$1F
.; COF2 1D DEC E
.; COF3 1C INC E
.; COF4 20 00 JR NZ,@COF6
.; COF6 C3 F6 CO JP $COF6
.; COF9 3E 20 LD A,$20
.; COFB 32 AF F3 LD ($F3AF),A
.; COFE CD 6F 00 CALL $006F
.; C101 3E 0D LD A,$0D
.; C103 32 DD F3 LD ($F3DD),A
.; C106 3E 0B LD A,$0B
.; C108 32 DC F3 LD ($F3DC),A
.; C10B CD 4B CO CALL $C04B

```

```

.; C10E CD 56 01      CALL $0156
.; C111 CD BA 00      CALL $00BA
.; C114 3E 00          LD A,$00
.; C116 32 A7 FC      LD ($FCA7),A
.; C119 CD 9F 00      CALL $009F
.; C11C 32 00 C0      LD ($C000),A
.; C11F 21 EE C0      LD HL,$COEE
.; C122 11 E0 C0      LD DE,$COEO
.; C125 7E             LD A,(HL)
.; C126 FE 00          CP $00
.; C128 CA 11 C1      JP Z,$C111
.; C12B 3A 00 C0      LD A,($C000)
.; C12E BE             CP (HL)
.; C12F C2 41 C1      JP NZ,$C141
.; C132 1A             LD A,(DE)
.; C133 32 F7 C0      LD ($COF7),A
.; C136 13             INC DE
.; C137 1A             LD A,(DE)
.; C138 32 F8 C0      LD ($COF8),A
.; C13B CD F6 C0      CALL $COF6
.; C13E C3 0B C1      JP $C10B
.; C141 23             INC HL
.; C142 13             INC DE
.; C143 13             INC DE
.; C144 C3 25 C1      JP $C125
.; C147 3E 1D          LD A,$1D
.

```

Als programmeervoorbeeld een karakterset-editor. Het wordt nu mogelijk om zelf een echte  $8 \times 8$  karakterset te ontwerpen.

Natuurlijk kunt u deze zelf ontworpen leestekens ook opslaan. Dit gaat vanuit BASIC met een video-SAVE-opdracht. Zo'n opgeslagen karakterset kunt u naderhand met een video-LOAD-commando weer ophalen.

U kunt met behulp van de karaktertabel uit appendix B en de listings heel eenvoudig uitzoeken hoe de karakterontwerper nu precies werkt. Dit zelf uitzoeken is niet zonder reden. Als u namelijk weet hoe het edit-principe in elkaar steekt, wordt het zeer eenvoudig om zelf commando's toe te voegen en eigen editors te ontwerpen.

Welke commando's en editors? Dat hangt van uw eigen fantasie en behoeften af.

## 18.2 De routines op een rij

VPEEK, u inmiddels bekend uit de vorige listing.

VPOKE, idem dito.

GETPOS berekent de plaats waar de informatie over het huidige karakter staat. Dit komt neer op een 'karakter maal 8'-berekening.

We doen dit met behulp van een schuifinstructie. Denkt u even mee! Als we een bitpatroon naar links schuiven, doen we niets meer dan vermenigvuldigen met 2. Drie keer schuiven is dus  $2 \times 2 \times 2 = 8$ . (Even tussendoor: naar rechts schuiven is delen door 2...)

GETLIN kijkt welk byte uit het patroon van 8 thans wordt bewerkt. We kijken hiervoor naar de Y-waarde van de cursor.

GETCOL kijkt welk BIT uit het byte thans wordt bewerkt. We kijken hiervoor naar de X-waarde van de cursor.

SETUP tovert een karakterset en het thans bekeken karakter te voorschijn. Eerst initialiseren we het HL-paar op \$1800 (begin van de schermtabel), en we 'videopoken' de waarden 0-255 op het beeld.

Van het nu bekeken karakter wordt de positie berekend en er wordt 8 (de hoogte) maal 8 keer (de breedte) het juiste pixel afgedrukt.

PLUS verhoogt het huidige karakter met één.

MIN verlaagt het huidige karakter met één.

UP verplaatst, indien mogelijk, de cursor één locatie naar beneden.

DOWN verplaatst, indien mogelijk, de cursor één positie naar boven.

LEFT verplaatst, indien mogelijk, de cursor één positie naar links.

RIGHT verplaatst, indien mogelijk, de cursor één positie naar rechts.

PIXEL berekent de pixelpositie van de cursor en XOR-t deze waarde met de desbetreffende bitadressering. Pixel aan wordt pixel uit en andersom.

JMPTAB bevat de adressen die worden aangeroepen als er op een commandotoets wordt gedrukt.

KEYTAB bevat de ASCII-waarden die horen bij de commandotoetsen. Een KEYTAB-adressering is niet meer dan een pointer naar de vorige tabel.

SELMOD is een CALL-routine die door de key scan wordt veranderd als er op een commandotoets wordt gedrukt.

Stel dat de PLUS-routine moet worden aangeroepen. Het low-byte van PLUS komt van KEYTAB en wordt op SELMOD + 1 geplaatst, Het high-byte van PLUS komt van KEYTAB + 1 en wordt op SELMOD + 2 geplaatst.

INIT stelt de juiste schermbreedte en screen mode in, waarna (met CTRL/STOP-check) de juiste commando's worden bepaald enz.

# 19 High resolution

In de high resolution mode of hires mode (zie afb. 19.1) neemt de schermtabel 768 (= 32 × 24) bytes in beslag. Het scherm bestaat uit een rooster van 32 × 24 karakters, in het videogeheugen beginnend op \$1800.

In tegenstelling tot de screen mode uit het vorige hoofdstuk kan iedere schermpositie nu over een eigen plaats in de karakterset beschikken. Dit wordt gedaan door drie verschillende karaktersets te gebruiken. Een karakterset voor de bovenste 8 regels (\$0000-\$07FF), een karakterset voor de middelste 8 regels (\$0800-\$0FFF) en een karakterset voor de onderste acht regels (\$1000-\$17FF). Elke waarde op een schermgedeelte (0-255) wijst naar een plaats in de bijpassende karakterset. Een 8 × 8 pixelpatroon dus, met 6K aan karakterinformatie.

De karakterkleuren (voorgrond en achtergrond) worden per regel bepaald. Voor de eerste karakterset staat de kleurentabel op \$2000-\$27FF, voor de tweede karakterset op \$2800-\$2FFF en voor de derde karakterset op \$3000-\$37FF. Er is dus ook 6K aan kleurinformatie.

Sprites zijn toegestaan; de attributentabel begint op \$1B00 en de patroontabel begint op \$3800.

## 19.1 Spelletje spelen?

```
10000 ORIGIN ($c000)
10010 --
10020 $000c CONSTANT RdSlt
10030 $0056 CONSTANT FilVrm
10040 $005c CONSTANT LDirVm
10050 $0072 CONSTANT IniGrp
10060 $00ba CONSTANT IscNtc
10070 $00d5 CONSTANT GtStck
10080 --
10090 $f91f CONSTANT CgPnt
10100 --
10110 CODE
10120 --
10130 :Big      .by $00
10140 :Tiny     .by $00
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
\$1800	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
\$1829	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
\$1848	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
\$1868	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
\$1888	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
\$18A8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
\$18C8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
\$18E8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
\$1908	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2			
\$1928	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2			
\$1948	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2			
\$1968	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2			
\$1988	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2			
\$19A8	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2			
\$19C8	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2			
\$19E8	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2			
\$1A08	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3			
\$1A28	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3			
\$1A48	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3			
\$1A68	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3			
\$1A88	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3			
\$1AA8	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3			
\$1AC8	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3			
\$1AE8	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3			

\$0000		\$2000	UUUUAAAA	....	\$1000		\$3000	UUUUAAAA	....	
	\$2001	UUUUAAAA	....		\$3001	UUUUAAAA	....			
	\$2002	UUUUAAAA	....		\$3002	UUUUAAAA	....			
	\$2003	UUUUAAAA	....		\$3003	UUUUAAAA	....			
	\$2004	UUUUAAAA	....		\$3004	UUUUAAAA	....			
	\$2005	UUUUAAAA	....		\$3005	UUUUAAAA	....			
	\$2006	UUUUAAAA	....		\$3006	UUUUAAAA	....			
	\$2007	UUUUAAAA	....		\$3007	UUUUAAAA	....			
\$0008	....	....	....	....	\$1008	....	....	....	....	
	....	....	....	....		....	....	....	....	
\$0788		\$2788	UUUUAAAA	....	\$1788		\$3788	UUUUAAAA	....	
	\$2789	UUUUAAAA	....		\$3789	UUUUAAAA	....			
	\$278A	UUUUAAAA	....		\$378A	UUUUAAAA	....			
	\$278B	UUUUAAAA	....		\$378B	UUUUAAAA	....			
	\$278C	UUUUAAAA	....		\$378C	UUUUAAAA	....			
	\$278D	UUUUAAAA	....		\$378D	UUUUAAAA	....			
	\$278E	UUUUAAAA	....		\$378E	UUUUAAAA	....			
	\$278F	UUUUAAAA	....		\$378F	UUUUAAAA	....			
\$0790	....	....	....	....	\$1798	....	....	....	....	
	....	....	....	....		....	....	....	....	
\$07FF	\$27FF	UUUUAAAA	....		\$17FF	\$37FF	UUUUAAAA	....		
	00000000	00000000	00000000			76543218	00000000	00000000		

Afb. 19.1. Screen mode 2

```
10150  --
10160  :Buffer    .wo $9000
10170  :Game      .wo $3000
10180  :Info      .wo $6000
10190  --
10200  :VFill
10210    di
10220    call #FilVrm
10230    ei
10240    ret
10250  --
10260  :RVMove
10270    di
10280    call #LDirVm
10290    ei
10300    ret
10310  --
10320  :Up
10330    ld hl,#Big
10340    ld a,(hl)
10350    cp #0
10360    retif z
10370    dec (hl)
10380    ret
10390  --
10400  :Down
10410    ld hl,#Big
10420    ld a,(hl)
10430    cp #240
10440    retif z
10450    inc (hl)
10460    ret
10470  --
10480  :Right
10490    ld hl,#Tiny
10500    dec (hl)
10510    ret
10520  --
10530  :Left
10540    ld hl,#Tiny
10550    inc (hl)
10560    ret
10570  --
10580  :ShowGame
10590    ld a,(#Big)
10600    ld l,a
10610    ld h,#0
10620    sla l
10630    rl h
10640    sla l
10650    rl h
```

```
10660    sla l
10670    rl h
10680    sla l
10690    rl h
10700    sla l
10710    rl h
10720    ld bc,(#Game)
10730    add hl,bc
10740    ld bc,#$0200
10750    ld de,#$1800
10760    jp #RVMove
10770    --
10780    :ShowInfo
10790    ld de,(#Buffer)
10800    ld hl,(#Info)
10810    ld a,(#Tiny)
10820    ld l,a
10830    ld c,#8
10840    :ShowInfo0
10850    ld b,#32
10860    :ShowInfo1
10870    ld a,(hl)
10880    ld (de),a
10890    inc de
10900    inc l
10910    djnz #ShowInfo1
10920    inc h
10930    ld a,(#Tiny)
10940    ld l,a
10950    dec c
10960    jpif nz,#ShowInfo0
10970    ld bc,#$0100
10980    ld de,#$1a00
10990    ld hl,(#Buffer)
11000    jp #RVMove
11010    --
11020    :Joy
11030    ld a,#0
11040    call #GtStck
11050    ld b,a
11060    cp #1
11070    IF z
11080        THEN
11090            call #Up
11100        FI
11110    ld a,b
11120    cp #3
11130    IF z
11140        THEN
11150            call #Right
11160        FI
11170    ld a,b
```

```
11180      cp #5
11190      IF z
11200          THEN
11210              call #Down
11220      FI
11230      ld a,b
11240      cp #7
11250      IF z
11260          THEN
11270              call #Left
11280      FI
11290      call #ShowGame
11300      jp #ShowInfo
11310  --
11320 :Init
11330      call #IniGrp
11340      ld bc,$0800
11350      ld de,(#Buffer)
11360      ld hl,(#CgPnt+1)
11370 :Init0
11380      push bc
11390      push de
11400      ld a,(#CgPnt)
11410      call #RdSlt
11420      pop de
11430      pop bc
11440      ld (de),a
11450      inc de
11460      inc hl
11470      dec c
11480      jpif nz,#Init0
11490      dec b
11500      jpif nz,#Init0
11510      ld bc,$0800
11520      ld de,$0000
11530      ld hl,(#Buffer)
11540      call #RVMove
11550      ld bc,$0800
11560      ld de,$0800
11570      ld hl,(#Buffer)
11580      call #RVMove
11590      ld bc,$0800
11600      ld de,$1000
11610      ld hl,(#Buffer)
11620      call #RVMove
11630      ld a,%00111100
11640      ld bc,$1000
11650      ld hl,$2000
11660      call #VFill
11670      ld a,%11110001
11680      ld bc,$0800
11690      ld hl,$3000
```

```

11700    call #VFill
11710 :Init1
11720    call #IscNtc
11730    call #Joy
11740    jp #Init1
11750 --
11760 ENDCODE

```

..: C000 00 00 00 90	"...."	..: C08C C0 CD 14 C0	"...."
..: C004 00 30 00 60	"..0.."	..: C090 78 FE 03 C2	"x...&"
..: C008 F3 CD 56 00	"..V.."	..: C094 99 C0 CD 26	"...&.."
..: C00C FB C9 F3 CD	"...."	..: C098 C0 78 FE 05	"..x.."
..: C010 5C 00 FB C9	"\...."	..: C09C C2 A2 C0 CD	"...."
..: C014 21 00 C0 7E	"!...."	..: COA0 1D C0 78 FE	"..x.."
..: C018 FE 00 C8 35	"...5.."	..: COA4 07 C2 AB C0	"...."
..: C01C C9 21 00 C0	"!...."	..: COA8 CD 2B C0 CD	"..+.."
..: C020 7E FE F0 C8	"...."	..: COAC 30 C0 C3 58	"O..X"
..: C024 34 C9 21 01	"4.!.."	..: COBO C0 CD 72 00	"..r.."
..: C028 C0 35 C9 21	"..5!.."	..: COB4 01 00 08 ED	"...."
..: C02C 01 C0 34 C9	"..4.."	..: COB8 5B 02 C0 2A	"[..*.."
..: C030 3A 00 C0 6F	"...o.."	..: COBC 20 F9 C5 D5	"...."
..: C034 26 00 CB 25	"&..%"	..: COCO 3A 1F F9 CD	"...."
..: C038 CB 14 CB 25	"...%"	..: COC4 0C 00 D1 C1	"...."
..: C03C CB 14 CB 25	"...%"	..: COC8 12 13 23 0D	"..#.."
..: C040 CB 14 CB 25	"...%"	..: COCC C2 BE C0 05	"...."
..: C044 CB 14 CB 25	"...%"	..: COD0 C2 BE C0 01	"...."
..: C048 CB 14 ED 4B	"...K.."	..: COD4 00 08 11 00	"...."
..: C04C 04 C0 09 01	"...."	..: COD8 00 2A 02 C0	"..*.."
..: C050 00 02 11 00	"...."	..: CODC CD 0E C0 01	"...."
..: C054 18 C3 0E C0	"...."	..: COEO 00 08 11 00	"...."
..: C058 ED 5B 02 C0	"..[.."	..: COE4 08 2A 02 C0	"..*.."
..: C05C 2A 06 C0 3A	"*..."	..: COE8 CD 0E C0 01	"...."
..: C060 01 C0 6F 0E	"..o.."	..: COEC 00 08 11 00	"...."
..: C064 08 06 20 7E	"...."	..: COFO 10 2A 02 C0	"..*.."
..: C068 12 13 2C 10	"...,"	..: COF4 CD 0E C0 3E	"...>.."
..: C06C FA 24 3A 01	"..\$.."	..: COF8 3C 01 00 10	"<...,"
..: C070 C0 6F 0D C2	"..o.."	..: COFC 21 00 20 CD	"!.."
..: C074 65 C0 01 00	"e...,"	..: C100 08 C0 3E F1	"..>.."
..: C078 01 11 00 1A	"...."	..: C104 01 00 08 21	"....!"
..: C07C 2A 02 C0 C3	"*...,"	..: C108 00 30 CD 08	"..0.."
..: C080 0E C0 3E 00	"..>.."	..: C10C C0 CD BA 00	"...."
..: C084 CD D5 00 47	"...G.."	..: C110 CD 82 C0 C3	"...."
..: C088 FE 01 C2 90	"...."	..: C114 OD C1 32 A7	"..2.."

```
; C008 F3          DI
; C009 CD 56 00    CALL $0056
; C00C FB          EI
; C00D C9          RET
; C00E F3          DI
; C00F CD 5C 00    CALL $005C
; C012 FB          EI
; C013 C9          RET
; C014 21 00 C0    LD HL,$C000
; C017 7E          LD A,(HL)
; C018 FE 00        CP $00
; C01A C8          RET Z
; C01B 35          DEC (HL)
; C01C C9          RET
; C01D 21 00 C0    LD HL,$C000
; C020 7E          LD A,(HL)
; C021 FE F0        CP $FO
; C023 C8          RET Z
; C024 34          INC (HL)
; C025 C9          RET
; C026 21 01 C0    LD HL,$C001
; C029 35          DEC (HL)
; C02A C9          RET
; C02B 21 01 C0    LD HL,$C001
; C02E 34          INC (HL)
; C02F C9          RET
; C030 3A 00 C0    LD A,($C000)
; C033 6F          LD L,A
; C034 26 00        LD H,$00
; C036 CB 25        SLA L
; C038 CB 14        RL H
; C03A CB 25        SLA L
; C03C CB 14        RL H
; C03E CB 25        SLA L
; C040 CB 14        RL H
; C042 CB 25        SLA L
; C044 CB 14        RL H
; C046 CB 25        SLA L
; C048 CB 14        RL H
; C04A ED 4B 04 C0 LD BC,($C004)
; C04E 09          ADD HL,BC
; C04F 01 00 02    LD BC,$0200
; C052 11 00 18    LD DE,$1800
; C055 C3 0E C0    JP $C00E
; C058 ED 5B 02 C0 LD DE,($C002)
; C05C 2A 06 C0    LD HL,($C006)
; C05F 3A 01 C0    LD A,($C001)
; C062 6F          LD L,A
; C063 0E 08        LD C,$08
; C065 06 20        LD B,$20
; C067 7E          LD A,(HL)
```

; C068 12 LD (DE),A  
; C069 13 INC DE  
; C06A 2C INC L  
; C06B 10 FA DJNZ @C067  
; C06D 24 INC H  
; C06E 3A 01 CO LD A,(\$C001)  
; C071 6F LD L,A  
; C072 0D DEC C  
; C073 C2 65 CO JP NZ,\$C065  
; C076 01 00 01 LD BC,\$0100  
; C079 11 00 1A LD DE,\$1A00  
; C07C 2A 02 CO LD HL,(\$C002)  
; C07F C3 0E CO JP \$C00E  
; C082 3E 00 LD A,\$00  
; C084 CD D5 00 CALL \$00D5  
; C087 47 LD B,A  
; C088 FE 01 CP \$01  
; C08A C2 90 CO JP NZ,\$C090  
; C08D CD 14 CO CALL \$C014  
; C090 78 LD A,B  
; C091 FE 03 CP \$03  
; C093 C2 99 CO JP NZ,\$C099  
; C096 CD 26 CO CALL \$C026  
; C099 78 LD A,B  
; C09A FE 05 CP \$05  
; C09C C2 A2 CO JP NZ,\$COA2  
; C09F CD 1D CO CALL \$C01D  
; COA2 78 LD A,B  
; COA3 FE 07 CP \$07  
; COA5 C2 AB CO JP NZ,\$COAB  
; COA8 CD 2B CO CALL \$C02B  
; COAB CD 30 CO CALL \$C030  
; COAE C3 58 CO JP \$C058  
; COB1 CD 72 00 CALL \$0072  
; COB4 01 00 08 LD BC,\$0800  
; COB7 ED 5B 02 CO LD DE,(\$C002)  
; COBB 2A 20 F9 LD HL,(\$F920)  
; COBE C5 PUSH BC  
; COBF D5 PUSH DE  
; COCO 3A 1F F9 LD A,(\$F91F)  
; COC3 CD 0C 00 CALL \$000C  
; COC6 D1 POP DE  
; COC7 C1 POP BC  
; COC8 12 LD (DE),A  
; COC9 13 INC DE  
; COCA 23 INC HL  
; COCB 0D DEC C  
; COCC C2 BE CO JP NZ,\$COBE  
; COCF 05 DEC B  
; COD0 C2 BE CO JP NZ,\$COBE  
; COD3 01 00 08 LD BC,\$0800  
; COD6 11 00 00 LD DE,\$0000

```

.; COD9 2A 02 C0 LD HL, ($C002)
.; CODC CD 0E C0 CALL $C00E
.; CODF 01 00 08 LD BC, $0800
.; COE2 11 00 08 LD DE, $0800
.; COE5 2A 02 C0 LD HL, ($C002)
.; COE8 CD 0E C0 CALL $C00E
.; COEB 01 00 08 LD BC, $0800
.; COEE 11 00 10 LD DE, $1000
.; COF1 2A 02 C0 LD HL, ($C002)
.; COF4 CD 0E C0 CALL $C00E
.; COF7 3E 3C LD A, $3C
.; COF9 01 00 10 LD BC, $1000
.; COFC 21 00 20 LD HL, $2000
.; COFF CD 08 C0 CALL $C008
.; C102 3E F1 LD A, $F1
.; C104 01 00 08 LD BC, $0800
.; C107 21 00 30 LD HL, $3000
.; C10A CD 08 C0 CALL $C008
.; C10D CD BA 00 CALL $00BA
.; C110 CD 82 C0 CALL $C082
.; C113 C3 0D C1 JP $C10D
.; C116 32 A7 FC LD ($FCA7), A
.

```

De hires mode is de geschiktste scherm-mode voor spelprogrammering. Alle karakters zijn afzonderlijk te definiëren en ook met de kleurmogelijkheden is het prima in orde. Als voorbeeld krijgt u dan ook een spelprobleem. Probleem? Jawel, want bij spelletjes gaat het vooral om snelheid en beeldkwaliteit. Het liefst ook nog eens zoveel mogelijk schermen, u kent dat wel.

We gaan wederom scrollen, maar op een totaal andere manier. Waar het in ons eerste programmeervoorbeeld van dit hoofdstuk om een 'op het scherm'-scroll ging, gaan we nu vanuit het normale geheugen scrollen, en wel zo snel mogelijk. Stelt u zich eens een verticale pijp voor, met een breedte van 32 tekens en een lengte van 256 tekens. Aangezien de toegestane schermhoogte 16 tekens (twee karaktersets) gaan worden, kunnen we onmogelijk het hele speelveld laten zien. Het scherm fungeert als het ware als een venster waardoor we steeds één zestiende van het totale oppervlak kunnen bekijken. (Scrollen met de cursortoetsen)

We nemen ook een horizontale pijp, met een breedte van 256 tekens en een hoogte van 8 tekens. De maximale breedte van een scherm is 32 tekens, dus ook hier moeten we scrollen, alweer met de cursortoetsen.

Door middel van een adresseringstruc worden de schermen uit het BASIC-ROM gekopieerd (zodat u iets ziet als u het de eerste keer probeert), maar het is uiteraard zeer eenvoudig om de bewuste pointers naar RAM te laten wijzen. RAM waar u zelf een mooi scherm kunt maken, met wellicht een zelfgeschreven editor?

## 19.2 De routines op een rij

VFILL vult een stuk videogeheugen met een door u bepaalde waarde. De accumulator bevat deze waarde. Het BC-registerpaar bevat de lengte, het HL-registerpaar het beginadres.

RVMOVE verplaatst een stuk gewoon geheugen naar het videogeheugen. In BC staat de lengte, in DE het beginadres van het videogeheugen en in HL het beginadres van het gewone geheugenblok.

UP scrollt indien mogelijk omhoog.

DOWN scrollt indien mogelijk omlaag.

RIGHT scrollt indien mogelijk naar rechts.

LEFT scrollt indien mogelijk naar links.

SHOWGAME laat een gedeelte van de verticale pijp op het scherm zien; het speelveld zullen we maar zeggen.

Het beginadres wordt met een 'maal 32'-berekening en een eenvoudige offset-optelling naar HL gecalculeerd. Als dit geheugenadres eenmaal is gevonden, volgt een razendsnelle BIOS-kopieerroutine.

SHOWINFO laat een gedeelte van de horizontale pijp op het scherm zien, laten we zeggen het informatieve gedeelte. (Tellers en zo...)

Nu moet u het high-byte en het low-byte even afzonderlijk bekijken. In het low-byte plaatsen we steeds de horizontale begin-pointer. In het high-byte komt het pagina-adres van de lijn die op dat moment aan de beurt is.

Tijdens het naar de buffer kopiëren van een horizontale lijn verhogen we alleen het low-byte (L), voor de volgende lijn verhogen we het high-byte (H).

We kopiëren niet meteen naar het scherm in verband met snelheidsverlies. Een RVMOVE-routine is altijd VEEL sneller dan afzonderlijk 'ge-poke'.

JOY kijkt naar de cursor-toetsen en verandert de scherm-pointers indien noodzakelijk.

INIT zet het scherm in de hires mode en gaat vervolgens de karakterset uit het ROM kopiëren. Eerst naar een buffer, om vervolgens diezelfde karakterset 3 keer naar het videogeheugen weg te zetten.

Met de VFILL-routines zorgen we voor de juiste karakterkleuren op de juiste plaatsen.

# 20 Multi Color

De veel kleuren-mode dus. Een verschrikkelijk ondoorzichtige scherm-mode (zie afb. 20.1), iets dat veel programmeurs afschrikt. Het enige dat u doet, is blokjes op het scherm zetten.  $64 \times 48$  blokjes, zo u wilt met ieder een eigen kleur.

		111111111111111111	
\$0800		0123456789ABCDEF	0123456789ABCDEF
\$0828			
\$0848			
\$0868			
\$0888			
\$08A8			
\$08C8			
\$08E8			
\$0908			
\$0928			
\$0948			
\$0968			
\$0988			
\$09A8			
\$09C8			
\$09E8			
\$0AA00			
\$0A20			
\$0A40			
\$0A60			
\$0A80			
\$0AAC0			
\$0AE8			

\$0000	AAAABBBB	- 8	⟩ (a)	
	CCCCDDDD	- 1	⟩ (b)	
	EEEEEFFF	- 2	⟩ (b)	
	GGGGHHHH	- 3	⟩ (c)	00001111
	IIIIJJJJ	- 4	⟩ (c)	00001111
	KKKKLLLL	- 5	⟩ (d)	00001111
	MMMMNNNN	- 6	⟩ (d)	00001111
	0000OPPP	- 7	⟩ (d)	00001111
\$0008	.....			00001111
	.....			22223333
	.....			22223333
\$0788	AAAABBBB	- 8	⟩ (a)	22223333
	CCCCDDDD	- 1	⟩ (b)	22223333
	EEEEEFFF	- 2	⟩ (b)	22223333
	GGGGHHHH	- 3	⟩ (c)	22223333
	IIIIJJJJ	- 4	⟩ (c)	22223333
	KKKKLLLL	- 5	⟩ (d)	22223333
	MMMMNNNN	- 6	⟩ (d)	22223333
	0000OPPP	- 7	⟩ (d)	22223333
\$0798	.....			.....
	.....			.....
\$07FF	.....			76543210

Afb. 20.1. Screen mode 3

In deze 'veel kleuren'-mode neemt de schermtabel 768 (=  $32 \times 24$ ) bytes in beslag. Het scherm bestaat uit een rooster van 32 bij 24, in het videogeheugen beginnend op \$0800.

De karakterset (beter is: 'kleurensset') begint op \$0000 en loopt tot \$07FF. In deze kleurensset zit 'm de grap. Twee bytes zijn namelijk voldoende om een  $8 \times 8$  blokje in te kleuren. Slim als de MSX-computer is, gebruikt hij de eerste twee bytes uit zo'n (normaal is de karakterlengte 8) kleurenkarakter voor de 0-regels, de tweede twee bytes zijn voor de 1-regels, de derde twee bytes voor de 3-regels en de vierde twee bytes tot slot zijn voor de 4-regels.

Even terug naar dat  $8 \times 8$ -blok. In zo'n  $8 \times 8$ -blok zit in feite een viertal  $4 \times 4$ -pixels ('bixels' in dit geval, want ze zijn een stuk groter) geadresseerd door twee bytes. Het high-nybble van het eerste byte zorgt voor het  $4 \times 4$ -bixel linksboven, het low-nybble van het eerste byte zorgt voor het  $4 \times 4$ -bixel rechtsboven, het high-nybble van het tweede byte zorgt voor het  $4 \times 4$ -bixel linksonder en het low-nybble van het tweede byte zorgt voor het  $4 \times 4$ -bixel rechtsonder.

Sprites zijn toegestaan. De attributentabel begint op \$1B00 en de patroontabel begint op \$3800.

## 20.1 Over lichtorgels

```
10000 ORIGIN ($c000)
10010 --
10020 $004a CONSTANT RdVrm
10030 $004d CONSTANT WrtVrm
10040 $0056 CONSTANT FilVrm
10050 $0062 CONSTANT ChgClr
10060 $0075 CONSTANT IniMlt
10070 $00ba CONSTANT IscNtc
10080 $00d5 CONSTANT GtStck
10090 $00d8 CONSTANT GtTrig
10100 --
10110 $f3e9 CONSTANT ForClr
10120 $f3ea CONSTANT BacClr
10130 $f3eb CONSTANT BdrClr
10140 --
10150 CODE
10160 --
10170 :Teller .by $00
10180 :Xpos .by $00
10190 :Ypos .by $00
10200 :Positie .wo $0000
10210 --
10220 :VPeak
10230 di
```

```
10240      call #RdVrm
10250      ei
10260      ret
10270  --
10280  :VPoke
10290      di
10300      call #WrtVrm
10310      ei
10320      ret
10330  --
10340  :VFill
10350      di
10360      call #FilVrm
10370      ei
10380      ret
10390  --
10400  :CurPos
10410      ld hl,#Xpos
10420      ld a,#0
10430      ld b,a
10440      ld a,(#Ypos)
10450      sla a
10460      rl b
10470      sla a
10480      rl b
10490      sla a
10500      rl b
10510      sla a
10520      rl b
10530      sla a
10540      rl b
10550      or (hl)
10560      ld l,a
10570      ld a,b
10580      ld h,b
10590      ld bc,#$0800
10600      add hl,bc
10610      ld a,#15
10620      jp #VPoke
10630  --
10640  :Plot
10650      ld a,#14
10660      jp #VPoke
10670  --
10680  :Cycle
10690      ld bc,#$0300
10700      ld hl,#$0800
10710  :Cycle0
10720      call #VPeek
10730      cp #0
10740      IF nz
```

```
10750      THEN
10760          dec a
10770          call #VPoke
10780      FI
10790      inc hl
10800      dec c
10810      jpif nz,#Cycle0
10820      dec b
10830      jpif nz,#Cycle0
10840      ret
10850  --
10860 :Boing
10870     call #CurPos
10880     ld bc,#$0020
10890     ld (#Positie),hl
10900     or a
10910     sbc hl,bc
10920     call #Plot
10930     ld hl,(#Positie)
10940     dec hl
10950     call #Plot
10960     inc hl
10970     inc hl
10980     call #Plot
10990     ld hl,(#Positie)
11000     add hl,bc
11010     jp #Plot
11020  --
11030 :Up
11040     ld a,(#Ypos)
11050     cp #1
11060     retif z
11070     dec a
11080     ld (#Ypos),a
11090     ret
11100  --
11110 :Right
11120     ld a,(#Xpos)
11130     cp #30
11140     retif z
11150     inc a
11160     ld (#Xpos),a
11170     ret
11180  --
11190 :Down
11200     ld a,(#Ypos)
11210     cp #22
11220     retif z
11230     inc a
11240     ld (#Ypos),a
11250     ret
11260  --
```

```
11270 :Left
11280     ld a,(#Xpos)
11290     cp #1
11300     retif z
11310     dec a
11320     ld (#Xpos),a
11330     ret
11340 --
11350 :Joy
11360     ld a,#0
11370     call #GtStck
11380     ld b,a
11390     cp #1
11400     IF z
11410         THEN
11420             call #Up
11430     FI
11440     ld a,b
11450     cp #2
11460     IF z
11470         THEN
11480             call #Up
11490             call #Right
11500     FI
11510     ld a,b
11520     cp #3
11530     IF z
11540         THEN
11550             call #Right
11560     FI
11570     ld a,b
11580     cp #4
11590     IF z
11600         THEN
11610             call #Right
11620             call #Down
11630     FI
11640     ld a,b
11650     cp #5
11660     IF z
11670         THEN
11680             call #Down
11690     FI
11700     ld a,b
11710     cp #6
11720     IF z
11730         THEN
11740             call #Down
11750             call #Left
11760     FI
11770     ld a,b
11780     cp #7
```

```
11790     IF z
11800         THEN
11810             call #Left
11820         FI
11830         ld a,b
11840         cp #8
11850         IF z
11860             THEN
11870                 call #Left
11880                 call #Up
11890             FI
11900             ld a,#0
11910             call #GtTrig
11920             cp #$ff
11930             jpif z,#Boing
11940             ret
11950 --
11960 :Init
11970     call #IniMlt
11980     ld a,#0
11990     ld (#Teller),a
12000     ld hl,$0000
12010 :Init0
12020     ld a,(#Teller)
12030     ld b,a
12040     sla b
12050     sla b
12060     sla b
12070     sla b
12080     or b
12090     ld b,#8
12100 :Init1
12110     call #VPoke
12120     inc hl
12130     djnz #Init1
12140     ld a,(#Teller)
12150     inc a
12160     ld (#Teller),a
12170     cp #16
12180     jpif nz,#Init0
12190     ld a,#0
12200     ld bc,$0300
12210     ld hl,$0800
12220     call #VFill
12230     ld a,#1
12240     ld (#ForClr),a
12250     ld (#BacClr),a
12260     ld (#BdrClr),a
12270     ld (#Xpos),a
12280     ld (#Ypos),a
12290     call #ChgClr
12300 :Init2
```

```

12310    call #IscNtc
12320    call #CurPos
12330    call #Cycle
12340    call #Joy
12350    jp #Init2
12360 --
12370 ENDCODE

```

..: C000 00 00 00 00	"...."	..: COA0 3C 32 02 C0	"<2.."
..: C004 00 F3 CD 4A	".J."	..: COA4 C9 3A 01 C0	".::."
..: C008 00 FB C9 F3	"...."	..: COA8 FE 01 C8 3D	".::="
..: CO0C CD 4D 00 FB	".M.."	..: COAC 32 01 C0 C9	"2.."
..: CO10 C9 F3 CD 56	".V"	..: COBO 3E 00 CD D5	
..: CO14 00 FB C9 21	".!."	..: COB4 00 47 FE 01	".G.."
..: CO18 01 C0 3E 00	".>."	..: COB8 C2 BE C0 CD	".::."
..: CO1C 47 3A 02 C0	"G:.."	..: COBC 84 C0 78 FE	".x.."
..: CO20 CB 27 CB 10	"'.."	..: COCO 02 C2 CA C0	".::."
..: CO24 CB 27 CB 10	"'.."	..: COC4 CD 84 C0 CD	".::."
..: CO28 CB 27 CB 10	"'.."	..: COC8 8F C0 78 FE	".x.."
..: CO2C CB 27 CB 10	"'.."	..: COCC 03 C2 D3 C0	".::."
..: CO30 CB 27 CB 10	"'.."	..: COD0 CD 8F C0 78	".::x.."
..: CO34 B6 6F 78 60	".ox"	..: COD4 FE 04 C2 DF	".::."
..: CO38 01 00 08 09	"...."	..: COD8 C0 CD 8F C0	".::."
..: CO3C 3E 0F C3 0B	..: CODC CD 9A C0 78	".::x.."	
..: CO40 C0 3E 0E C3	".>.."	..: COE0 FE 05 C2 E8	".::."
..: CO44 0B C0 01 00	"...."	..: COE4 C0 CD 9A C0	".::."
..: CO48 03 21 00 08	".!.."	..: COE8 78 FE 06 C2	".x.."
..: CO4C CD 05 C0 FE	"...."	..: COEC F4 C0 CD 9A	".::."
..: CO50 00 CA 58 C0	".X.."	..: COFO C0 CD A5 C0	".::."
..: CO54 3D CD 0B C0	"=..."	..: COF4 78 FE 07 C2	"x.."
..: CO58 23 OD C2 4C	"#.L"	..: COF8 FD C0 CD A5	".::."
..: CO5C C0 05 C2 4C	"...L"	..: COFC C0 78 FE 08	".x.."
..: CO60 C0 C9 CD 17	"...."	..: C100 C2 09 C1 CD	".::."
..: CO64 C0 01 20 00	"...."	..: C104 A5 C0 CD 84	".::."
..: CO68 22 03 C0 B7	"...."	..: C108 C0 3E 00 CD	".>.."
..: CO6C ED 42 CD 41	".B.A"	..: C10C D8 00 FE FF	".::."
..: CO70 C0 2A 03 C0	".*.."	..: C110 CA 62 C0 C9	".b.."
..: CO74 2B CD 41 C0	".+A."	..: C114 CD 75 00 3E	".u.>"
..: CO78 23 23 CD 41	"##.A"	..: C118 00 32 00 C0	".2.."
..: CO7C C0 2A 03 C0	".*.."	..: C11C 21 00 00 3A	".!..:"
..: CO80 09 C3 41 C0	".A."	..: C120 00 C0 47 CB	".G.."
..: CO84 3A 02 C0 FE	"...."	..: C124 20 CB 20 CB	".::."
..: CO88 01 C8 3D 32	".=2"	..: C128 20 CB 20 B0	".::."
..: CO8C 02 C0 C9 3A	"...."	..: C12C 06 08 CD OB	".::."
..: CO90 01 C0 FE 1E	"...."	..: C130 C0 23 10 FA	".#.."
..: CO94 C8 3C 32 01	".<2."	..: C134 3A 00 C0 3C	".::<"
..: CO98 C0 C9 3A 02	"...."	..: C138 32 00 C0 FE	"2.."
..: CO9C C0 FE 16 C8	"...."	..: C13C 10 C2 1F C1	".::."

```

.: C140 3E 00 01 00 ">..." .: C158 C0 32 02 C0 ".2.."
.: C144 03 21 00 08 ".!.." .: C15C CD 62 00 CD ".b.."
.: C148 CD 11 C0 3E "...>" .: C160 BA 00 CD 17 "....."
.: C14C 01 32 E9 F3 ".2.." .: C164 C0 CD 46 C0 ".F.."
.: C150 32 EA F3 32 "2..2" .: C168 CD B0 C0 C3 "....."
.: C154 EB F3 32 01 "...2.." .: C16C 5F C1 44 C0 "._.D.."

```

```

.; C005 F3 DI
.; C006 CD 4A 00 CALL $004A
.; C009 FB EI
.; C00A C9 RET
.; C00B F3 DI
.; C00C CD 4D 00 CALL $004D
.; C00F FB EI
.; C010 C9 RET
.; C011 F3 DI
.; C012 CD 56 00 CALL $0056
.; C015 FB EI
.; C016 C9 RET
.; C017 21 01 C0 LD HL,$C001
.; C01A 3E 00 LD A,$00
.; C01C 47 LD B,A
.; C01D 3A 02 C0 LD A,($C002)
.; C020 CB 27 SLA A
.; C022 CB 10 RL B
.; C024 CB 27 SLA A
.; C026 CB 10 RL B
.; C028 CB 27 SLA A
.; C02A CB 10 RL B
.; C02C CB 27 SLA A
.; C02E CB 10 RL B
.; C030 CB 27 SLA A
.; C032 CB 10 RL B
.; C034 B6 OR (HL)
.; C035 6F LD L,A
.; C036 78 LD A,B
.; C037 60 LD H,B
.; C038 01 00 08 LD BC,$0800
.; C03B 09 ADD HL,BC
.; C03C 3E 0F LD A,$0F
.; C03E C3 0B C0 JP $C00B
.; C041 3E 0E LD A,$0E
.; C043 C3 0B C0 JP $C00B
.; C046 01 00 03 LD BC,$0300
.; C049 21 00 08 LD HL,$0800
.; C04C CD 05 C0 CALL $C005
.; C04F FE 00 CP $00
.; C051 CA 58 C0 JP Z,$C058
.; C054 3D DEC A

```

.; C055 CD 0B CO CALL \$C00B  
.; C058 23 INC HL  
.; C059 0D DEC C  
.; C05A C2 4C CO JP NZ,\$C04C  
.; C05D 05 DEC B  
.; C05E C2 4C CO JP NZ,\$C04C  
.; C061 C9 RET  
.; C062 CD 17 CO CALL \$C017  
.; C065 01 20 00 LD BC,\$0020  
.; C068 22 03 CO LD (\$C003),HL  
.; C06B B7 OR A  
.; C06C ED 42 SBC HL,BC  
.; C06E CD 41 CO CALL \$C041  
.; C071 2A 03 CO LD HL,(\$C003)  
.; C074 2B DEC HL  
.; C075 CD 41 CO CALL \$C041  
.; C078 23 INC HL  
.; C079 23 INC HL  
.; C07A CD 41 CO CALL \$C041  
.; C07D 2A 03 CO LD HL,(\$C003)  
.; C080 09 ADD HL,BC  
.; C081 C3 41 CO JP \$C041  
.; C084 3A 02 CO LD A,(\$C002)  
.; C087 FE 01 CP \$01  
.; C089 C8 RET Z  
.; C08A 3D DEC A  
.; C08B 32 02 CO LD (\$C002),A  
.; C08E C9 RET  
.; C08F 3A 01 CO LD A,(\$C001)  
.; C092 FE 1E CP \$1E  
.; C094 C8 RET Z  
.; C095 3C INC A  
.; C096 32 01 CO LD (\$C001),A  
.; C099 C9 RET  
.; C09A 3A 02 CO LD A,(\$C002)  
.; C09D FE 16 CP \$16  
.; C09F C8 RET Z  
.; COA0 3C INC A  
.; COA1 32 02 CO LD (\$C002),A  
.; COA4 C9 RET  
.; COA5 3A 01 CO LD A,(\$C001)  
.; COA8 FE 01 CP \$01  
.; COAA C8 RET Z  
.; COAB 3D DEC A  
.; COAC 32 01 CO LD (\$C001),A  
.; COAF C9 RET  
.; COBO 3E 00 LD A,\$00  
.; COB2 CD D5 00 CALL \$00D5  
.; COB5 47 LD B,A  
.; COB6 FE 01 CP \$01  
.; COB8 C2 BE CO JP NZ,\$C0BE  
.; COBB CD 84 CO CALL \$C084

; COBE	78	LD A,B
; COBF	FE 02	CP \$02
; COC1	C2 CA C0	JP NZ,\$COCA
; COC4	CD 84 C0	CALL \$C084
; COC7	CD 8F C0	CALL \$C08F
; COCA	78	LD A,B
; COCB	FE 03	CP \$03
; COCD	C2 D3 C0	JP NZ,\$C0D3
; COD0	CD 8F C0	CALL \$C08F
; COD3	78	LD A,B
; COD4	FE 04	CP \$04
; COD6	C2 DF C0	JP NZ,\$C0DF
; COD9	CD 8F C0	CALL \$C08F
; CODC	CD 9A C0	CALL \$C09A
; CODF	78	LD A,B
; COE0	FE 05	CP \$05
; COE2	C2 E8 C0	JP NZ,\$COE8
; COE5	CD 9A C0	CALL \$C09A
; COE8	78	LD A,B
; COE9	FE 06	CP \$06
; COEB	C2 F4 C0	JP NZ,\$C0F4
; COEE	CD 9A C0	CALL \$C09A
; COF1	CD A5 C0	CALL \$COA5
; COF4	78	LD A,B
; COF5	FE 07	CP \$07
; COF7	C2 FD C0	JP NZ,\$COFD
; COFA	CD A5 C0	CALL \$COA5
; COFD	78	LD A,B
; COFE	FE 08	CP \$08
; C100	C2 09 C1	JP NZ,\$C109
; C103	CD A5 C0	CALL \$COA5
; C106	CD 84 C0	CALL \$C084
; C109	3E 00	LD A,\$00
; C10B	CD D8 00	CALL \$00D8
; C10E	FE FF	CP \$FF
; C110	CA 62 C0	JP Z,\$C062
; C113	C9	RET
; C114	CD 75 00	CALL \$0075
; C117	3E 00	LD A,\$00
; C119	32 00 C0	LD (\$C000),A
; C11C	21 00 00	LD HL,\$0000
; C11F	3A 00 C0	LD A,(\$C000)
; C122	47	LD B,A
; C123	CB 20	SLA B
; C125	CB 20	SLA B
; C127	CB 20	SLA B
; C129	CB 20	SLA B
; C12B	B0	OR B
; C12C	06 08	LD B,\$08
; C12E	CD 0B C0	CALL \$CO0B
; C131	23	INC HL
; C132	10 FA	DJNZ @C12E

```

.; C134 3A 00 C0      LD A,($C000)
.; C137 3C      INC A
.; C138 32 00 C0      LD ($C000),A
.; C13B FE 10      CP $10
.; C13D C2 1F C1      JP NZ,$C11F
.; C140 3E 00      LD A,$00
.; C142 01 00 03      LD BC,$0300
.; C145 21 00 08      LD HL,$0800
.; C148 CD 11 C0      CALL $C011
.; C14B 3E 01      LD A,$01
.; C14D 32 E9 F3      LD ($F3E9),A
.; C150 32 EA F3      LD ($F3EA),A
.; C153 32 EB F3      LD ($F3EB),A
.; C156 32 01 C0      LD ($C001),A
.; C159 32 02 C0      LD ($C002),A
.; C15C CD 62 00      CALL $0062
.; C15F CD BA 00      CALL $00BA
.; C162 CD 17 C0      CALL $C017
.; C165 CD 46 C0      CALL $C046
.; C168 CD B0 C0      CALL $C0B0
.; C16B C3 5F C1      JP $C15F
.; C16E 44      LD B,H
.

```

Deze veel kleuren-mode gilt om een veel kleuren-voorbeeld. Helaas heeft de MSX1 maar 15 verschillende kleuren, dus daar moeten we het dan maar mee doen. Zwart als achtergrond knalt altijd. Blijven er nog veertien kleuren over voor de effecten. We gaan namelijk een lichtorgel maken.

Het principe is heel simpel: De kleuren verlagen tot transparant is bereikt. We maken 16 verschillende karakters, met zwart als achtergrond en als karakter 0. Als er een groter dan 0-karakter (dus een kleur) op het scherm staat, verlagen we deze met één.

In de tussentijd kunt u met de cursortoetsen aan het wandelen slaan, waarbij de spatiebalk voor een effect zorgt. Zien is geloven. Bekijk dit effect ook eens in een volledig verduisterde kamer...

## 20.2 De routines op een rij

VPEEK, het video-adres in HL, en na terugkeer staat de data in de accumulator.

VPOKE, het video-adres in HL, de data in de accumulator.

**VFILL** vult een stuk videogeheugen met een door u bepaalde waarde. De data in de accumulator, de lengte in BC en het beginadres van het videogeheugen in HL.

**CURPOS** vindt een 16 bits-adres dat bij de X- en de Y-waarde van de cursor hoort. Het adres is het brandpunt van de routine.

**PLOT** plaatst een brandpunt min 1-waarde op het scherm.

**CYCLE** is de routine die het scherm afzoekt op waarden groter dan nul. Zo ja, dan wordt de waarde verlaagd en teruggeplaatst.

**BOING** verzorgt het effect. Uitgaande van het brandpunt wordt rondge-'**PLOT**'.

**UP** verplaatst het brandpunt indien mogelijk naar boven.

**RIGHT** verplaatst het brandpunt indien mogelijk naar rechts.

**DOWN** verplaatst het brandpunt indien mogelijk naar beneden.

**LEFT** verplaatst het brandpunt indien mogelijk naar links.

**JOY** leest de joystick (cursortoetsen) uit en onderneemt actie. Tot slot wordt naar de vuurknop gekeken voor een eventuele **BOING**.

**INIT** initialiseert het scherm naar 'veel kleuren'-mode. Er worden zestien kleuren-karakters aangemaakt, van transparant tot en met wit. Het scherm wordt gewist, nieuwe schermkleuren worden ingesteld en de hoofdroutine treedt in werking.

## 21 Sprites dus

De videochip is in staat om 32 verschillende sprites tegelijkertijd op het beeld te plaatsen.. Sprites staan altijd vóór de achtergrond en er kunnen er maar 4 tegelijk naast elkaar staan. Elke volgende verdwijnt automatisch.

Om de sprites aan te sturen, bestaat er een speciale sprite-attributentabel (zie afb. 21.1) die vijf ingangen bevat.

Eerst de verticale- of Y-positie waarin een waarde tussen 0 en 255 de hoogte op het scherm aangeeft.

Ten tweede de horizontale- of X-positie, wederom een waarde tussen 0 en 255 voor de breedte op het scherm.

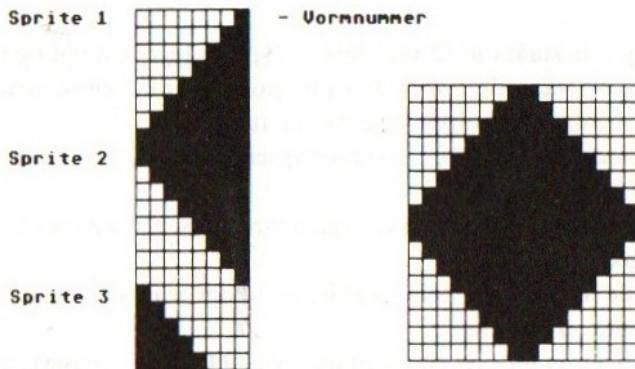
Ten derde een wijzer naar de patroontabel. Vanaf het geadresseerde videogeheugen wordt de sprite-vorm opgehaald. (Zie figuur.)

Het vierde byte zorgt voor de sprite-kleur (0-15) EN een zogenaamd early clock bit. Dit (zevende) bit zorgt ervoor dat de sprite 32 extra naar links wordt gezet.

Sprite 0	V-positie	\$1B00		\$3800 \$3801 \$3802 \$3803 \$3804 \$3805 \$3806 \$3807	Vorm 0
	X-positie	\$1B01			
	Vormnummer	\$1B02			
Sprite 1	EC / Kleurcode	\$1B03			
	V-positie	\$1B04		...	
	X-positie	\$1B05		...	
	Vormnummer	\$1B06		\$3C00 \$3C01 \$3C02 \$3C03 \$3C04 \$3C05 \$3C06 \$3C07	Vorm 128
	EC / Kleurcode	\$1B07			
	.	.			
	.	.			
	.	.			
Sprite 31	V-positie	\$1B7C		\$3FF8 \$3FF9 \$3FFA \$3FFB \$3FFC \$3FFD \$3FFE \$3FFF	Vorm 255
	X-positie	\$1B7D			
	Vormnummer	\$1B7E			
	EC / Kleurcode	\$1B7F			

Afb. 21-1. Sprites op het beeld

In de VDP-registers bevinden zich ook nog een aantal speciale sprite-bits. In VDP-register 1 zorgt bit 0 voor de sprite-grootte: 0 = normaal / 1 = vergroot. In hetzelfde register zorgt bit 1 voor de sprite-maat. Zie de betreffende figuur in verband met het nu te gebruiken vormnummer. (0 =  $8 \times 8$  / 1 =  $16 \times 16$ )



Afb. 21.2. Sprites 'uitvergroot'

## 21.1 Sprites ontwerpen

```
10000 ORIGIN ($c000)
10010 --
10020 $0047 CONSTANT WrtVDP
10030 $004a CONSTANT RdVrm
10040 $004d CONSTANT WrtVrm
10050 $0056 CONSTANT FilVrm
10060 $005c CONSTANT LDirVM
10070 $006f CONSTANT Init32
10080 $009f CONSTANT ChGet
10090 $00a2 CONSTANT ChPut
10100 $00ba CONSTANT IscNtc
10110 $0156 CONSTANT KilBuf
```

```
10120 --
10130 $f3af CONSTANT Lin32
10140 $f3dd CONSTANT CrsX
10150 $f3dc CONSTANT CrsY
10160 $f3e0 CONSTANT Rg1Sav
10170 $fca7 CONSTANT EscCnt
10180 --
10190 CODE
10200 --
10210 :Help      .by $00
10220 :Teller    .by $00
10230 :Positie   .wo $0000
10240 --
10250 :Stuff     .by 10,140,0,0
10260           .by 10,160,0,1
10270           .by 10,180,0,2
10280           .by 10,200,0,3
10290           .by 30,140,0,4
10300           .by 30,160,0,5
10310           .by 30,180,0,6
10320           .by 30,200,0,7
10330           .by 50,140,0,8
10340           .by 50,160,0,9
10350           .by 50,180,0,10
10360           .by 50,200,0,11
10370           .by 70,140,0,12
10380           .by 70,160,0,13
10390           .by 70,180,0,14
10400           .by 70,200,0,15
10410 --
10420 :VPeek
10430   di
10440   call #RdVrm
10450   ei
10460   ret
10470 --
10480 :VPoke
10490   di
10500   call #WrtVrm
10510   ei
10520   ret
10530 --
10540 :VFill
10550   di
10560   call #FilVrm
10570   ei
10580   ret
10590 --
10600 :RVMove
10610   di
10620   call #LDirVM
10630   ei
```

```
10640     ret
10650   --
10660 :GetPos
10670     ld a,#0
10680     ld b,a
10690     ld a,(#Teller)
10700     ld c,a
10710     sla c
10720     rl b
10730     sla c
10740     rl b
10750     sla c
10760     rl b
10770     sla c
10780     rl b
10790     sla c
10800     rl b
10810     ld hl,#$3800
10820     add hl,bc
10830     ld (#Positie),hl
10840     ret
10850   --
10860 :Hight
10870     ld a,(#CrsY)
10880 :Hight0
10890     cp #1
10900     retif z
10910     ld de,(#Positie)
10920     inc de
10930     ld (#Positie),de
10940     dec a
10950     jp #Hight0
10960   --
10970 :Width
10980     ld a,(#CrsX)
10990     cp #9
11000     retif cs
11010     ld hl,(#Positie)
11020     ld de,#$0010
11030     add hl,de
11040     ld (#Positie),hl
11050     ret
11060   --
11070 :GetLin
11080     call #Hight
11090     jp #Width
11100   --
11110 :GetCol
11120     ld a,(#CrsX)
11130     dec a
11140     and #7
11150     ld b,#128
```

```
11160 :GetCol0
11170     cp #0
11180     retif z
11190     srl b
11200     dec a
11210     jp #GetCol0
11220 --
11230 :Part
11240     ld d,#8
11250 :Part0
11260     ld e,#8
11270     push hl
11280     ld hl,(#Positie)
11290     call #VPeek
11300     inc hl
11310     ld (#Positie),hl
11320     pop hl
11330     ld b,#128
11340 :Part1
11350     ld (#Help),a
11360     and b
11370     IF z
11380         THEN
11390             ld a,#196
11400         ELSE
11410             ld a,#219
11420     FI
11430     call #VPoke
11440     inc hl
11450     srl b
11460     ld a,(#Help)
11470     dec e
11480     jpif nz,Part1
11490     ld bc,#$0018
11500     add hl,bc
11510     dec d
11520     jpif nz,#Part0
11530     ret
11540 --
11550 :Setup
11560     call #GetPos
11570     ld hl,$1800
11580     call #Part
11590     ld hl,$1900
11600     call #Part
11610     ld hl,$1808
11620     call #Part
11630     ld hl,$1908
11640     call #Part
11650     ret
11660 --
11670 :Plus
```

```
11680      ld hl,#Teller
11690      inc (hl)
11700 :MakeOk
11710      ld a,(hl)
11720      and #63
11730      ld (hl),a
11740      sla a
11750      sla a
11760      ld b,#16
11770      ld hl,$1b02
11780 :MakeOk0
11790      call #VPoke
11800      inc hl
11810      inc hl
11820      inc hl
11830      inc hl
11840      djnz #MakeOk0
11850      ret
11860 --
11870 :Min
11880      ld hl,#Teller
11890      dec (hl)
11900      jp #MakeOk
11910 --
11920 :Up
11930      ld a,(#CrsY)
11940      cp #1
11950      retif z
11960      ld a,#$1e
11970      jp #ChPut
11980 --
11990 :Down
12000      ld a,(#CrsY)
12010      cp #16
12020      retif z
12030      ld a,#$1f
12040      jp #ChPut
12050 --
12060 :Left
12070      ld a,(#CrsX)
12080      cp #1
12090      retif z
12100      ld a,#$1d
12110      jp #ChPut
12120 --
12130 :Right
12140      ld a,(#CrsX)
12150      cp #16
12160      retif z
12170      ld a,#$1c
12180      jp #ChPut
12190 --
```

```
12200 :Home
12210     call #GetPos
12220     ld a,#0
12230     ld bc,#$0020
12240     ld hl,(#Positie)
12250     jp #VFill
12260 --
12270 :Invert
12280     call #GetPos
12290     ld hl,(#Positie)
12300     ld b,#32
12310 :Invert0
12320     call #VPeek
12330     xor #255
12340     call #VPoke
12350     inc hl
12360     djnz #Invert0
12370     ret
12380 --
12390 :Pixel
12400     call #GetPos
12410     call #GetLin
12420     call #GetCol
12430     ld hl,(#Positie)
12440     call #VPeek
12450     xor b
12460     jp #VPoke
12470 --
12480 :JmpTab    .wo #Plus
12490           .wo #Min
12500           .wo #Up
12510           .wo #Down
12520           .wo #Left
12530           .wo #Right
12540           .wo #Home
12550           .wo #Invert
12560           .wo #Pixel
12570 --
12580 :KeyTab    .by 43
12590           .by 45
12600           .by $1e
12610           .by $1f
12620           .by $1d
12630           .by $1c
12640           .by $0b
12650           .by 120
12660           .by 32
12670           .by 0
12680 --
12690 :SelfMod
12700     jp #SelfMod
12710 --
```

```
12720 :Sprites
12730     ld bc,#$0040
12740     ld de,#$1b00
12750     ld hl,#Stuff
12760     call #RVMove
12770     ld a,(#Rg1Sav)
12780     or #2
12790     ld b,a
12800     ld c,#1
12810     call #WrtVDP
12820     ret
12830 --
12840 :Init
12850     ld a,#32
12860     ld (#Lin32),a
12870     call Init32
12880     ld a,#1
12890     ld (#CrsX),a
12900     ld (#CrsY),a
12910     call #Sprites
12920 :Init0
12930     call #Setup
12940     call #KilBuf
12950 :Init1
12960     call #IscNtc
12970     ld a,#0
12980     ld (#EscCnt),a
12990     call #ChGet
13000     ld (#Help),a
13010     ld hl,#KeyTab
13020     ld de,#JmpTab
13030 :Init2
13040     ld a,(hl)
13050     cp #0
13060     jpif z,#Init1
13070     ld a,(#Help)
13080     cp (hl)
13090     IF z THEN
13100         ld a,(de)
13110         ld (#SelfMod+1),a
13120         inc de
13130         ld a,(de)
13140         ld (#SelfMod+2),a
13150         call #SelfMod
13160         jp #Init0
13170     FI
13180     inc hl
13190     inc de
13200     inc de
13210     jp #Init2
13220 --
13230 ENDCODE
```

.. C000 00 00 00 00	"...."	.. COCC 32 00 C0 A0	"2..."
.. C004 0A 8C 00 00	"...."	.. COD0 C2 D8 C0 3E	"...>"
.. C008 0A A0 00 01	"...."	.. COD4 C4 C3 DA C0	"...."
.. COOC 0A B4 00 02	"...."	.. COD8 3E DB CD 4A	">..J"
.. CO10 0A C8 00 03	"...."	.. CODC C0 23 CB 38	".#.8"
.. CO14 1E 8C 00 04	"...."	.. COE0 3A 00 C0 1D	"...."
.. CO18 1E A0 00 05	"...."	.. COE4 C2 CC C0 01	"...."
.. CO1C 1E B4 00 06	"...."	.. COE8 18 00 09 15	"...."
.. CO20 1E C8 00 07	"...."	.. COEC C2 BC C0 C9	"...."
.. CO24 32 8C 00 08	"2..."	.. COFO CD 5C C0 21	"..\!."
.. CO28 32 A0 00 09	"2..."	.. COF4 00 18 CD BA	"...."
.. CO2C 32 B4 00 0A	"2..."	.. COF8 C0 21 00 19	"..\!."
.. CO30 32 C8 00 0B	"2..."	.. COFC CD BA C0 21	"....!"
.. CO34 46 8C 00 OC	"F..."	.. C100 08 18 CD BA	"...."
.. CO38 46 A0 00 OD	"F..."	.. C104 C0 21 08 19	"..\!."
.. CO3C 46 B4 00 OE	"F..."	.. C108 C3 BA C0 21	"....!"
.. CO40 46 C8 00 OF	"F..."	.. C10C 01 C0 34 7E	".4."
.. CO44 F3 CD 4A 00	"..J."	.. C110 E6 3F 77 CB	".?w."
.. CO48 FB C9 F3 CD	"...."	.. C114 27 CB 27 06	"',."
.. CO4C 4D 00 FB C9	"M..."	.. C118 10 21 02 1B	"..\!."
.. CO50 F3 CD 56 00	"..V."	.. C11C CD 4A C0 23	".J.#"
.. CO54 FB C9 F3 CD	"...."	.. C120 23 23 23 10	"###."
.. CO58 5C 00 FB C9	"\..."	.. C124 F7 C9 21 01	"..\!."
.. CO5C 3E 00 47 3A	">.G:"	.. C128 C0 35 C3 OF	".5.."
.. CO60 01 C0 4F CB	"..O."	.. C12C C1 3A DC F3	"...."
.. CO64 21 CB 10 CB	"!..."	.. C130 FE 01 C8 3E	"...>"
.. CO68 21 CB 10 CB	"!..."	.. C134 1E C3 A2 00	"...."
.. CO6C 21 CB 10 CB	"!..."	.. C138 3A DC F3 FE	"...."
.. CO70 21 CB 10 CB	"!..."	.. C13C 10 C8 3E 1F	"...>."
.. CO74 21 CB 10 21	"!..!"	.. C140 C3 A2 00 3A	"...;"
.. CO78 00 38 09 22	".8.**"	.. C144 DD F3 FE 01	"...."
.. CO7C 02 C0 C9 3A	"...."	.. C148 C8 3E 1D C3	".>.."
.. CO80 DC F3 FE 01	"...."	.. C14C A2 00 3A DD	"...."
.. CO84 C8 ED 5B 02	"..[."	.. C150 F3 FE 10 C8	"...."
.. CO88 C0 13 ED 53	"...S"	.. C154 3E 1C C3 A2	
.. CO8C 02 C0 3D C3	"..=."	.. C158 00 CD 5C C0	"..\!."
.. CO90 82 C0 3A DD	"...:."	.. C15C 3E 00 01 20	".>.."
.. CO94 F3 FE 09 D8	"...."	.. C160 00 2A 02 C0	".*.."
.. CO98 2A 02 C0 11	"*..."	.. C164 C3 50 C0 CD	".P.."
.. CO9C 10 00 19 22	"...."	.. C168 5C C0 2A 02	"..\*."
.. COA0 02 C0 C9 CD	"...."	.. C16C C0 06 20 CD	"...;"
.. COA4 7F C0 C3 92	"...."	.. C170 44 C0 EE FF	"D..."
.. COA8 C0 3A DD F3	"...."	.. C174 CD 4A C0 23	".J.#"
.. COAC 3D E6 07 06	"=..."	.. C178 10 F5 C9 CD	"...."
.. COB0 80 FE 00 C8	"...."	.. C17C 5C C0 CD A3	"\..."
.. COB4 CB 38 3D C3	".8=."	.. C180 C0 CD A9 C0	"...."
.. COB8 B1 C0 16 08	"...."	.. C184 2A 02 C0 CD	"*..."
.. COBC 1E 08 E5 2A	"...*"	.. C188 44 C0 A8 C3	"D..."
.. COCO 02 C0 CD 44	"...D"	.. C18C 4A C0 0B C1	"J..."
.. COC4 C0 23 22 02	".#".."	.. C190 26 C1 2D C1	".&.-."
.. COC8 C0 E1 06 80	"...."	.. C194 38 C1 43 C1	"8.C."

.: C198	4E	C1	59	C1	"N.Y."	.: C1D8	F0	C0	CD	56	"...V"
.: C19C	67	C1	7B	C1	"g.{."	.: C1DC	01	CD	BA	00	"...."
.: C1A0	2B	2D	1E	1F	"+-.."	.: C1E0	3E	00	32	A7	
.: C1A4	1D	1C	0B	78	"...x"	.: C1E4	FC	CD	9F	00	"...."
.: C1A8	20	00	C3	AA	"..."	.: C1E8	32	00	C0	21	"2...!"
.: C1AC	C1	01	40	00	"..@."	.: C1EC	A0	C1	11	8E	"...."
.: C1B0	11	00	1B	21	"...!"	.: C1FO	C1	7E	FE	00	"...."
.: C1B4	04	C0	CD	56	"...V"	.: C1F4	CA	DD	C1	3A	"...."
.: C1B8	C0	3A	E0	F3	"...:."	.: C1F8	00	C0	BE	C2	"...."
.: C1BC	F6	02	47	OE	"..G."	.: C1FC	0D	C2	1A	32	"...2"
.: C1C0	01	C3	47	00	"..G."	.: C200	AB	C1	13	1A	"...."
.: C1C4	3E	20	32	AF	"> 2."	.: C204	32	AC	C1	CD	"2..."
.: C1C8	F3	CD	6F	00	"..o."	.: C208	AA	C1	C3	D7	"...."
.: C1CC	3E	01	32	DD	">.2."	.: C20C	C1	23	13	13	".#.."
.: C1D0	F3	32	DC	F3	"..2.."	.: C210	C3	F1	C1	00	"...."
.: C1D4	CD	AD	C1	CD	"...."	.	.	.	.	.	.

.: C044	F3	DI		
.: C045	CD	4A	00	CALL \$004A
.: C048	FB	EI		
.: C049	C9	RET		
.: C04A	F3	DI		
.: C04B	CD	4D	00	CALL \$004D
.: C04E	FB	EI		
.: C04F	C9	RET		
.: C050	F3	DI		
.: C051	CD	56	00	CALL \$0056
.: C054	FB	EI		
.: C055	C9	RET		
.: C056	F3	DI		
.: C057	CD	5C	00	CALL \$005C
.: C05A	FB	EI		
.: C05B	C9	RET		
.: C05C	3E	00	LD A,\$00	
.: C05E	47	LD B,A		
.: C05F	3A	01	C0	LD A,(\$C001)
.: C062	4F	LD C,A		
.: C063	CB	21	SLA C	
.: C065	CB	10	RL B	
.: C067	CB	21	SLA C	
.: C069	CB	10	RL B	
.: C06B	CB	21	SLA C	
.: C06D	CB	10	RL B	
.: C06F	CB	21	SLA C	
.: C071	CB	10	RL B	
.: C073	CB	21	SLA C	
.: C075	CB	10	RL B	
.: C077	21	00	38	LD HL,\$3800
.: C07A	09	ADD HL,BC		
.: C07B	22	02	C0	LD (\$C002),HL

```

.; C07E C9          RET
.; C07F 3A DC F3    LD A,($F3DC)
.; C082 FE 01        CP $01
.; C084 C8          RET Z
.; C085 ED 5B 02 CO LD DE,($C002)
.; C089 13          INC DE
.; C08A ED 53 02 CO LD ($C002),DE
.; C08E 3D          DEC A
.; C08F C3 82 CO    JP $C082
.; C092 3A DD F3    LD A,($F3DD)
.; C095 FE 09        CP $09
.; C097 D8          RET C
.; C098 2A 02 CO    LD HL,($C002)
.; C09B 11 10 00    LD DE,$0010
.; C09E 19          ADD HL,DE
.; C09F 22 02 CO    LD ($C002),HL
.; COA2 C9          RET
.; COA3 CD 7F CO    CALL $C07F
.; COA6 C3 92 CO    JP $C092
.; COA9 3A DD F3    LD A,($F3DD)
.; COAC 3D          DEC A
.; COAD E6 07        AND $07
.; COAF 06 80        LD B,$80
.; COB1 FE 00        CP $00
.; COB3 C8          RET Z
.; COB4 CB 38        SRL B
.; COB6 3D          DEC A
.; COB7 C3 B1 CO    JP $COB1
.; COBA 16 08        LD D,$08
.; COBC 1E 08        LD E,$08
.; COBE E5          PUSH HL
.; COBF 2A 02 CO    LD HL,($C002)
.; COC2 CD 44 CO    CALL $CO44
.; COC5 23          INC HL
.; COC6 22 02 CO    LD ($C002),HL
.; COC9 E1          POP HL
.; COCA 06 80        LD B,$80
.; COCC 32 00 CO    LD ($C000),A
.; COCF A0          AND B
.; COD0 C2 D8 CO    JP NZ,$COD8
.; COD3 3E C4        LD A,$C4
.; COD5 C3 DA CO    JP $CODA
.; COD8 3E DB        LD A,$DB
.; CODA CD 4A CO    CALL $CO4A
.; CODD 23          INC HL
.; CODE CB 38        SRL B
.; COEO 3A 00 CO    LD A,($C000)
.; COE3 1D          DEC E
.; COE4 C2 CC CO    JP NZ,$COCC
.; COE7 01 18 00    LD BC,$0018
.; COEA 09          ADD HL,BC
.; COEB 15          DEC D

```

```

.; COEC C2 BC CO      JP NZ,$COBC
.; COEF C9      RET
.; COFO CD 5C CO      CALL $CO5C
.; COF3 21 00 18      LD HL,$1800
.; COF6 CD BA CO      CALL $COBA
.; COF9 21 00 19      LD HL,$1900
.; COFC CD BA CO      CALL $COBA
.; COFF 21 08 18      LD HL,$1808
.; C102 CD BA CO      CALL $COBA
.; C105 21 08 19      LD HL,$1908
.; C108 C3 BA CO      JP $COBA
.; C10B 21 01 CO      LD HL,$C001
.; C10E 34      INC (HL)
.; C10F 7E      LD A,(HL)
.; C110 E6 3F      AND $3F
.; C112 77      LD (HL),A
.; C113 CB 27      SLA A
.; C115 CB 27      SLA A
.; C117 06 10      LD B,$10
.; C119 21 02 1B      LD HL,$1B02
.; C11C CD 4A CO      CALL $CO4A
.; C11F 23      INC HL
.; C120 23      INC HL
.; C121 23      INC HL
.; C122 23      INC HL
.; C123 10 F7      DJNZ @C11C
.; C125 C9      RET
.; C126 21 01 CO      LD HL,$C001
.; C129 35      DEC (HL)
.; C12A C3 OF C1      JP $C10F
.; C12D 3A DC F3      LD A,($F3DC)
.; C130 FE 01      CP $01
.; C132 C8      RET Z
.; C133 3E 1E      LD A,$1E
.; C135 C3 A2 00      JP $00A2
.; C138 3A DC F3      LD A,($F3DC)
.; C13B FE 10      CP $10
.; C13D C8      RET Z
.; C13E 3E 1F      LD A,$1F
.; C140 C3 A2 00      JP $00A2
.; C143 3A DD F3      LD A,($F3DD)
.; C146 FE 01      CP $01
.; C148 C8      RET Z
.; C149 3E 1D      LD A,$1D
.; C14B C3 A2 00      JP $00A2
.; C14E 3A DD F3      LD A,($F3DD)
.; C151 FE 10      CP $10
.; C153 C8      RET Z
.; C154 3E 1C      LD A,$1C
.; C156 C3 A2 00      JP $00A2
.; C159 CD 5C CO      CALL $CO5C
.; C15C 3E 00      LD A,$00

```

; C15E 01 20 00 LD BC,\$0020  
; C161 2A 02 C0 LD HL,(\$C002)  
; C164 C3 50 C0 JP \$C050  
; C167 CD 5C C0 CALL \$C05C  
; C16A 2A 02 C0 LD HL,(\$C002)  
; C16D 06 20 LD B,\$20  
; C16F CD 44 C0 CALL \$C044  
; C172 EE FF XOR \$FF  
; C174 CD 4A C0 CALL \$C04A  
; C177 23 INC HL  
; C178 10 F5 DJNZ @C16F  
; C17A C9 RET  
; C17B CD 5C C0 CALL \$C05C  
; C17E CD A3 C0 CALL \$COA3  
; C181 CD A9 C0 CALL \$COA9  
; C184 2A 02 C0 LD HL,(\$C002)  
; C187 CD 44 C0 CALL \$C044  
; C18A A8 XOR B  
; C18B C3 4A C0 JP \$C04A  
; C18E 0B DEC BC  
; C18F C1 POP BC  
; C190 26 C1 LD H,\$C1  
; C192 2D DEC L  
; C193 C1 POP BC  
; C194 38 C1 JR C,@C157  
; C196 43 LD B,E  
; C197 C1 POP BC  
; C198 4E LD C,(HL)  
; C199 C1 POP BC  
; C19A 59 LD E,C  
; C19B C1 POP BC  
; C19C 67 LD H,A  
; C19D C1 POP BC  
; C19E 7B LD A,E  
; C19F C1 POP BC  
; C1A0 2B DEC HL  
; C1A1 2D DEC L  
; C1A2 1E 1F LD E,\$1F  
; C1A4 1D DEC E  
; C1A5 1C INC E  
; C1A6 0B DEC BC  
; C1A7 78 LD A,B  
; C1A8 20 00 JR NZ,@C1AA  
; C1AA C3 AA C1 JP \$C1AA  
; C1AD 01 40 00 LD BC,\$0040  
; C1B0 11 00 1B LD DE,\$1B00  
; C1B3 21 04 C0 LD HL,\$C004  
; C1B6 CD 56 C0 CALL \$C056  
; C1B9 3A E0 F3 LD A,(\$F3E0)  
; C1BC F6 02 OR \$02  
; C1BE 47 LD B,A

```

.; C1BF 0E 01 LD C,$01
.; C1C1 C3 47 00 JP $0047
.; C1C4 3E 20 LD A,$20
.; C1C6 32 AF F3 LD ($F3AF),A
.; C1C9 CD 6F 00 CALL $006F
.; C1CC 3E 01 LD A,$01
.; C1CE 32 DD F3 LD ($F3DD),A
.; C1D1 32 DC F3 LD ($F3DC),A
.; C1D4 CD AD C1 CALL $C1AD
.; C1D7 CD F0 C0 CALL $COFO
.; C1DA CD 56 01 CALL $0156
.; C1DD CD BA 00 CALL $00BA
.; C1E0 3E 00 LD A,$00
.; C1E2 32 A7 FC LD ($FCA7),A
.; C1E5 CD 9F 00 CALL $009F
.; C1E8 32 00 C0 LD ($C000),A
.; C1EB 21 A0 C1 LD HL,$C1A0
.; C1EE 11 8E C1 LD DE,$C18E
.; C1F1 7E LD A,(HL)
.; C1F2 FE 00 CP $00
.; C1F4 CA DD C1 JP Z,$C1DD
.; C1F7 3A 00 C0 LD A,($C000)
.; C1FA BE CP (HL)
.; C1FB C2 0D C2 JP NZ,$C20D
.; C1FE 1A LD A,(DE)
.; C1FF 32 AB C1 LD ($C1AB),A
.; C202 13 INC DE
.; C203 1A LD A,(DE)
.; C204 32 AC C1 LD ($C1AC),A
.; C207 CD AA C1 CALL $C1AA
.; C20A C3 D7 C1 JP $C1D7
.; C20D 23 INC HL
.; C20E 13 INC DE
.; C20F 13 INC DE
.; C210 C3 F1 C1 JP $C1F1
.

```

Voordat u iets met sprites kunt doen, moet u wel aan de sprite-vormen kunnen komen. Vandaar als voorbeeld een sprite-editor. Ten eerste ziet u hoe u de sprites op beeld krijgt, maar bovenal hoe u  $16 \times 16$ -sprites (Niemand die  $8 \times 8$ -sprites maakt. Of wel soms?) kunt gebruiken.

Vanzelfsprekend kunt u deze zelf ontworpen sprites ook opslaan. Dit gaat vanuit BASIC door middel van een video-SAVE-opdracht. Zo'n weggeschreven spriteset kunt u naderhand met een video-LOAD-commando weer binnenhalen.

De sprite-editor lijkt nogal wat op de karakter-editor. Het zal dus inmiddels voor u geen enkel probleem opleveren om uit te zoeken hoe het programma werkt.

## **21.2 De routines op een rij**

STUFF is een tabel voor alle sprite-benodigdheden. Erg handig omdat u met een RAM/video-kopieerinstructie meteen alle sprites initialiseert.

VPEEK, u weet wel.

VPOKE, dito.

VFILL, u welbekend.

RVMOVE, dezelfde instructie als bij het hires-voorbeeld.

GETPOS berekent de plaats van de huidige sprite-informatie. Dit komt neer op een 'sprite maal 32'-berekening.

HIGHT calculeert een te adresseren patroonpositie in, rekening houdend met de Y-waarde van de cursor.

WIDTH kijkt naar de X-waarde van de cursor om, indien u in het derde of vierde sprite-blokje edit, 16 extra bij te tellen.

GETLIN kijkt welk byte uit het patroon van  $16 \times 2$  bytes thans wordt bewerkt.

GETCOL kijkt welk bit uit het byte thans wordt bewerkt. We kijken 'slim' naar de X-waarde van de cursor.

PART zet een  $8 \times 8$  bit-matrix op het scherm. Te vergelijken met SETUP uit de karakter-editor.

SETUP zet een  $16 \times 16$ -matrix op het scherm, gebruik makend van het feit dat een  $16 \times 16$  sprite uit vier  $8 \times 8$ -blokken bestaat.

PLUS verhoogt de nu bekijken sprite met één. Na afloop wordt voor de 16 voorbeeld-sprites de nieuwe sprite-pointer aangemaakt.

MIN verlaagt de nu bekijken sprite met één.

UP verplaatst indien mogelijk wordt de cursor één locatie naar boven.

DOWN verplaatst indien mogelijk de cursor één locatie naar beneden.

LEFT verplaatst indien mogelijk de cursor één locatie naar links.

RIGHT verplaatst indien mogelijk de cursor één naar rechts.

HOME wist de inhoud van de sprite (alles op 0).

INVERT inverteert het geadresseerde sprite-patroon. Alle 0-bits worden geset en andersom.

PIXEL berekent de pixelpositie van de cursor en XOR-t deze waarde met de desbetreffende bitadressering. Pixel aan wordt pixel uit en andersom.

JMPTAB bevat de adressen die worden aangeroepen als er op een juiste toets wordt gedrukt.

KEYTAB bevat de waarden die horen bij een commandotoets op het toetsenbord. Een KEYTAB-adressering is niet meer dan een pointer naar de vorige tabel.

SELMOD is een CALL-routine die door de key scan wordt veranderd als er op een juiste toets wordt gedrukt.

Stel dat de PLUS-routine moet worden aangeroepen. Het low-byte van PLUS komt van KEYTAB en wordt op SELMOD + 1 geplaatst. Het high-byte van PLUS komt van KEYTAB + 1 en wordt op SELMOD + 2 geplaatst.

SPRITES initialiseert de gebruikte sprites. Ook worden de sprites op  $16 \times 16$ -grootte ingesteld.

INIT stelt de juiste schermbreedte en scherm-mode in, waarna (met CTRL/STOP-check) de juiste commando's worden bepaald enz.

## **De software**

Als ontwikkelings-software bieden we u SPEEDTYPER en HERBYMON aan. Deze machinecodelisting-verwerker en machinecode-monitor kunt u aan de hand van de listings invoeren.

De SPEEDTYPER heeft u nodig om de HERBYMON te kunnen invoeren. Dit is expres zo gedaan: HERBYMON in BASIC-DATA-regels zou ruim 40% groter worden dan een SPEEDTYPER plus HERBYMON. Tevens zorgt de SPEEDTYPER voor een invoercontrole.

De HERBYMON gebruikt u onder andere om de voorbeelden uit dit boek te kunnen proberen en eventueel ook om eigen machinecode te ontwikkelen. De HERBYMON beslaat het geheugengebied \$A000-\$C000 en het startadres is 49059 decimaal. U krijgt ook nog een startprogramma voor de HERBYMON. Als u een diskdrive heeft, zou u dit programma als AUTOEXEC.BAS op de schijf kunnen zetten. Dit heeft tot gevolg dat de monitor automatisch start.

Van beide programma's volgt de handleiding na de listing.

# 22 De SPEEDTYPER

## 22.1 De listing

```
1000 REM ****
1010 REM *** SPEEDTYPER ***
1020 REM ****
1030 :
1040 SCREEN1:WIDTH30
1050 CLEAR200,&H9FFF
1060 CR$=CHR$(13)
1070 LF$=CR$+CHR$(10)
1080 BG$=CR$+CHR$(27)+CHR$(&H4D)
1090 :
1100 CLS
1110 PRINT ****
1120 PRINT "*** SPEEDTYPER ***"
1130 PRINT ****
1140 PRINT
1150 INPUT "BEGIN-ADRES";BA
1160 IFHEX$(BA)<HEX$(&HA000)THEN1150
1170 INPUT " EIND-ADRES";EA
1180 IFHEX$(EA)<HEX$(BA)THEN1170
1190 :
1200 PRINT
1210 PRINT "BEGIN-ADRES - &h";HEX$(BA)
1220 PRINT " EIND-ADRES - &h";HEX$(EA)
1230 PRINT
1240 INPUT "IS DIT OK? (j/n)";JN$
1250 IFJN$<>"j"THENRUN
1260 :
1270 REM ***
1280 :
1290 CLS
1300 PRINT "*** SPEEDTYPER OPTIES ***"
1310 PRINT
1320 PRINT" (1) - 'LEEG POKEN' "
1330 PRINT" (2) - DATA BEKIJKEN"
1340 PRINT" (3) - DATA INVOEREN"
1350 PRINT" (4) - DATA LOADEN"
1360 PRINT" (5) - DATA SAVEN"
1370 PRINT" (6) - STOPPEN"
1380 PRINT
```

```
1390 PRINT"      <MAAK UW KEUZE>"  
1400 IP$=INKEY$  
1410 IFIP$<"1"ORIP$>"6"THEN1400  
1420 CLS  
1430 ONVAL("++"+IP$)GOTO1540,1840,2270,1640,1740,1470  
1440 :  
1450 REM ***  
1460 :  
1470 PRINT"***STOPPEN***"  
1480 INPUT"ZEKER WETEN? (j/n)";JN$  
1490 IFJN$="j"THENRUN  
1500 GOTO1290  
1510 :  
1520 REM ***  
1530 :  
1540 PRINT"***'LEEG POKEN'***"  
1550 INPUT"ZEKER WETEN? (j/n)";JN$  
1560 IFJN$<>"j"THEN1290  
1570 FORT=BATOEA  
1580 : POKE(T),0  
1590 NEXT  
1600 GOTO1290  
1610 :  
1620 REM ***  
1630 :  
1640 PRINT"***DATA LOADEN***"  
1650 INPUT"ZEKER WETEN? (j/n)";JN$  
1660 IFJN$<>"j"THEN1290  
1670 PRINT  
1680 INPUT"FILENAAM: ";N$  
1690 BLOADN$  
1700 GOTO1290  
1710 :  
1720 REM ***  
1730 :  
1740 PRINT"***DATA SAVEN***"  
1750 INPUT"ZEKER WETEN? (j/n)";JN$  
1760 IFJN$<>"j"THEN1290  
1770 PRINT  
1780 INPUT"FILENAAM: ";N$  
1790 BSAVEN$,BA,EA  
1800 GOTO1290  
1810 :  
1820 REM ***  
1830 :  
1840 PRINT"***DATA BEKIJKEN***"  
1850 INPUT"ZEKER WETEN? (j/n)";JN$  
1860 IFJN$<>"j"THEN1290  
1870 PRINT  
1880 INPUT"(p)RINTER / (s)CHERM";DV$  
1890 INPUT"ZEKER WETEN? (j/n)";JN$  
1900 IFJN$<>"j"THEN1890
```

```
1910 PRINT
1920 PRINT "SPATIEBALK=PAUZE"
1930 PRINT " CURSOR=OPTIES!"
1940 PRINT
1950 PRINT " <SPATIEBALK> "
1960 IFSTRIG(0)=OTHEN1960
1970 CLS
1980 NA=BA
1990 DO$="SPEEDDUMP VAN "+HEX$(BA)+" TOT "+HEX$(EA)+LF$
2000 GOSUB 2140
2010 DO$=HEX$(NA)+": "
2020 GOSUB2180:D1=CK
2030 DO$=DO$+
2040 GOSUB2180:D2=CK
2050 DO$=DO$+
2060 CK$=HEX$(D1+D2): IFLEN(CK$)=1THENCK$="00"+CK$
2070 IFLEN(CK$)=2THENCK$="0"+CK$
2080 DO$=DO$+CK$+CR$
2090 GOSUB2140
2100 AB=STICK(0):IFAB<>0THEN1290
2110 IFNA<EATHEN2010
2120 GOTO1290
2130 :
2140 PRINTDO$
2150 IFDV$="p"THENLPRINTDO$
2160 IF STRIG(0)THEN2160
2170 RETURN
2180 CK=0:FORNA=NATONA+3
2190 : AD$=HEX$(PEEK(NA))
2200 : IFLEN(AD$)=1THENAD$="0"+AD$
2210 : DO$=DO$+AD$
2220 : CK=CK+PEEK(NA)
2230 NEXT:RETURN
2240 :
2250 REM ***
2260 :
2270 PRINT"***DATA INVOEREN***"
2280 INPUT"ZEKER WETEN? (j/n)";JN$
2290 IFJN$<>"j"THEN1290
2300 PRINT
2310 INPUT"INVOER-ADRES";TA
2320 IF(TA<BA)OR(TA>EA)THEN2310
2330 PRINT
2340 PRINT" BEGIN-ADRES - &h";HEX$(BA)
2350 PRINT"INVOER-ADRES - &h";HEX$(TA)
2360 PRINT" EIND-ADRES - &h";HEX$(EA)
2370 PRINT
2380 INPUT"IS DIT OK? (j/n)";JNS
2390 IFJN$="n"THEN2300
2400 PRINT" 'X'=OPTIES"
2410 PRINT
2420 PRINT"<SPATIEBALK>"
```

```

2430 IFSTRIG(0)=OTHEN2430
2440 IFSTRIG(0)THEN2440
2450 CLS:I$=INKEY$
2460 NA=TA
2470 DO$=HEX$(NA)+": "
2480 PRINTBG$;DO$;CHR$(95);
2490 I$=INKEY$:IFI$=""THEN2490
2500 AS=ASC(I$)
2510 IFAS=120THEN1290
2520 IF(AS=127)OR(AS=8)THEN2590
2530 IF(AS<48)OR(AS>102)THEN2490
2540 IF(AS>57)AND(AS<96)THEN2490
2550 IFLEN(DO$)=13THENI$=I$+" "
2560 IFLEN(DO$)=22THENI$=I$+" "
2570 DO$=DO$+I$:IFLEN(DO$)=27THEN2640
2580 GOTO2480
2590 IFLEN(DO$)<7THEN2490
2600 DO$=LEFT$(DO$,LEN(DO$)-1)
2610 IFLEN(DO$)=14THENDO$=LEFT$(DO$,13)
2620 IFLEN(DO$)=23THENDO$=LEFT$(DO$,22)
2630 GOTO2480
2640 PRINTBG$;DO$;
2650 P1$=MID$(DO$,7,8)
2660 P2$=MID$(DO$,16,8)
2670 CK$=MID$(DO$,25,3)
2680 PT$=P1$+P2$+" "
2690 CK=0
2700 FORT=OT07
2710 : PB$="&h"+LEFT$(PT$,2)
2720 : PT$=RIGHT$(PT$,LEN(PT$)-2)
2730 : PW(T)=VAL(PB$)
2740 : CK=CK+PW(T)
2750 NEXT
2760 IFCK=(VAL(" "&h"+CK$))THEN2780
2770 PRINT"CHECKSUM ERROR!":GOTO2470
2780 FORT=OT07
2790 : POKENA,PW(T)
2800 : NA=NA+1
2810 NEXT
2820 IFHEX$(NA)<HEX$(EA)THEN2470
2830 GOTO1290

```

## 22.2 Werken met de SPEEDTYPER

Het moet natuurlijk geen enkel probleem hebben opgeleverd om de SPEEDTYPER in te typen. Laten we gemakshalve meteen maar aannemen dat u dit foutloos heeft gedaan.

U start de SPEEDTYPER vanuit BASIC met het RUN-commando. Eerst wordt om een beginadres gevraagd. Dit is in ons geval het beginadres van de HERBYMON, te weten &HA000.

Vervolgens een eindadres. Bij het intypen van de HERBYMON moet dit &HC000 zijn. (&HA000 - &HC000 dus)

Na het J-antwoord komt u in het hoofdmenu terecht. U kunt hier kiezen uit zes opties:

- 1) 'LEEG POKEN'
- 2) DATA BEKIJKEN
- 3) DATA INVOEREN
- 4) DATA LOADEN
- 5) DATA SAVEN
- 6) STOPPEN

Door op de corresponderende cijfertoets te drukken, selecteert u de optie. Vervolgens wordt u door middel van vraag en antwoord op uw wenken bediend.

### **'LEEG POKEN'**

Dit 'leeg poken' doet u eigenlijk maar één keer: Aan het begin van uw marathon-intypsessie.

Het gevolg is dat het toegewezen geheugengebied op 0 wordt gezet. Dit heeft als voordeel dat u later gemakkelijk kunt opzoeken waar u de vorige keer was gebleven.

### **DATA BEKIJKEN**

U kunt nu het toegewezen geheugengebied gaan nakijken. Vanaf het beginadres tot aan het eindadres kunt u het geheugen bekijken in het speciale SPEEDTYPER-formaat, dus met checksum. U kunt de data niet alleen op het scherm bekijken; u kunt de data ook naar de printer sturen. Komt misschien nog wel eens van pas bij een moeilijk te vinden intypfout...

Met de spatiebalk pauzeert u en het indrukken van een cursortoets brengt u weer in het hoofdmenu.

## **DATA INVOEREN**

In feite de belangrijkste optie. U kunt nu aan de hand van een SPEEDTYPER-listing een programma (De HERBYMON dus) invoeren, terwijl de invoer automatisch wordt gecontroleerd. Fouten maken wordt haast onmogelijk, op voorwaarde dat u wel de juiste adressen in de gaten houdt!

Er wordt om een invoeradres gevraagd. Dit is het beginadres van het toegewezen geheugengebied als dit uw allereerste poging wordt. In een ander geval dient het laatst getypte adres als invoeradres, dus het adres waar u de vorige keer bent opgehouden.

Nog even een kleine controle en daar gaat u (met de X-toets komt u weer in het hoofdmenu terecht).

U neemt in dit geval de HERBYMON-listing en u zoekt naar het adres uit deze listing dat overeenkomt met het adres dat u links in beeld ziet. Gevonden? Dan typt u de regel netjes na. ('BS' als u een foutje maakt!)

Als u de regel foutloos heeft overgenomen, wordt deze geaccepteerd en krijgt u het volgende adres om in te vullen. Als u een fout maakt moet u na een 'CHECKSUM ERROR!' de regel nog eens proberen.

Dit gaat net zolang door tot de hele listing is ingevoerd, of totdat u er genoeg van heeft. In het laatste geval drukt u op de 'X'-toets. Vergeet u nu vooral niet om de ingevoerde data 'weg te SAVEn' met de gelijknamige optie!

## **DATA LOADEN**

U kunt nu het gedeelte binnenhalen waaraan u de vorige sessie hebt gewerkt. Misschien om het af te maken, of om een eventuele adresfout te verwijderen. Denk om de juiste bestandsnaam!

## **DATA SAVEN**

Vlak voordat u stopt met deze intyp-sessie, of gewoon voor de zekerheid, kunt u de thans ingevoerde data opslaan op cassette en/of diskette. U geeft een gewenste file-naam, opslaan en klaar!

## **STOPPEN**

Einde van het programma...

# 23 De HERBYMON

## 23.1 De programmalisting

SPEEDDUMP VAN A000 TOT C000

A000:	4E4F503E	4C442042	21D	A120:	2CC33E41	44442048	25E
A008:	432CC23E	4C442028	247	A128:	4C2C484C	3E4C4420	1FA
A010:	4243292C	413E494E	1FO	A130:	484C2C28	C2202020	20A
A018:	43204243	3E494E43	200	A138:	20293E44	45432048	1BB
A020:	20423E44	45432042	1CE	A140:	4C3E494E	43204C3E	20E
A028:	3E4C4420	422CC13E	25B	A148:	44454320	4C3E4C44	206
A030:	524C4341	3E455820	21D	A150:	204C2CC1	3E43504C	276
A038:	41462C41	46273E41	1E0	A158:	3E4A5220	4E432CC3	27A
A040:	44442048	4C2C4243	1ED	A160:	3E4C4420	53502CC2	27F
A048:	3E4C4420	412C2842	1C5	A168:	3E4C4420	28C22020	218
A050:	43293E44	45432042	1D8	A170:	2020292C	413E494E	1AB
A058:	433E494E	4320433E	1FC	A178:	43205350	3E494E43	21E
A060:	44454320	433E4C44	1FD	A180:	2028484C	293E4445	1CC
A068:	20432CC1	3E525243	275	A188:	43202848	4C293E4C	1D2
A070:	413E444A	4E5A20C3	298	A190:	44202848	4C292CC1	236
A078:	3E4C4420	44452CC2	265	A198:	3E534346	3E4A5220	214
A080:	3E4C4420	28444529	1C8	A1A0:	432CC33E	41444420	259
A088:	2C413E49	4E432044	1E9	A1A8:	484C2C53	503E4C44	231
A090:	453E494E	4320443E	1FF	A1B0:	20412C28	C2202020	1D7
A098:	44454320	443E4C44	1FE	A1B8:	20293E44	45432053	1C6
AOA0:	20442CC1	3E524C41	26E	A1C0:	503E494E	4320413E	207
AOA8:	3E4A5220	C33E4144	280	A1C8:	44454320	413E4C44	1FB
AOB0:	4420484C	2C44453E	1EB	A1D0:	20412CC1	3E434346	258
AOB8:	4C442041	2C284445	1CE	A1D8:	3E4C4420	422C423E	1DC
AOC0:	293E4445	43204445	1DC	A1E0:	4C442042	2C433E4C	1EB
AOC8:	3E494E43	20453E44	1FF	A1E8:	4420422C	443E4C44	1E4
AODO:	45432045	3E4C4420	1DB	A1F0:	20422C45	3E4C4420	1C1
AOD8:	452CC13E	5252413E	293	A1F8:	422C483E	4C442042	1E6
AOEO:	4A52204E	5A2CC33E	291	A200:	2C4C3E4C	4420422C	1D4
AOE8:	4C442048	4C2CC23E	270	A208:	28484C29	3E4C4420	1D3
AOF0:	4C442028	C2202020	1FA	A210:	422C413E	4C442043	1EO
AOF8:	20292C48	4C3E494E	1DE	A218:	2C423E4C	4420432C	1CB
A100:	4320484C	3E494E43	20F	A220:	433E4C44	20432C44	1E4
A108:	20483E44	45432048	1DA	A228:	3E4C4420	432C453E	1EO
A110:	3E4C4420	482CC13E	261	A230:	4C442043	2C483E4C	1F1
A118:	4441413E	4A52205A	21A	A238:	4420432C	4C3E4C44	1ED

A240:	20432C28	484C293E	1B2	A3E0:	3E414444	20412C48	1DC
A248:	4C442043	2C413E4C	1EA	A3E8:	3E414444	20412C4C	1E0
A250:	4420442C	423E4C44	1E4	A3F0:	3E414444	20412C28	1BC
A258:	20442C43	3E4C4420	1C1	A3F8:	484C293E	41444420	1E4
A260:	442C443E	4C442044	1E6	A400:	412C413E	414444320	1D4
A268:	2C453E4C	4420442C	1CF	A408:	412C423E	414444320	1D5
A270:	483E4C44	20442C4C	1F2	A410:	412C433E	414444320	1D6
A278:	3E4C4420	442C2848	1CE	A418:	412C443E	414444320	1D7
A280:	4C293E4C	4420442C	1D3	A420:	412C453E	414444320	1D8
A288:	413E4C44	20452C42	1E2	A428:	412C483E	414444320	1DB
A290:	3E4C4420	452C433E	1EO	A430:	412C4C3E	414444320	1DF
A298:	4C442045	2C443E4C	1EF	A438:	412C2848	4C293E41	1D1
A2A0:	4420452C	453E4C44	1E8	A440:	44432041	2C413E53	1E6
A2A8:	20452C48	3E4C4420	1C7	A448:	55422042	3E535542	221
A2B0:	452C4C3E	4C442045	1FO	A450:	20433E53	55422044	1EF
A2B8:	2C28484C	293E4C44	1DF	A458:	3E535542	20453E53	21E
A2C0:	20452C41	3E4C4420	1CO	A460:	55422048	3E535542	227
A2C8:	482C423E	4C442048	1EC	A468:	204C3E53	55422028	1DC
A2D0:	2C433E4C	4420482C	1D1	A470:	484C293E	53554220	205
A2D8:	443E4C44	20482C45	1EB	A478:	413E5342	4320412C	1E4
A2E0:	3E4C4420	482C483E	1E8	A480:	423E5342	4320412C	1E5
A2E8:	4C442048	2C4C3E4C	1FA	A488:	433E5342	4320412C	1E6
A2F0:	4420482C	28484C29	1BD	A490:	443E5342	4320412C	1E7
A2F8:	3E4C4420	482C413E	1E1	A498:	453E5342	4320412C	1E8
A300:	4C44204C	2C423E4C	1F4	A4A0:	483E5342	4320412C	1EB
A308:	442042C2	433E4C44	1ED	A4A8:	4C3E5342	4320412C	1EF
A310:	204C2C44	3E4C4420	1CA	A4B0:	28484C29	3E534243	1FB
A318:	4C2C453E	4C44204C	1F7	A4B8:	20412C41	3E414E44	1DF
A320:	2C483E4C	442042C2C	1DA	A4C0:	20423E41	4E442043	1D6
A328:	4C3E4C44	204C2C28	1DA	A4C8:	3E414E44	20443E41	1F4
A330:	484C293E	4C44204C	1F7	A4D0:	4E442045	3E414E44	208
A338:	2C413E4C	44202848	1CB	A4D8:	20483E41	4E44204C	1E5
A340:	4C292C42	3E4C4420	1D1	A4E0:	3E414E44	2028484C	1ED
A348:	28484C29	2C433E4C	1DE	A4E8:	293E414E	4420413E	1D9
A350:	44202848	4C292C44	1B9	A4F0:	584F5220	423E584F	240
A358:	3E4C4420	28484C29	1D3	A4F8:	5220433E	584F5220	20C
A360:	2C453E4C	44202848	1CF	A500:	443E584F	5220453E	21E
A368:	4C292C48	3E4C4420	1D7	A508:	584F5220	483E584F	246
A370:	28484C29	2C4C3E48	1E3	A510:	52204C3E	584F5220	215
A378:	414C543E	4C442028	1F7	A518:	28484C29	3E584F52	21C
A380:	484C292C	413E4C44	1F8	A520:	20413E4F	5220423E	1EO
A388:	20412C42	3E4C4420	1BD	A528:	4F522043	3E4F5220	203
A390:	412C433E	4C442041	1DF	A530:	443E4F52	20453E4F	215
A398:	2C443E4C	4420412C	1CB	A538:	5220483E	4F52204C	205
A3A0:	453E4C44	20412C48	1E8	A540:	3E4F5220	28484C29	1E4
A3A8:	3E4C4420	412C4C3E	1E5	A548:	3E4F5220	413E4350	211
A3B0:	4C442041	2C28484C	1D9	A550:	20423E43	5020433E	1D4
A3B8:	293E4C44	20412C41	1C5	A558:	43502044	3E435020	1E8
A3C0:	3E414444	20412C42	1D6	A560:	453E4350	20483E43	1FF
A3C8:	3E414444	20412C43	1D7	A568:	50204C3E	43502028	1D5
A3D0:	3E414444	20412C44	1D8	A570:	484C293E	43502041	1EF
A3D8:	3E414444	20412C45	1D9	A578:	3E524554	204E5A3E	22F

A580:	504F5020	42433E4A	21C	A720:	4A50204D	2CC23E45	278
A588:	50204E5A	2CC23E4A	28E	A728:	493E4341	4C4C204D	210
A590:	5020C23E	43414C4C	28C	A730:	2CC23EC7	20203E43	2B4
A598:	204E5A2C	C23E5055	299	A738:	5020C13E	52535420	288
A5A0:	53482042	433E4144	203	A740:	33383E52	4C43203E	1E8
A5A8:	4420412C	C13E5253	275	A748:	52524320	3E524C20	203
A5B0:	54203030	3E524554	1FD	A750:	3E525220	3E534C41	220
A5B8:	205A3E52	45543E4A	22B	A758:	203E5352	41203E53	1F5
A5C0:	50205A2C	C23EC420	2DA	A760:	4C49203E	53524C20	204
A5C8:	203E4341	4C4C205A	1F4	A768:	3E424954	20302C3E	1D7
A5D0:	2CC23E43	414C4C20	268	A770:	42495420	312C3E42	1DC
A5D8:	C23E4144	4320412C	255	A778:	49542032	2C3E4249	1E4
A5E0:	C13E5253	54203038	280	A780:	5420332C	3E424954	1FO
A5E8:	3E524554	204E433E	218	A788:	20342C3E	42495420	1BD
A5F0:	504F5020	44453E4A	220	A790:	352C3E42	49542036	1D4
A5F8:	50204E43	2CC23E4F	27C	A798:	2C3E4249	5420372C	1CC
A600:	55542028	C1202029	21B	A7A0:	3E524553	20302C3E	1E2
A608:	2C413E43	414C4C20	1E7	A7A8:	52455320	312C3E52	1F7
A610:	4E432CC2	3E505553	2B5	A7B0:	45532032	2C3E5245	1EB
A618:	48204445	3E535542	219	A7B8:	5320332C	3E524553	1FA
A620:	20C13E52	53542031	269	A7C0:	20342C3E	52455320	1C8
A628:	303E5245	5420433E	1FA	A7C8:	352C3E52	45532036	1DF
A630:	4558583E	4A502043	230	A7D0:	2C3E5245	5320372C	1D7
A638:	2CC23E49	4E20412C	250	A7D8:	3E534554	20302C3E	1E4
A640:	28C12020	293E4341	214	A7E0:	53455420	312C3E53	1FA
A648:	4C4C2043	2CC23EC5	2EC	A7E8:	45542032	2C3E5345	1ED
A650:	20203E53	42432041	1B7	A7F0:	5420332C	3E534554	1FD
A658:	2CC13E52	53542031	275	A7F8:	20342C3E	53455420	1CA
A660:	383E5245	5420504F	220	A800:	352C3E53	45542036	1E1
A668:	3E504F50	20484C3E	21F	A808:	2C3E5345	5420372C	1D9
A670:	4A502050	4F2CC23E	285	A810:	3E423E43	3E443E45	206
A678:	45582028	5350292C	1DD	A818:	3E483E4C	3E28484C	20A
A680:	484C3E43	414C4C20	20E	A820:	293E413E	494E2042	1DF
A688:	504F2CC2	3E505553	2C3	A828:	2C284329	3E4F5554	1F6
A690:	4820484C	3E414E44	20D	A830:	20284329	2C423E53	1B3
A698:	20C13E52	53542032	26A	A838:	42432048	4C2C4243	1EA
A6A0:	303E5245	54205045	20E	A840:	3E4C4420	28C22020	218
A6A8:	3E4A5020	28484C29	1DD	A848:	2020292C	42433E4E	1A6
A6B0:	3E4A5020	50452CC2	27B	A850:	45473E52	45544E3E	241
A6B8:	3E455820	44452C48	1F8	A858:	494D2030	3E4C4420	1D4
A6C0:	4C3E4341	4C4C2050	216	A860:	492C413E	494E2043	1EE
A6C8:	452CC23E	C620203E	2B5	A868:	2C284329	3E4F5554	1F6
A6D0:	584F5220	C13E5253	2BD	A870:	20284329	2C433E41	1A2
A6D8:	54203238	3E524554	207	A878:	44432048	4C2C4243	1EC
A6E0:	20503E50	4F502041	1FE	A880:	3E4C4420	42432C28	1C7
A6E8:	463E4A50	20502CC2	27C	A888:	C2202020	20293E52	1FB
A6F0:	3E44493E	43414C4C	225	A890:	4554493E	4C442052	222
A6F8:	20502CC2	3E505553	294	A898:	2C413E49	4E20442C	1D2
A700:	48204146	3E4F5220	1EE	A8A0:	2843293E	4F555420	1EA
A708:	C13E5253	54203330	27B	A8A8:	2843292C	443E5342	1D7
A710:	3E524554	204D3E4C	220	A8B0:	4320484C	2C44453E	1EA
A718:	44205350	2C484C3E	205	A8B8:	4C442028	C2202020	1FA

A8C0:	20292C44	453E494D	1D2	AA60:	3E444543	20C93E49	27A
A8C8:	20313E4C	4420412C	1AC	AA68:	4E432028	C9202BC8	2B5
A8D0:	493E494E	20452C28	1D7	AA70:	2020293E	44454320	193
A8D8:	43293E4F	55542028	1EA	AA78:	28C9202B	C8202029	26D
A8E0:	43292C45	3E414443	1E3	AA80:	3E4C4420	28C9202B	22A
A8E8:	20484C2C	44453E4C	1F3	AA88:	C8202029	2CC13E41	29D
A8F0:	44204445	2C28C220	223	AA90:	444420C9	202C5350	260
A8F8:	20202029	3E494D20	17D	AA98:	3E4C4420	422C28C9	24D
A900:	323E4C44	20412C52	1DF	AAA0:	202BC820	20293E4C	206
A908:	3E494E20	482C2843	1D4	AAA8:	4420432C	28C9202B	20F
A910:	293E4F55	54202843	1EA	AAB0:	C8202029	3E4C4420	21F
A918:	292C483E	53424320	1D3	AAB8:	442C28C9	202BC820	294
A920:	484C2C48	4C3E5252	236	AAC0:	20293E4C	4420452C	1A8
A928:	443E494E	204C2C28	1D9	AAC8:	28C9202B	C8202029	26D
A930:	43293E4F	55542028	1EA	AAD0:	3E4C4420	482C28C9	253
A938:	43292C4C	3E414443	1EA	AAD8:	202BC820	20293E4C	206
A940:	20484C2C	484C3E52	204	AAE0:	44204C2C	28C9202B	218
A948:	4C443E49	4E20462C	1F7	AAE8:	C8202029	3E4C4420	21F
A950:	2843293E	53424320	1CA	AAFO:	28C9202B	C8202029	26D
A958:	484C2C53	503E4C44	231	AAF8:	2C423E4C	442028C9	24D
A960:	2028C220	20202029	1B3	AB00:	202BC820	20292C43	1EB
A968:	2C53503E	494E2041	205	AB08:	3E4C4420	28C9202B	22A
A970:	2C284329	3E4F5554	1F6	AB10:	C8202029	2C443E4C	22B
A978:	20284329	2C413E41	1A0	AB18:	442028C9	202BC820	288
A980:	44432048	4C2C5350	20A	AB20:	20292C45	3E4C4420	1A8
A988:	3E4C4420	53502C28	1E5	AB28:	28C9202B	C8202029	26D
A990:	C2202020	20293E4C	1F5	AB30:	2C483E4C	442028C9	253
A998:	44493E43	50493E49	22E	AB38:	202BC820	20292C4C	1F4
A9A0:	4E493E4F	5554493E	254	AB40:	3E4C4420	28C9202B	22A
A9A8:	4C44443E	4350443E	227	AB48:	C8202029	2C413E4C	228
A9B0:	494E443E	4F555444	255	AB50:	4420412C	28C9202B	20D
A9B8:	3E4C4449	523E4350	23A	AB58:	C8202029	3E414444	238
A9C0:	49523E49	4E49523E	249	AB60:	20412C28	C9202BC8	291
A9C8:	4F544952	3E4C4444	250	AB68:	2020293E	41444320	18F
A9D0:	523E4350	44523E49	240	AB70:	412C28C9	202BC820	291
A9D8:	4E44523E	4F544452	25B	AB78:	20293E53	55422028	1B9
A9E0:	3E404142	43444546	213	AB80:	C9202BC8	2020293E	283
A9E8:	4748494A	4B4D4F50	259	AB88:	53424320	412C28C9	256
A9F0:	51525356	5758595A	2AE	AB90:	202BC820	20293E41	1FB
A9F8:	5B5E5F60	61626768	30A	AB98:	4E442028	C9202BC8	2B6
AA00:	696A6F70	72737879	388	ABA0:	2020293E	584F5220	1C0
AA08:	7A7BA0A1	A2A3A8A9	4CC	ABA8:	28C9202B	C8202029	26D
AA10:	AAABB0B1	B2B3B8B9	58C	ABBO:	3E4F5220	28C9202B	23B
AA18:	BABB0041	444420C9	327	ABB8:	C8202029	3E435020	222
AA20:	202C4243	3E414444	1D8	ABC0:	28C9202B	C8202029	26D
AA28:	20C9202C	44453E4C	248	ABC8:	3E524C43	2028C920	250
AA30:	4420C920	2CC23E4C	2C5	ABDO:	2BC82020	293E5252	23E
AA38:	442028C2	20202020	1CE	ABD8:	432028C9	202BC820	287
AA40:	292CC93E	494E4320	256	ABE0:	20293E52	4C2028C9	236
AA48:	C93E4144	4420C920	2D9	ABE8:	202BC820	20293E52	20C
AA50:	2CC93E4C	4420C920	2CC	ABFO:	522028C9	202BC820	296
AA58:	2C28C220	20202029	1BF	ABF8:	20293E53	4C412028	1AF

AC00:	C9202BC8	2020293E	283	ADA0:	28535029	2CC93E50	277
AC08:	53524120	28C9202B	242	ADA8:	55534820	C93E4A50	2B1
AC10:	C8202029	3E534C49	257	ADBO:	2028C920	293E4C44	228
AC18:	2028C920	2BC82020	264	ADB8:	2053502C	C93E0919	218
AC20:	293E5352	4C2028C9	269	ADC0:	21222329	2A2B3435	14D
AC28:	202BC820	20293E42	1FC	ADC8:	3639464E	565E666E	28B
AC30:	49542030	2C28C920	22A	ADD0:	70717273	7475777E	3A4
AC38:	2BC82020	293E4249	225	ADD8:	868E969E	A6AEB6BE	510
AC40:	5420312C	28C9202B	20D	ADE0:	CBCECBCB	CBCBCBCB	658
AC48:	C8202029	3E424954	24E	ADE8:	CBCBCBCB	CBCBCBCB	658
AC50:	20322C28	C9202BC8	282	ADF0:	CBCECBCB	CBCBCBCB	658
AC58:	2020293E	42495420	1A6	ADF8:	CBCBCBCB	CBCBCBCB	658
AC60:	332C28C9	202BC820	283	AE00:	E1E3E5E9	F9000000	48B
AC68:	20293E42	49542034	1BA	AE08:	00000000	00000000	000
AC70:	2C28C920	2BC82020	270	AE10:	00000000	00000000	000
AC78:	293E4249	5420352C	1C7	AE18:	00000000	00000000	000
AC80:	28C9202B	C8202029	26D	AE20:	00000000	00000000	000
AC88:	3E424954	20362C28	1C7	AE28:	060E161E	262E363E	110
AC90:	C9202BC8	2020293E	283	AE30:	464E565E	666E767E	310
AC98:	42495420	372C28C9	253	AE38:	868E969E	A6AEB6BE	510
ACA0:	202BC820	20293E52	20C	AE40:	C6CED6DE	E6EEF6FE	710
ACA8:	45532030	2C28C920	225	AE48:	00000000	00005630	086
ACB0:	2BC82020	293E5245	231	AE50:	00000000	00000000	000
ACB8:	5320312C	28C9202B	20C	AE58:	00000000	00000000	000
ACCO:	C8202029	3E524553	259	AE60:	00000000	00000000	000
ACC8:	20322C28	C9202BC8	282	AE68:	00000000	00000000	000
ACD0:	2020293E	52455320	1B1	AE70:	00000000	00000000	000
ACD8:	332C28C9	202BC820	283	AE78:	00000000	00000000	000
ACE0:	20293E52	45532034	1C5	AE80:	00000000	00000000	000
ACE8:	2C28C9A0	2BC82020	2F0	AE88:	00000000	00000000	000
ACF0:	293E5245	5320352C	1D2	AE90:	00000A0D	20484552	116
ACF8:	28C9202B	C8202029	26D	AE98:	42594D4F	4E202856	223
AD00:	3E524553	20362C28	1D2	AEA0:	312E3129	0A0D2020	110
AD08:	C9202BC8	2020293E	283	AEA8:	20535020	20204959	1C5
AD10:	52455320	372C28C9	25E	AEBO:	20202049	58202020	161
AD18:	202BC820	20293E53	20D	AEB8:	484C2020	20444520	19D
AD20:	45542030	2C28C920	226	AEC0:	20204243	20202041	166
AD28:	2BC82020	293E5345	232	AEC8:	0A0D2E2A	20002020	OCF
AD30:	5420312C	28C9202B	20D	AED0:	20202020	20202020	100
AD38:	C8202029	3E534554	25B	AED8:	20202020	20200020	OEO
AD40:	20322C28	C9202BC8	282	AEE0:	20202020	20202020	100
AD48:	2020293E	53455420	1B3	AEE8:	20202020	20202000	OEO
AD50:	332C28C9	202BC820	283	AEF0:	3E3F3F3F	3E20222E	1A9
AD58:	20293E53	45542034	1C7	AEF8:	2E2E2E22	000A0D2E	OF1
AD60:	2C28C920	2BC82020	270	AF00:	00492E0D	2E002020	OF2
AD68:	293E5345	5420352C	1D4	AF08:	20202020	20202020	100
AD70:	28C9202B	C8202029	26D	AF10:	20202020	20202000	OEO
AD78:	3E534554	20362C28	1D4	AF18:	00202020	20202020	OEO
AD80:	C9202BC8	2020293E	283	AF20:	20202020	20202020	100
AD88:	53455420	372C28C9	260	AF28:	20200000	0A0D2053	OCA
AD90:	202BC820	20293E50	20A	AF30:	4156494E	473A2000	1CF
AD98:	4F5020C9	3E4555820	283	AF38:	0A0D2053	45415243	1A5

AF40:	48494E47	3A20000A	18A	BOE0:	3E00B7C9	FE3038F4	418
AF48:	0D20464F	554E443A	1E3	BOE8:	FE4730F0	FE3A38F0	4C5
AF50:	20000A0D	204C4F41	133	BOFO:	FE4138E8	18EA3A6A	405
AF58:	44494E47	3A20000A	186	BOF8:	AE473A6C	AEB838DC	415
AF60:	0D202A2A	4552524F	1B9	B100:	C2EOB03A	6BAE473A	426
AF68:	522A2A00	30313233	16C	B108:	69AEB838	D318CDCB	48A
AF70:	34353637	38394142	1CA	B110:	27CB27CB	27CB27C9	3C6
AF78:	43444546	F5CDA200	376	B118:	CDBDB0CD	0FB13266	45F
AF80:	3A50AEB7	CA8E AFF1	4E7	B120:	AECDDBD0	473A66AE	47D
AF88:	F5CDA500	F1C9F1C9	5DB	B128:	B03266AE	CDAAB0FE	51B
AF90:	F3CD4A00	FBC9CD7C	517	B130:	20C28FB0	C9CDBDB0	524
AF98:	AF237EB7	20F8C921	409	B138:	CD0FB132	68AECDBD	45F
AFA0:	03AF18F6	21FDAE18	3A4	B140:	B0473A68	AEB03268	391
AFA8:	F1CD5601	3E0032A7	32C	B148:	AECDDBD0	CD0FB132	4A7
AFB0:	FCCD9F00	FE7FCAC7	576	B150:	67AECDBD	B0473A67	437
AFB8:	AFFE7BDA	C0AF3E00	4AF	B158:	AEB03267	AE18CDCD	457
AFC0:	FE61DAC7	AFD620CD	572	B160:	18B100C9	CD35B1ED	432
AFC8:	A200FE0D	20DEC921	395	B168:	4B67AEED	4369AEC9	470
AFD0:	6CAF4F06	00097EC9	2C0	B170:	CD35B1ED	4B67AEED	4ED
AFD8:	57CB3FCB	3FCB3FCB	440	B178:	436BAEC9	CD35B1ED	4C5
AFE0:	3FCDCFAF	3257AE7A	43B	B180:	4B67AEED	436DAEC9	474
AFE8:	E60FCDCF	AF3258AE	478	B188:	D1D1C9CD	A4AF3E00	4C9
AFF0:	C9CDD8AF	3A57AECD	529	B190:	CDD800FE	FF28F73E	4FF
AFF8:	7CAF3A58	AEC37CAF	459	B198:	00CDD500	FE0020E8	3A8
B000:	3E20C37C	AFCDF1AF	4B9	B1A0:	C9C8EB1	00C93E2E	40A
B008:	18F6CD7C	AF18F132	441	B1A8:	C92A69AE	7FEF2038	3DE
B010:	5EAE0680	3A5EAEAO	378	B1B0:	F5FE7E30	F1C92A69	4EE
B018:	C220B03E	30C322B0	395	B1B8:	AE7E3256	AECD05B0	3E4
B020:	3E31CD7C	AFCB83C2	42C	B1C0:	2A69AE23	2269AE3A	2D7
B028:	14B0C910	27E80364	313	B1C8:	52AE3D32	52AE3A56	2FF
B030:	000A0001	00012BB0	0E7	B1D0:	AEC9CD8B	B13E3ACD	4C5
B038:	3E05F50A	5F030A57	205	B1D8:	OAB0ED5B	69AECD68	44E
B040:	033EFFB7	C5444DED	43A	B1E0:	B0CDA9B1	32F7AECD	57B
B048:	523CD245	B06069C1	3DF	B1E8:	B6B1CDA9	B132F8AE	566
B050:	C630CD7C	AFF13DC2	4DE	B1F0:	CDB6B1CD	A9B132F9	586
B058:	3AB0C300	B07ACD0F	3B3	B1F8:	AECD6B1	CDA9B132	53B
B060:	B07BCD0F	B0C300B0	42A	B200:	FAAECD86	B121F5AE	5A0
B068:	7ACDF1AF	3A58AE32	459	B208:	C39AAF2A	69AE226B	3DA
B070:	5AAE3A57	AE3259AE	380	B210:	AECD2B1	CDF6B028	599
B078:	7BCDF1AF	C300B0CD	528	B218:	F8C3A4AF	CD64B1CD	5BD
B080:	C0003E00	3251AE3E	26D	B220:	AAB0FE20	28E5CDCB4	506
B088:	3FC37CAF	C118F0C1	4B7	B228:	B0CD70B1	18E37E12	429
B090:	18FAC118	FA21D8FF	4DD	B230:	23137EFFE	3EC22EB2	392
B098:	1128003A	DCF34719	2A2	B238:	C911CEAE	18F021DF	45E
B0A0:	10FD3ADD	F3472310	391	B240:	AE18F6CD	3EB221F1	48B
B0A8:	FDC921DD	F334CD95	54D	B248:	AE18EE3A	52AEB7CA	46F
B0B0:	B0C390AF	21DDF335	4D8	B250:	5EB247CD	00B0CD00	3A1
B0B8:	18F4E60F	C9CDAAB0	4F1	B258:	B0CD00B0	10F5CD00	3FF
B0C0:	FE3038CE	FE4730CA	473	B260:	B021CEAE	C39AAF3A	493
B0C8:	FE3A38EE	FE4138C2	497	B268:	53AE3C32	53AEC93A	373
B0D0:	D63718E6	FE3A38E2	45D	B270:	53AE473E	003253AE	2B9
B0D8:	D63718DE	3E01B7C9	3C2	B278:	C9CD3EB2	CDB6B1CB	585

B280:	3FCB3FCB	3FCD72B2	444	B420:	72B221BE	AD7E23FE	44F
B288:	2143A7B8	CAA1B223	403	B428:	00CA43B2	B8CA3BB4	430
B290:	7FEF3EC2	99B2CD67	4FB	B430:	FE00CA43	B2CD67B2	4A3
B298:	B23A53AE	B8C28FB2	4A8	B438:	3E01B7C2	25B4211B	2CD
B2A0:	23CD39B2	3A56AEE6	3FF	B440:	AACD6FB2	B8CA5AB4	528
B2A8:	07CD72B2	2111A8B8	38A	B448:	237FEF3E	C252B4CD	472
B2B0:	CAC5B223	7FEF3EC2	4E0	B450:	67B23A53	AEB8C248	416
B2B8:	BDB2CD67	B23A53AE	490	B458:	B423CD39	B2CD97B3	4A6
B2C0:	B8C2B3B2	23C32EB2	4A5	B460:	3E01325D	AE21CEAE	319
B2C8:	E5CDB6B1	E13E2477	4D3	B468:	7EFEC8C2	78B4CDFD	5FC
B2D0:	233A57AE	77233A58	28E	B470:	B23E0032	5DAE3E00	26B
B2D8:	AE77C9E5	CDB6B13A	541	B478:	B723C268	B43A5DAE	3FD
B2E0:	58AE325A	AE3A57AE	37F	B480:	FE00CA88	B421CEAE	4A1
B2E8:	3259AECD	B6B1E1CD	51B	B488:	7EFEC1CA	C8B2FEC2	641
B2F0:	CDB2233A	59AE7723	37D	B490:	CADBB2B7	23C288B4	52F
B2F8:	3A5AAE77	C9E5CDB6	4EA	B498:	C93E5832	02AFC314	319
B300:	B1E13E26	18C93E26	33B	B4A0:	B43E5932	02AFC314	305
B308:	77233A5B	AE77233A	2B1	B4A8:	B421CEAE	7EFEC1CA	558
B310:	5CAE77C9	E5CDB6B1	563	B4B0:	C8B2FEC2	CADBB2FE	68F
B318:	2A69AE4F	E680CA26	3E6	B4B8:	C3CA14B3	FEC4CA79	559
B320:	B33EFFC3	28B33E00	3CC	B4C0:	B2FEC5CA	99B4FEC6	650
B328:	47097D32	5EAE7CCD	354	B4C8:	CA44B3FE	C7CAA1B4	5A5
B330:	D8AFE13E	40CDCFB2	534	B4D0:	B723C2AC	B4C9CDB6	548
B338:	23E53A5E	AECDD8AF	4A2	B4D8:	B1CD73B2	2100A03A	39E
B340:	E1C3D1B2	CD3EB2CD	5B1	B4E0:	56AE473A	53AEB8CA	408
B348:	B6B1CD72	B221E1A9	503	B4E8:	FCB4237E	FE3EC2F4	543
B350:	7E23FE00	CA43B2B8	416	B4F0:	B4CD67B2	3A53AEB8	48D
B358:	CA66B3FE	00CA43B2	4A0	B4F8:	C2EAB423	C339B2CD	4FE
B360:	CD67B23E	01B7C250	3EE	B500:	8BB13E3B	CD0ABOED	429
B368:	B32124A8	CD6FB2B8	446	B508:	5B69AECD	68B0CD3E	462
B370:	CA85B323	7FEF3EC2	4A1	B510:	B23E0432	52AECD6	3C9
B378:	7DB3CD67	B23A53AE	451	B518:	B4CDA9B4	C34BB22A	4C8
B380:	B8C273B3	23CD39B2	47B	B520:	69AE2322	6BAECFFF	441
B388:	21CEAE7E	FEC2CADC	580	B528:	B4CDF6B0	28F8C3A4	5AE
B390:	B2B723C2	8BB3C921	476	B530:	AFCD64B1	CDAAB0FE	5B6
B398:	CEAE237E	FE00CA95	47A	B538:	2028E4CD	B4B0CD70	49A
B3A0:	AFFEC9C2	9AB33A01	4C0	B540:	B118E3F1	C92A69AE	4A7
B3A8:	AF77233A	02AF77C3	36E	B548:	3A75AE77	232269AE	330
B3B0:	97B3CDB6	B13A58AE	4BE	B550:	C92A69AE	3A72AE77	3DB
B3B8:	325CAE3A	57AE325B	308	B558:	232269AE	2A69AE3A	2D7
B3C0:	AECDB6B1	E607FE06	4D3	B560:	71AE7723	2269AEC9	3BB
B3C8:	C243B23A	56AECD72	434	B568:	ED4B69AE	2A79AE2B	3CB
B3D0:	B22106AE	7E23B8CA	3AA	B570:	B7ED427D	E680C282	50D
B3D8:	E0B3CD67	B23E01B7	46F	B578:	B57CFE00	C28CB0C3	4F0
B3E0:	C2D4B321	1BAACD6F	46B	B580:	88B57CFE	FFC28CB0	5B4
B3E8:	B2B8CAFF	B3237EFE	585	B588:	7D2A69AE	77232269	2E3
B3F0:	3EC2F7B3	CD67B23A	4CA	B590:	AEC90600	1100A021	24F
B3F8:	53AEB8C2	EDB323CD	50B	B598:	19AF1ABE	CAB0B5FE	4CD
B400:	39B2CD97	B321CEAE	49F	B5A0:	3ECAABB5	131AFE3E	3D1
B408:	7EFEC8CA	06B3B723	4A1	B5A8:	C2A4B513	0420E8C9	403
B410:	C208B4C9	CD3EB2CD	4D1	B5B0:	2313FE3E	C29AB53E	3C1
B418:	B6B1FECB	CAB2B3CD	62C	B5B8:	00325FAE	2A69AE78	2F8

B5C0:	77232269	AE3A60AE	31B	B760:	3260AE3A	71AECD1A	380
B5C8:	FE00CA51	B53A61AE	417	B768:	B73A72AE	B03271AE	412
B5D0:	FE00CA45	B53A62AE	40C	B770:	3A73AECD	1AB73A74	3A7
B5D8:	FE00CA68	B5C91124	3E3	B778:	AEB03272	AEC331B7	45B
B5E0:	A801E1A9	2119AF1A	336	B780:	23C334B7	2106AF7E	325
B5E8:	BECAFFB5	FE3ECAF8	63A	B788:	FE00CA95	AFFE40C2	50C
B5F0:	B5131AFAE	3EC2F1B5	486	B790:	D3B70600	1179AE23	2EB
B5F8:	13030AB7	20E6C923	2C9	B798:	7ECDE4B0	20E97ECD	533
B600:	13FE3EC2	E7B53E00	3EB	B7A0:	D4B01213	043E04B8	2A7
B608:	325FAE2A	69AE3EED	3AB	B7A8:	C297B7CD	24B73EC3	4B9
B610:	77232269	AE2A69AE	314	B7B0:	773E0032	62AE3A79	2AA
B618:	0A772322	69AE3A60	277	B7B8:	AECD1AB7	3A7AAEBO	45E
B620:	AEFE00CA	51B5C911	456	B7C0:	327AAE3A	7BAECD1A	3A4
B628:	1BAA01BE	AD2119AF	31A	B7C8:	B73A7CAE	B03279AE	424
B630:	1ABECA48	B6FE3ECA	4A6	B7D0:	C384B723	C387B721	443
B638:	41B6131A	FE3EC23A	35C	B7D8:	06AF7EFE	00CA95AF	43F
B640:	B613030A	B720E6C9	35C	B7E0:	FE24C219	B8060011	2CC
B648:	2313FE3E	C230B63E	358	B7E8:	75AE237E	CDE4B020	445
B650:	00325FAE	2A69AE3A	2BA	B7F0:	E97ECDD4	B0121304	3E1
B658:	63AE7723	2269AE2A	30E	B7F8:	3E02B8C2	EAB7CD2A	452
B660:	69AE0A77	3263AE23	2FE	B800:	B73EC177	3E003261	2FE
B668:	2269AE3A	65AEFE00	384	B808:	AE3A75AE	CD1AB73A	3E3
B670:	C27EB63A	81AE2A69	3F2	B810:	76AEBO32	75AEC3D7	4C3
B678:	AE772322	69AE3A60	31B	B818:	B723C3DA	B72106AF	404
B680:	AEFE00CA	51B53A61	417	B820:	7EFE00CA	95AFFE26	4AE
B688:	AEFE00CA	45B53A63	40D	B828:	C25FB806	001181AE	31F
B690:	AEFECBC2	95AF2148	4E6	B830:	237ECDE4	B020E97E	489
B698:	00097E2A	69AE7723	262	B838:	CDD4B012	13043E02	2BA
B6A0:	2269AEC9	1143A706	303	B840:	B8C230B8	CD2AB73E	44E
B6A8:	002119AF	1ABECA12	29D	B848:	C8773E00	3265AE3A	2FC
B6B0:	B7FE3EC2	O2B7D522	465	B850:	81AECD1A	B73A82AE	437
B6B8:	7DAE1111	A80E002A	22D	B858:	B03281AE	C31DB823	3CC
B6C0:	7DAE1ABE	CAD7B613	46D	B860:	C320B821	06AF7EFE	3ED
B6C8:	1AFE3EC2	C7B6130C	3B4	B868:	00CA95AF	FE49C29D	4B4
B6D0:	79FE0820	EAD1C923	446	B870:	B8237EFE	58C280B8	4A9
B6D8:	13FE3EC2	C2B6D13E	498	B878:	3EDD3263	AEC38FB8	468
B6E0:	00325FAE	2A69AE3E	2BE	B880:	FE59C28D	B83EFD32	4CB
B6E8:	CB772322	69AE2A69	331	B888:	63AEC38F	B818D73E	448
B6F0:	AECB20CB	20CB2079	3E8	B890:	20772B3E	C9773E00	27E
B6F8:	B0772322	69AEC9C3	40F	B898:	3264AE18	C62318C6	323
B700:	12B7131A	FE3EC202	2F6	B8A0:	2106AF11	19AF7E12	23F
B708:	B7130478	FE20C2A9	3CF	B8A8:	23137EB7	C2A6B83E	3C9
B710:	B6C92313	FE3EC2AC	45F	B8B0:	3E121B1A	FE20CAA	31C
B718:	B6C9CB27	CB27CB27	455	B8B8:	B8C92A6F	AE2269AE	401
B720:	CB2747C9	3E20772B	302	B8C0:	3E1ECD7C	AFCD1FB5	3F5
B728:	772B3E20	772B772B	244	B8C8:	CD9FAF3E	41CDOABO	421
B730:	C92106AF	7FEF00CA	3E5	B8D0:	ED5B69AE	CD68B03E	482
B738:	95AFFE24	C280B706	465	B8D8:	003251AE	C92A6FAE	341
B740:	001171AE	237ECDE4	382	B8E0:	2269AE3E	1ECD7CAF	38D
B748:	B020E97E	CDD4B012	49A	B8E8:	CD26B5CD	9FAF3E00	401
B750:	13043E04	B8C244B7	2CE	B8F0:	3251AEC9	2A6FAE22	363
B758:	CD24B73E	C2773E00	35D	B8F8:	69AE3E1E	CD7CAFCD	438

B900:	0BE2CD9F	AF3E3ACD	41D	BAA0:	69AE2287	AECD64B1	450
B908:	0AB0ED5B	69AECD68	44E	BAA8:	2A69AE22	89AECD64	3CB
B910:	B03E0032	51AEC9CD	3B5	BAB0:	B12A69AE	228BAECD	41A
B918:	64B11106	AFD5CDA	427	BAB8:	64B12A69	AE228DAE	3B3
B920:	B0D11213	1AB720F5	38C	BAC0:	CD64B12A	69AE228F	3D4
B928:	3E013260	AE3261AE	2C0	BAC8:	AECD5FB1	3A66AE32	40B
B930:	3262AE32	64AE3265	31D	BAD0:	91AE3E1E	CD7CAFCD	460
B938:	AE325FAE	CD31B7CD	46F	BAD8:	7CAFCD7C	AFC3F5B9	594
B940:	D7B7CD84	B7CDA0B8	5BB	BAE0:	CD64B1CD	70B1CD5F	4FC
B948:	2A69AE22	6FAECD92	3DF	BAE8:	B12A69AE	3A66AE77	3B7
B950:	B53A5FAE	FE00CABA	47E	BAF0:	232269AE	CDF6B028	3F7
B958:	B8CDDEB3	3A5FAEFE	55D	BAF8:	F0C3A4AF	CD64B1CD	5B5
B960:	00CABAB8	CDA4B63A	49D	BB00:	70B1CD7C	B12A69AE	45C
B968:	5FAEFE00	CABAB8CD	514	BB08:	7E232269	AE2A6DAE	31F
B970:	1DB8CD63	B8CDA0B8	4E2	BB10:	7723226D	AECDF6B0	44A
B978:	CD27B63A	5FAEFE00	3EF	BB18:	28EBC3A4	AFCD64B1	50B
B980:	CABAB83E	0932DDF3	485	BB20:	CD70B121	06AF226D	353
B988:	C37FB0CD	64B12A69	467	BB28:	AE3E0032	56AECDAA	399
B990:	AE226FAE	3E003256	2B3	BB30:	B0FE22C2	62BBCDAA	526
B998:	AECD5FB1	2A69AE3A	406	BB38:	B02A6DAE	77FE22C2	44E
B9A0:	66AE7723	2269AECD	3B4	BB40:	47BB3E00	C351BB3A	349
B9A8:	AAB0FE20	CADD83A	511	BB48:	DDF3FE1E	C251BB3E	4F8
B9B0:	56AE3C32	56AEFE04	378	BB50:	0023226D	AE2156AE	285
B9B8:	CADD8CD	B4B018D9	581	BB58:	34FE00C2	36BB35C3	3DD
B9C0:	CD64B12A	69AE226F	3B4	BB60:	90BBCDB4	BOCD5FB1	559
B9C8:	AE3E0032	56AECD5F	34E	BB68:	2A6DAE3A	66AE7723	32D
B9D0:	B12A69AE	3A66AE77	3B7	BB70:	226DAE21	56AE34CD	363
B9D8:	232269AE	CDAA0FE	481	BB78:	AAB0FE20	C285BB3E	4B8
B9E0:	20CAF4B8	3A56AE3C	410	BB80:	00B7C38D	BBCDB4B0	4F3
B9E8:	3256AEFE	04CAF4B8	4AE	BB88:	3A56AEFE	08C265BB	426
B9F0:	CDB4B018	D92192AE	483	BB90:	3E063254	AE2106AF	24E
B9F8:	CD9AAFED	5B85AECD	55E	BB98:	226DAE2A	69AE2267	307
BA00:	68B0ED5B	87AECD68	4CA	BBA0:	AE3A56AE	472A67AE	372
BA08:	BOED5B89	AECD68B0	514	BBA8:	7E2A6DAE	BE CABFB	4C5
BA10:	ED5B8BAE	CD68B0ED	553	BBB0:	2A69AE23	2269AECD	36A
BA18:	5B8DAECD	68B0ED5B	4C3	BBB8:	F6B028D9	C3A4AF2A	4E7
BA20:	8FAECD68	B03A91AE	49B	BBC0:	67AE2322	67AE2A6D	306
BA28:	CDF1AFC3	A4AFCD00	550	BBC8:	AE23226D	AE0520D5	308
BA30:	00FD2287	AEDD2289	3DC	BBD0:	3A54AE3C	3254AEFE	3AA
BA38:	AE228BAE	ED538DAE	484	BBD8:	07C2EEBB	3E0ACD7C	403
BA40:	ED438FAE	3291AEC3	4A1	BBE0:	AF3E0DCD	7CAFCD1	460
BA48:	F5B9CD64	B12A69AE	4D1	BBE8:	B13E0032	54AEED5B	36B
BA50:	222FBAA3	91AEE4B	3BC	BBF0:	69AECD68	B02A69AE	43D
BA58:	8FAEED5B	8DAE2A8B	475	BBF8:	232269AE	CDF6B0CA	499
BA60:	AEDD2A89	AEFD2A87	49A	BC00:	95BBC3A4	AFCD64B1	548
BA68:	AE18C3C3	0000CD64	37D	BC08:	3E0ACD7C	AF3E0DCD	358
BA70:	B12A69AE	226CBA3A	374	BC10:	7CAF3E25	CD7CAFED	473
BA78:	91AEE4B	8FAEED5B	4FC	BC18:	5B69AECD	5DB03E24	3AE
BA80:	8DAE2A8B	AEDD2A89	42E	BC20:	CD7CAFED	5B69AECD	524
BA88:	AEFD2A87	AEED7B85	4F7	BC28:	68B03E23	CD7CAF2A	39B
BA90:	AE18D8CD	64B12A69	413	BC30:	69AECD35	B0C3A4AF	4DF
BA98:	AE2285AE	CD64B12A	40F	BC38:	3E083252	AECDAAB0	39F

BC40:	FE30C24A	BC0600C3	3BF	BDE0:	F92119AF	06067ECD	339
BC48:	54BCFE31	CA52BCC3	4DA	BDE8:	9ABD2310	F9CDF000	440
BC50:	8CB00601	3A66AECB	35C	BDF0:	2A69AEE5	ED5B6BAE	487
BC58:	27B03266	AE2152AE	33E	BDF8:	EBB7ED52	E5226BAE	501
BC60:	3520DAC9	CD38BC32	3EB	BE00:	CD8CED06	02E17DCD	449
BC68:	6AAECD38	BC3269AE	422	BE08:	9ABD7CCD	9ABD10F5	4FC
BC70:	CDAAB0FE	20C27FB0	536	BE10:	ED5B69AE	2A6BAE1A	3BC
BC78:	C308BC3E	003252AE	2F7	BE18:	CD9ABD13	2B7CB5C2	455
BC80:	21000001	2BB0E5C5	2A7	BE20:	17BECDF0	00D5ED5B	4AF
BC88:	CDAAB0FE	30DA92B0	571	BE28:	69AECD68	B0D1CD68	502
BC90:	FE3AD292	BOE60F32	473	BE30:	BOC3A4AF	3E003250	386
BC98:	53AEC10A	5F030A57	28F	BE38:	AEED737D	AEC4D42BD	505
BCA0:	03E13A53	AEB7CAAЕ	44E	BE40:	3E003256	AE321FAF	274
BCA8:	BC193DC2	A9BC3A52	3C5	BE48:	CDAAB0FE	20CA5BBE	528
BCB0:	AE3C3252	AEFE0520	33F	BE50:	CDB4B0CD	64B13E01	452
BCB8:	CD7C326A	AE7D3269	3AB	BE58:	3256AE21	38AFCD9A	3A5
BCC0:	AEC9CD7B	BCCDAAB0	5A2	BE60:	AF2119AF	CD9AAFC4D	47B
BCC8:	FE20C27F	B0C308BC	496	BE68:	AEBD060A	CDA4BD4E	4A7
BCD0:	CD64B1CD	70B12A69	463	BE70:	80C26ABE	10F62106	397
BCD8:	AEED4B6B	AE09CDB9	48E	BE78:	AF0606CD	A4BD7723	383
BCE0:	BCC308BC	CD64B1CD	4F2	BE80:	10F93600	CDE70021	314
BCE8:	70B12A69	AEED4B6B	405	BE88:	47AFCD9A	AF2106AF	3E2
BCF0:	AEB7ED42	18E8CD64	4C5	BE90:	CD9AAF11	19AF2106	316
BCF8:	B1CD70B1	3A69AE21	411	BE98:	AF06061A	BEC25BBE	36E
BD00:	6BAEA65F	3A6AAE21	391	BEA0:	132310F7	2152AFCD	32C
BD08:	6CAEA657	EB18CFCD	4B6	BEA8:	9AAFCDAE	BD0602CD	456
BD10:	64B1CD70	B13A69AE	454	BEBO:	A4BD6FCD	A4BD67E5	54A
BD18:	216BAEB6	5F3A6AAE	3A1	BEB8:	10F5D13A	56AEB7CA	495
BD20:	216CAEB6	57EB18B6	401	BEC0:	C6BEED5B	69AEED53	523
BD28:	CD64B1CD	70B13A69	473	BEC8:	69AAE1CD	A4BD1213	44B
BD30:	AE216BAE	AE5F3A6A	399	BED0:	2B7CB5C2	CBBECDE7	55B
BD38:	AE216CAE	AE57EBC3	49C	BED8:	00D5ED5B	69AECD68	469
BD40:	DEBCCDAA	B0FE22C2	5A3	BEE0:	B0D1CD68	BOC3A4AF	57C
BD48:	8CB02119	AF0600E5	310	BEE8:	3E013250	AEC3A4AF	385
BD50:	C5CDAAB0	C1E1FE22	5AE	BEFO:	3E0018F6	CD64B1CD	3FB
BD58:	CA66BD77	23043E11	2DA	BEF8:	42BD2A69	AE1119AF	319
BD60:	B820ECC3	8CB03E20	421	BF00:	1AB7CA0A	BF771323	311
BD68:	7723043E	12B820F6	2BC	BF08:	18F6226B	AEC311B2	3CF
BD70:	3E0077CD	AAB0FE20	3FA	BF10:	F5B9C2BC	05BC64BC	50D
BD78:	C28CB0C9	ED7B7DAE	55A	BF18:	93BAD0BC	E4BCC0B9	5F2
BD80:	CDF00021	5FAFC9A	453	BF20:	8BB917B9	4ABA31B5	3FE
BD88:	AFC3A4AF	C5D5E53E	582	BF28:	E0BA1DBB	6EBA1CB2	468
BD90:	00CDEA00	38E6E1D1	487	BF30:	FCBA43B5	B8BD34BE	515
BD98:	C1C9C5D5	E5CDED00	5C3	BF38:	E8BEF0BE	F6BC0FBD	5D2
BDAO:	38DA18F2	C5D5E5CD	568	BF40:	28BDF4BE	21232425	324
BDA8:	E40038D0	18E8C5D5	486	BF48:	2A2B2D3A	3B414344	1BF
BDB0:	E5CDE100	38C618DE	487	BF50:	46484A4D	5458534C	270
BDB8:	3E003250	AEED737D	34B	BF58:	50512640	2F5700CD	25A
BDC0:	AECD42BD	3E00321F	309	BF60:	9FAFC37F	B03E0132	3B1
BDC8:	AFCD64B1	CD70B121	4A0	BF68:	51AECD95	B0CD90AF	51D
BDD0:	2CAFCD9A	AFCD8CBD	507	BF70:	FE2EC25F	BFCDAAB0	533
BDD8:	060A3E80	CD9ABD10	302	BF78:	572110BF	O144BF0A	255

BF80: BAC298BF	7E326CBA	4A9	BFC0: AEF3CD6C	00CDCC00	473
BF88: 237E326D	BACDAAB0	421	BFC8: CDC3003E	003250AE	2FE
BF90: FE20C27F	BOC36BBA	4F7	BFDO: CDF5B93E	013251AE	3EB
BF98: 032323FE	00C27FBF	347	BFD8: 3A51AEFE	01C2E3BF	49C
BFA0: C37FB0ED	7385AEFD	582	BFE0: CD9FAFCD	A9AFCD65	572
BFA8: 2287AE2D	2289AE22	3AF	BFE8: BF18EDFF	FFFFFFFF	6BF
BFB0: 8BAEED53	8DAEED43	4E4	BFF0: FFFFFFFF	FFFFFFFF	7F8
BFB8: 8FAE3291	AE3E2732	345	BFF8: FFFFFFOO	FFFFFFFF	6F9

## 23.2 Werken met de HERBYMON

```

10 CLEAR200 ,&HFFF:SCREEN0:CLS
20 BLOAD"HERBY.MON"
30 DEFUSR=49059!:KEY1,"M=USR(0)"+CHR$(13)
40 PRINT"F1 - HERBYMON":NEW

```

Laten we aannemen dat het u is gelukt om de HERBYMON op te starten. Waarschijnlijk doet u dit met behulp van een simpele BASIC-loader (listing 23.1) of misschien zelfs meteen met behulp van het 'BLOAD"HERBY.MON"', R'-commando. In beide gevallen krijgt u in 40-koloms mode het volgende openingsscherm:

HERBYMON (V1.1)

SP	IY	IX	HL	DE	BC	A
----	----	----	----	----	----	---

.* 9D17	DE92	F38B	BF96	0007	0300	00
---------	------	------	------	------	------	----

De hexadecimale waarden kunnen afwijken; daarover straks meer. De uitdrukking 'HERBYMON (V1.1)' snapt u wel, maar waar slaan die letters op? Welnu: die verwijzen naar de inmiddels welbekende interne Z80-registers.

SP: stack pointer (of stapelwijzer)

IY: index-Y register

IX: index-X register

HL: HL-registerpaar

DE: DE-registerpaar

BC: BC-registerpaar

A: accumulator

De getallen die zich onder dit illustere zevental bevinden, zullen eventueel veranderen onder invloed van uw eigen routines en/of een speciaal HERBYMON-commando. Bedenk dat u deze waarden kunt gebruiken tijdens de beoordeling van routines in de testfase.

U heeft 26 commando's tot uw beschikking. Even in het kort:

- A - assembleren
- C - subroutine-aanroep, met terugkeer in de HERBYMON
- D - disassembleren
- F - een geheugengebied met een bepaalde waarde vullen
- H - een geheugengebied doorzoeken op de aanwezigheid van een gewenste reeks tekens
- J - een sprong vanuit de HERBYMON naar uw eigen programma
- L - een HERBYMON-file van cassette laden
- M - een memory-dump in bytes en ASCII
- P - de printer activeren
- Q - de printer deactiveren
- S - een stuk geheugen als HERBYMON-file naar cassette saven
- T - een stuk geheugen verplaatsen
- W - een stuk ASCII-tekst op een gewenste geheugenplaats schrijven
- X - verlaten van de HERBYMON
- ! - te voorschijn halen van de 'interne' registers
- # - conversie van een decimale waarde
- \$ - conversie van een hexadecimale waarde
- % - conversie van een binaire waarde
- & - logische AND op twee waarden
- \* - invullen van de interne registers
- + - de som van twee waarden berekenen
- - het verschil van twee waarden berekenen
- / - logische XOR op twee waarden
- : - bytes plaatsen met memory-dump

- ; - bytes plaatsen met disassemblering
- @ - logische OR op twee waarden

Van de meeste commando's zult u de syntax en het effect waarschijnlijk al vermoeden. Toch krijgt u nog even een nadere uitleg.

## A - ASSEMBLEREN

Het per regel assembleren van machinecode. U doet het als volgt:

.A [startadres] [mnemonic] [operand(s)]

Tussen elke parameter staat één spatie en na het indrukken van de RETURN-toets zal de HERBYMON op twee manieren kunnen reageren:

- 1 De syntax wordt geaccepteerd, omgezet en gedisassembleerd. Het gevolg is dat u vanaf de volgende (logische) geheugenplaats verder kunt gaan.
- 2 U maakte een foutje! En zie, een vraagteken. (In dit geval heeft u een niet-bestaaende instructie bedacht of zit de vergissing in de operand.)

Om goed te kunnen assembleren even de volgende wenken.

Een relatieve sprong geeft u aan met '@....', een index-offset met '&..' en de gewone bytes en words met respectievelijk '\$..' en '\$....'!

Bijvoorbeeld:

```
.A E000 DJNZ @E000
.A E000 LD (IX+&7F),$FF
.A E000 LD HL,$1234
.A E000 RET
```

TIP: tussen \$C000 en \$F380 ligt een prima gebied dat u vanuit BASIC beschermt met de opdracht CLEAR200,&hBFFF!)

## C - SUBROUTINE

U springt naar een zelfgeschreven machinecode-subroutine en keert als alles goed gaat weer netjes in de HERBYMON terug. Dat gaat als volgt:

.C [startadres]

Bijvoorbeeld:

.C 00C0

TIP: maak gebruik van dit commando om gewenste of ongewenste effecten te doorgronden. Het is mogelijk om eerst de interne registers te veranderen, dit voor optimaal gemak.

## D – DISASSEMBLEREN

U laat een aantal bytes decoderen naar leesbare Z80-instructies. Dit commando is dus het tegenovergestelde van het assembleercommando. Doe het zo:

.D [startadres] of

.D [startadres] [eindadres]

Bijvoorbeeld:

.D 1000

.D 1000 2000

(TIP: U pauzeert het disassembleren door op de spatiebalk te drukken. U onderbreekt voortijdig met behulp van één van de cursortoetsen.)

## F – FILL of ‘VOL-POKEN’

U vult een stuk geheugen met een gewenste byte-waarde:

.F [startadres] [eindadres] [byte]

Bijvoorbeeld:

.F D000 E000 00

Kijk nu maar met het M-commando naar de inhoud van hetzelfde gebied.

**TIP:** om te weten of uw routines niets verknocien, kunt u een stuk geheugen vullen en dan na afloop kijken of er iets fout is gegaan.

## **H – HUNT of ZOEKEN**

U onderzoekt een stuk geheugen op de aanwezigheid van een bepaald teken of een bepaalde reeks tekens. Dat kan op twee manieren:

.H [startadres] [eindadres] [byte] ... [byte]  
.H [startadres] [eindadres] [string]

Bijvoorbeeld:

```
.H 1000 2000 FF FF  
.H 1000 8000 BASIC
```

**TIP:** zie de D-tip. Gebruik dit commando om dom zoeken te voorkomen. Handig als u niet meer weet waar uw teksten zijn gebleven!

## **J – JUMP of ‘SPRINGEN’**

U springt vanuit de HERBYMON naar uw zelfgeschreven machinetaalprogramma. Vooraf kunt u nog even de interne registers wijzigen; deze worden gekopieerd voordat de sprong in het diepe wordt gemaakt. De syntax:

.J [startadres]

Bijvoorbeeld:

```
.J 0000
```

(TIP: spring vanuit de HERBYMON om zeker te weten dat u ‘goed aankomt’... Bekijk vooraf altijd even de interne registers!)

## **L – LOADEN**

Een file in HERBYMON-formaat(!) van cassette laden. Twee mogelijkheden:

.L [file-naam] of  
.L [file-naam] [loadadres]

Bijvoorbeeld:

```
.L TEST  
.L TEST D880
```

TIP: HERBYMON-formaat is veiligheidshalve afwijkend van het standaard-MSX-formaat. Een file die is opgeslagen vanuit de HERBYMON kunt u ook weer binnenvullen. Een 'vreemde' file leest u in met een BASIC-loader, dus met het BLOAD-commando.

## M – MEMORY-DUMP of GEHEUGENDUMP

U kunt een stuk geheugen bekijken in zowel bytes als ASCII. Op twee verschillende manieren:

.M [startadres]  
.M [startadres] [eindadres]

Bijvoorbeeld:

```
.M 2000  
.M 2000 3000
```

TIP: zie de D-tip. Gebruik dit commando om bijvoorbeeld teksten te kunnen lezen of om gewoon een stuk geheugen te onderzoeken.

## P – PRINTER ACTIVEREN

Na het geven van dit commando krijgt u alles zowel op het scherm als op de printer, mits er een MSX-compatible printer wordt gebruikt

.P

TIP: als de printer niet staat aangesloten, komt u 'los' met behulp van CONTROL-STOP.

## **Q – PRINTER DEACTIVEREN**

Printen kost papier. Als u klaar bent met het printen van het gewenste materiaal, schakelt u de simultane printerwerking met dit commando weer uit:

.Q

## **S – SAVEN**

U bewaart een file in het afwijkende HERBYMON-formaat op cassette:

.S [file-naam] [startadres] [eindadres]

Bijvoorbeeld:

.S TEST 1000 2000

TIP: als u zich heeft vergist, drukt u op CONTROL-STOP. Als u een file in het standaard MSX-formaat wilt opslaan, gaat u met behulp van het X-commando terug naar BASIC en gebruikt u de 'BSAVE'-instructie.

## **T – TRANSFER of VERPLAATSEN**

U kunt een stuk geheugen verplaatsen naar een nieuw adres:

.T [beginadres] [eindadres] [verplaatsadres]

Bijvoorbeeld:

.T A000 C000 D000

TIP: gebruik dit commando in combinatie met de 'LAAD OP ADRES'-mogelijkheid, om uw subroutinebibliotheek te linken.

## **W – WRITE of SCHRIJF**

Een commando om teksten met de HERBYMON te kunnen plaatsen. Dit bespaart u erg veel tijd, omdat u anders met een ASCII-tabel en het :-commando moet knoeien. Het gaat zo:

.W [startadres] [string]

Bijvoorbeeld:

.W D800 maximaal 16

TIP: kies altijd vaste gebieden voor uw teksten, aangezien de combinatie met machinecode anders een grote puinhoop kan worden!

## **X – EXIT of VERLATEN VAN DE HERBYMON**

U gaat weer netjes terug naar BASIC. Om te laden, op te slaan, of om een combinatie BASIC-machinecode te testen. Gewoon:

.X

## **! – INTERNE REGISTERS**

Als de HERBYMON opstart, worden alle registers intern weggeschreven. Dit gebeurt ook na de terugkeer uit een subroutine. Het !-commando gebruikt u om te kijken hoe het ‘met ze gaat’:

.!

TIP: handig VOORDAT u het \*-commando gebruikt.

## # – DECIMAAL

U krijgt het binaire en het hexadecimale broertje van een decimale waarde (van maximaal 5 cijfers) te zien:

.# [decimaal word]

Bijvoorbeeld:

.# 01234

## \$ – HEXADECIMAAL

Display van het binaire en decimale equivalent van een hexadecimale waarde (lengte maximaal 4 tekens):

.\$ [hexadecimaal word]

Bijvoorbeeld:

.\$ 1234

## % – BINAIR

Conversie naar decimaal en hexadecimaal van een binaire waarde (lengte maximaal 16 tekens):

.% [binaire waarde]

Bijvoorbeeld:

.% 1010101010101010

## & – AND

De logische AND van twee words:

.& [word] [word]

Bijvoorbeeld:

.& 1111 FFFF

TIP: met één byte wordt dit bijvoorbeeld: .& 0011 00FF

#### \* – AANPASSEN VAN DE INTERNE REGISTERS

U verandert de interne registers (eerst het !-commando geven):

.\* [SP] [IY] [IX] [HL] [DE] [BC] [A]

Bijvoorbeeld:

.\* 0000 1111 2222 3333 4444 5555 66

TIP: een prima mogelijkheid om een subroutine te controleren of om een sprong goed te laten starten.

#### + – SOM

U krijgt de som van twee hexadecimale waarden:

.+ [word] [word]

Bijvoorbeeld:

.+ 1010 2020

#### -- VERSCHIL

U krijgt het verschil van twee hexadecimale waarden:

.- [word] [word]

Bijvoorbeeld:

. - 2020 1010

### @ - OR

De logische OR van twee words:

./@ [word] [word]

Bijvoorbeeld:

./@ FFFF 0F0F

### / - XOR

De logische XOR van twee words:

./ [word] [word]

Bijvoorbeeld:

./ FFFF 0F0F

TIP: de XOR-instructie is een krachtig maar zeer ondoorzichtig wapen. Wat experimenteren is dan ook vereist.

# Appendix A

## De Z80-instructieset

De instructies staan in alfabetische volgorde en hebben de in HERBYMON gebruikte syntax

[cs]MNEMONIC	HEX	FLAGS	Leg.	Cycli	TIJD[rm] [*]0,28 [gm]sec
		S Z H N P C			
ADC A,(HL)	8E	R R R O R R	*	2	7
ADC A,(IX+&oo)	DD 8E oo	R R R O R R	*	5	19
ADC A,(IY+&oo)	FD 8E oo	R R R O R R	*	5	19
ADC A,A	BF	R R R O R R	*	1	4
ADC A,B	88	R R R O R R	*	1	4
ADC A,C	89	R R R O R R	*	1	4
ADC A,D	8A	R R R O R R	*	1	4
ADC A,E	8B	R R R O R R	*	1	4
ADC A,H	8C	R R R O R R	*	1	4
ADC A,L	8D	R R R O R R	*	1	4
ADC A,\$bb	CE bb	R R R O R R	*	2	7
ADC HL,BC	ED 4A	R R R O R R	*	4	15
ADC HL,DE	ED 5A	R R R O R R	*	4	15
ADC HL,HL	ED 6A	R R R O R R	*	4	15
ADC HL,SP	ED 7A	R R R O R R	*	4	15
ADD A,(HL)	86	R R R O R R	*	2	7
ADD A,(IX+&oo)	DD 86 oo	R R R O R R	*	5	19
ADD A,(IY+&oo)	FD 86 oo	R R R O R R	*	5	19
ADD A,A	87	R R R O R R	*	1	4
ADD A,B	80	R R R O R R	*	1	4
ADD A,C	81	R R R O R R	*	1	4
ADD A,D	82	R R R O R R	*	1	4
ADD A,E	83	R R R O R R	*	1	4
ADD A,H	84	R R R O R R	*	1	4
ADD A,L	85	R R R O R R	*	1	4

ADD A,\$bb	C6 bb	R R R 0 R R	*	2	7
ADD HL,BC	09	-- R 0 - R	*	3	11
ADD HL,DE	19	-- R 0 - R	*	3	11
ADD HL,HL	29	-- R 0 - R	*	3	11
ADD HL,SP	39	-- R 0 - R	*	3	11
ADD IX,BC	DD 09	-- R 0 - R	*	4	15
ADD IX,DE	DD 19	-- R 0 - R	*	4	15
ADD IX,IX	DD 29	-- R 0 - R	*	4	15
ADD IX,SP	DD 39	-- R 0 - R	*	4	15
ADD IY,BC	FD 09	-- R 0 - R	*	4	15
ADD IY,DE	FD 19	-- R 0 - R	*	4	15
ADD IY,IY	FD 29	-- R 0 - R	*	4	15
ADD IY,SP	FD 39	-- R 0 - R	*	4	15
AND (HL)	A6	R R 1 0 R 0	*	2	7
AND (IX+&oo)	DD A6 oo	R R 1 0 R 0	*	5	19
AND (IY+&oo)	FD A6 oo	R R 1 0 R 0	*	5	19
AND A	A7	R R 1 0 R 0	*	1	4
AND B	A0	R R 1 0 R 0	*	1	4
AND C	A1	R R 1 0 R 0	*	1	4
AND D	A2	R R 1 0 R 0	*	1	4
AND E	A3	R R 1 0 R 0	*	1	4
AND H	A4	R R 1 0 R 0	*	1	4
AND L	A5	R R 1 0 R 0	*	1	4
AND \$bb	E6 bb	R R 1 0 R 0	*	2	7
BIT 0,(HL)	CB 46	- R 1 0 --	*	3	12
BIT 0,(IX+&oo)	DD CB oo 46	- R 1 0 --	*	5	20
BIT 0,(IY+&oo)	FD CB oo 46	- R 1 0 --	*	5	20
BIT 0,A	CB 47	- R 1 0 --	*	2	8
BIT 0,B	CB 40	- R 1 0 --	*	2	8
BIT 0,C	CB 41	- R 1 0 --	*	2	8
BIT 0,D	CB 42	- R 1 0 --	*	2	8
BIT 0,E	CB 43	- R 1 0 --	*	2	8
BIT 0,H	CB 44	- R 1 0 --	*	2	8
BIT 0,L	CB 45	- R 1 0 --	*	2	8
BIT 1,(HL)	CB 4E	- R 1 0 --	*	3	12
BIT 1,(IX+&oo)	DD CB oo 4E	- R 1 0 --	*	5	20
BIT 1,(IY+&oo)	FD CB oo 4E	- R 1 0 --	*	5	20
BIT 1,A	CB 4F	- R 1 0 --	*	2	8

BIT 1,B	CB 48	- R 1 0 - -	*	2	8
BIT 1,C	CB 49	- R 1 0 - -	*	2	8
BIT 1,D	CB 4A	- R 1 0 - -	*	2	8
BIT 1,E	CB 4B	- R 1 0 - -	*	2	8
BIT 1,H	CB 4C	- R 1 0 - -	*	2	8
BIT 1,L	CB 4D	- R 1 0 - -	*	2	8
BIT 2,(HL)	CB 56	- R 1 0 - -	*	3	12
BIT 2,(IX+&oo)	DD CB oo 56	- R 1 0 - -	*	5	20
BIT 2,(IY+&oo)	FD CB oo 56	- R 1 0 - -	*	5	20
BIT 2,A	CB 57	- R 1 0 - -	*	2	8
BIT 2,B	CB 50	- R 1 0 - -	*	2	8
BIT 2,C	CB 51	- R 1 0 - -	*	2	8
BIT 2,D	CB 52	- R 1 0 - -	*	2	8
BIT 2,E	CB 53	- R 1 0 - -	*	2	8
BIT 2,H	CB 54	- R 1 0 - -	*	2	8
BIT 2,L	CB 55	- R 1 0 - -	*	2	8
BIT 3,(HL)	CB 5E	- R 1 0 - -	*	3	12
BIT 3,(IX+&oo)	DD CB oo 5E	- R 1 0 - -	*	5	20
BIT 3,(IY+&oo)	FD CB oo 5E	- R 1 0 - -	*	5	20
BIT 3,A	CB 5F	- R 1 0 - -	*	2	8
BIT 3,B	CB 58	- R 1 0 - -	*	2	8
BIT 3,C	CB 59	- R 1 0 - -	*	2	8
BIT 3,D	CB 5A	- R 1 0 - -	*	2	8
BIT 3,E	CB 5B	- R 1 0 - -	*	2	8
BIT 3,H	CB 5C	- R 1 0 - -	*	2	8
BIT 3,L	CB 5D	- R 1 0 - -	*	2	8
BIT 4,(HL)	CB 66	- R 1 0 - -	*	3	12
BIT 4,(IX+&oo)	DD CB oo 66	- R 1 0 - -	*	5	20
BIT 4,(IY+&oo)	FD CB oo 66	- R 1 0 - -	*	5	20
BIT 4,A	CB 67	- R 1 0 - -	*	2	8
BIT 4,B	CB 60	- R 1 0 - -	*	2	8
BIT 4,C	CB 61	- R 1 0 - -	*	2	8
BIT 4,D	CB 62	- R 1 0 - -	*	2	8
BIT 4,E	CB 63	- R 1 0 - -	*	2	8
BIT 4,H	CB 64	- R 1 0 - -	*	2	8
BIT 4,L	CB 65	- R 1 0 - -	*	2	8
BIT 5,(HL)	CB 6E	- R 1 0 - -	*	3	12

BIT 5,(IX+&oo)	DD CB oo 6E	- R 1 0 - -	*	5	20
BIT 5,(IY+&oo)	FD CB oo 6E	- R 1 0 - -	*	5	20
BIT 5,A	CB 6F	- R 1 0 - -	*	2	8
BIT 5,B	CB 68	- R 1 0 - -	*	2	8
BIT 5,C	CB 69	- R 1 0 - -	*	2	8
BIT 5,D	CB 6A	- R 1 0 - -	*	2	8
BIT 5,E	CB 6B	- R 1 0 - -	*	2	8
BIT 5,H	CB 6C	- R 1 0 - -	*	2	8
BIT 5,L	CB 6D	- R 1 0 - -	*	2	8
BIT 6,(HL)	CB 76	- R 1 0 - -	*	3	12
BIT 6,(IX+&oo)	DD CB oo 76	- R 1 0 - -	*	5	20
BIT 6,(IY+&oo)	FD CB oo 76	- R 1 0 - -	*	5	20
BIT 6,A	CB 77	- R 1 0 - -	*	2	8
BIT 6,B	CB 70	- R 1 0 - -	*	2	8
BIT 6,C	CB 71	- R 1 0 - -	*	2	8
BIT 6,D	CB 72	- R 1 0 - -	*	2	8
BIT 6,E	CB 73	- R 1 0 - -	*	2	8
BIT 6,H	CB 74	- R 1 0 - -	*	2	8
BIT 6,L	CB 75	- R 1 0 - -	*	2	8
BIT 7,(HL)	CB 7E	- R 1 0 - -	*	3	12
BIT 7,(IX+&oo)	DD CB oo 7E	- R 1 0 - -	*	5	20
BIT 7,(IY+&oo)	FD CB oo 7E	- R 1 0 - -	*	5	20
BIT 7,A	CB 7F	- R 1 0 - -	*	2	8
BIT 7,B	CB 78	- R 1 0 - -	*	2	8
BIT 7,C	CB 79	- R 1 0 - -	*	2	8
BIT 7,D	CB 7A	- R 1 0 - -	*	2	8
BIT 7,E	CB 7B	- R 1 0 - -	*	2	8
BIT 7,H	CB 7C	- R 1 0 - -	*	2	8
BIT 7,L	CB 7D	- R 1 0 - -	*	2	8
CALL \$hhll	CD 11 hh	- - - - -	*	5	17
CALL C,\$hhll	DC 11 hh	- - - - -	*	5 (3)	17 (10)
CALL M,\$hhll	FC 11 hh	- - - - -	*	5 (3)	17 (10)
CALL NC,\$hhll	D4 11 hh	- - - - -	*	5 (3)	17 (10)
CALL NZ,\$hhll	C4 11 hh	- - - - -	*	5 (3)	17 (10)
CALL P,\$hhll	F4 11 hh	- - - - -	*	5 (3)	17 (10)
CALL PE,\$hhll	EC 11 hh	- - - - -	*	5 (3)	17 (10)
CALL PO,\$hhll	E4 11 hh	- - - - -	*	5 (3)	17 (10)
CALL Z,\$hhll	CC 11 hh	- - - - -	*	5 (3)	17 (10)

CCF	3F	- - C 0 - R	*	1	4
CP (HL)	BE	R R R 1 R R	*	2	7
CP (IX+&oo)	DD BE oo	R R R 1 R R	*	5	19
CP (IY+&oo)	FD BE oo	R R R 1 R R	*	5	19
CP A	BF	R R R 1 R R	*	1	4
CP B	B9	R R R 1 R R	*	1	4
CP C	B9	R R R 1 R R	*	1	4
CP D	BA	R R R 1 R R	*	1	4
CP E	BB	R R R 1 R R	*	1	4
CP H	BC	R R R 1 R R	*	1	4
CP L	BD	R R R 1 R R	*	1	4
CP \$bb	FE bb	R R R 1 R R	*	2	7
CPD	ED A9	R R R 1 R -	*	4	16
CPDR	ED B9	R R R 1 R -	*	4 (5)	16 (21)
CPI	ED A1	R R R 1 R -	*	4	16
CPIR	ED B1	R R R 1 R -	*	4 (5)	16 (21)
CPL	2F	- - 1 1 - -	*	1	4
DAA	27	R R R - R R	*	1	4
DEC (HL)	35	R R R 1 R -	*	3	11
DEC (IX+&oo)	DD 35 oo	R R R 1 R -	*	6	23
DEC (IY+&oo)	FD 35 oo	R R R 1 R -	*	6	23
DEC A	3D	R R R 1 R -	*	1	4
DEC B	05	R R R 1 R -	*	1	4
DEC BC	0B	- - - - -	*	1	6
DEC C	0D	R R R 1 R -	*	1	4
DEC D	15	R R R 1 R -	*	1	4
DEC DE	1B	- - - - -	*	1	6
DEC E	1D	R R R 1 R -	*	1	4
DEC H	25	R R R 1 R -	*	1	4
DEC HL	2B	- - - - -	*	1	6
DEC IX	DD 2B	- - - - -	*	2	10
DEC IY	FD 2B	- - - - -	*	2	10
DEC L	2D	R R R 1 R -	*	1	4

DEC SP	3B	- - - - -	*	1	6
DI	F3	- - - - -	*	1	4
DJNZ @rrrr	10 rr	- - - - -	*	3 (2)	13 (8)
EI	FB	- - - - -	*	1	4
EX (SP),HL	E3	- - - - -	*	5	19
EX (SP),IX	DD E3	- - - - -	*	6	23
EX (SP),IY	FD E3	- - - - -	*	6	23
EX AF,AF'	08	R R R R R R	*	1	4
EX DE,HL	EB	- - - - -	*	1	4
EXX	D9	- - - - -	*	1	4
HALT	76	- - - - -	*	1	4
IM 0	ED 46	- - - - -	*	2	8
IM 1	ED 56	- - - - -	*	2	8
IM 2	ED 5E	- - - - -	*	2	8
IN A,(C)	ED 78	R R R O R -	*	3	12
IN A,(\$bb)	DB bb	- - - - -	*	3	11
IN B,(C)	ED 40	R R R O R -	*	3	12
IN C,(C)	ED 48	R R R O R -	*	3	12
IN D,(C)	ED 50	R R R O R -	*	3	12
IN E,(C)	ED 58	R R R O R -	*	3	12
IN F,(C)	ED 70	R R R O R -	*	3	12
IN H,(C)	ED 60	R R R O R -	*	3	12
IN L,(C)	ED 68	R R R O R -	*	3	12
INC (HL)	34	R R R O R -	*	3	11
INC (IX+\$oo)	DD 34 oo	R R R O R -	*	6	23
INC (IY+\$oo)	FD 34 oo	R R R O R -	*	6	23
INC A	3C	R R R O R -	*	1	4
INC B	04	R R R O R -	*	1	4
INC BC	03	- - - - -	*	1	6
INC C	0C	R R R O R -	*	1	4

INC D	14	R R R O R -	*	1	4
INC DE	13	- - - - -	*	1	6
INC E	1C	R R R O R -	*	1	4
INC H	24	R R R O R -	*	1	4
INC HL	23	- - - - -	*	1	6
INC IX	DD 23	- - - - -	*	2	10
INC IY	FD 23	- - - - -	*	2	10
INC L	2C	R R R O R -	*	1	4
INC SP	33	- - - - -	*	1	6
IND	ED AA	- R - 1 - -	*	4	16
INDR	ED BA	- 1 - 1 - -	*	4 (5)	16 (21)
INI	ED A2	- R - 1 - -	*	4	16
INIR	ED B2	- 1 - 1 - -	*	4 (5)	16 (21)
JP (HL)	E9	- - - - -	*	1	4
JP (IX)	DD E9	- - - - -	*	2	8
JP (IY)	FD E9	- - - - -	*	2	8
JP \$hhll	C3 11 hh	- - - - -	*	3	10
JP C,\$hhll	DA 11 hh	- - - - -	*	3	10
JP M,\$hhll	FA 11 hh	- - - - -	*	3	10
JP NC,\$hhll	D2 11 hh	- - - - -	*	3	10
JP NZ,\$hhll	C2 11 hh	- - - - -	*	3	10
JP P,\$hhll	F2 11 hh	- - - - -	*	3	10
JP PE,\$hhll	EA 11 hh	- - - - -	*	3	10
JP PO,\$hhll	E2 11 hh	- - - - -	*	3	10
JP Z,\$hhll	CA 11 hh	- - - - -	*	3	10
JR @rrrr	18 rr	- - - - -	*	3	12
JR C,@rrrr	38 rr	- - - - -	*	3 (2)	12 (7)
JR NC,@rrrr	30 rr	- - - - -	*	3 (2)	12 (7)
JR NZ,@rrrr	20 rr	- - - - -	*	3 (2)	12 (7)
JR Z,@rrrr	28 rr	- - - - -	*	3 (2)	12 (7)
LD (BC),A	02	- - - - -	*	2	7
LD (DE),A	12	- - - - -	*	2	7
LD (HL),A	77	- - - - -	*	2	7
LD (HL),B	70	- - - - -	*	2	7

LD (HL),C	71	- - - - -	*	2	7
LD (HL),D	72	- - - - -	*	2	7
LD (HL),E	73	- - - - -	*	2	7
LD (HL),H	74	- - - - -	*	2	7
LD (HL),L	75	- - - - -	*	2	7
LD (HL),\$bb	36 bb	- - - - -	*	3	10
LD (IX+&oo),A	DD 77 oo	- - - - -	*	5	19
LD (IX+&oo),B	DD 70 oo	- - - - -	*	5	19
LD (IX+&oo),C	DD 71 oo	- - - - -	*	5	19
LD (IX+&oo),D	DD 72 oo	- - - - -	*	5	19
LD (IX+&oo),E	DD 73 oo	- - - - -	*	5	19
LD (IX+&oo),H	DD 74 oo	- - - - -	*	5	19
LD (IX+&oo),L	DD 75 oo	- - - - -	*	5	19
LD (IX+&oo),\$bb	DD 36 oo bb	- - - - -	*	5	19
LD (IY+&oo),A	FD 77 oo	- - - - -	*	5	19
LD (IY+&oo),B	FD 70 oo	- - - - -	*	5	19
LD (IY+&oo),C	FD 71 oo	- - - - -	*	5	19
LD (IY+&oo),D	FD 72 oo	- - - - -	*	5	19
LD (IY+&oo),E	FD 73 oo	- - - - -	*	5	19
LD (IY+&oo),H	FD 74 oo	- - - - -	*	5	19
LD (IY+&oo),L	FD 75 oo	- - - - -	*	5	19
LD (IY+&oo),\$bb	FD 36 oo bb	- - - - -	*	5	19
LD (\$hhll),A	32 11 hh	- - - - -	*	4	13
LD (\$hhll),BC	ED 43 11 hh	- - - - -	*	6	20
LD (\$hhll),DE	ED 53 11 hh	- - - - -	*	6	20
LD (\$hhll),HL	22 11 hh	- - - - -	*	5	16
LD (\$hhll),IX	DD 22 11 hh	- - - - -	*	6	20
LD (\$hhll),IY	FD 22 11 hh	- - - - -	*	6	20
LD (\$hhll),SP	ED 73 11 hh	- - - - -	*	6	20
LD A,(BC)	0A	- - - - -	*	2	7
LD A,(DE)	1A	- - - - -	*	2	7
LD A,(HL)	7E	- - - - -	*	2	7
LD A,(IX+&oo)	DD 7E oo	- - - - -	*	5	19
LD A,(IY+&oo)	FD 7E oo	- - - - -	*	5	19
LD A,\$hhll)	3A 11 hh	- - - - -	*	4	13
LD A,A	7F	- - - - -	*	1	4
LD A,B	78	- - - - -	*	1	4
LD A,C	79	- - - - -	*	1	4
LD A,D	7A	- - - - -	*	1	4
LD A,E	7B	- - - - -	*	1	4
LD A,H	7C	- - - - -	*	1	4

LD A,I	ED 57	R R O O R -	*	2	9
LD A,L	7D	- - - - -	*	1	4
LD A,\$bb	3E bb	- - - - -	*	2	7
LD A,R	ED 5F	R R O O R -	*	2	9
LD B,(HL)	46	- - - - -	*	2	7
LD B,(IX+&oo)	DD 46 oo	- - - - -	*	5	19
LD B,(IY+&oo)	FD 46 oo	- - - - -	*	5	19
LD B,A	47	- - - - -	*	1	4
LD B,B	40	- - - - -	*	1	4
LD B,C	41	- - - - -	*	1	4
LD B,D	42	- - - - -	*	1	4
LD B,E	43	- - - - -	*	1	4
LD B,H	44	- - - - -	*	1	4
LD B,L	45	- - - - -	*	1	4
LD B,\$bb	06 bb	- - - - -	*	2	7
LD BC,(\$hhll)	ED 4B 11 hh	- - - - -	*	6	20
LD BC,\$hhll	01 11 hh	- - - - -	*	3	10
LD C,(HL)	4E	- - - - -	*	2	7
LD C,(IX+&oo)	DD 4E oo	- - - - -	*	5	19
LD C,(IY+&oo)	FD 4E oo	- - - - -	*	5	19
LD C,A	4F	- - - - -	*	1	4
LD C,B	48	- - - - -	*	1	4
LD C,C	49	- - - - -	*	1	4
LD C,D	4A	- - - - -	*	1	4
LD C,E	4B	- - - - -	*	1	4
LD C,H	4C	- - - - -	*	1	4
LD C,L	4D	- - - - -	*	1	4
LD C,\$bb	0E bb	- - - - -	*	2	7
LD D,(HL)	56	- - - - -	*	2	7
LD D,(IX+&oo)	DD 56 oo	- - - - -	*	5	19
LD D,(IY+&oo)	FD 56 oo	- - - - -	*	5	19
LD D,A	57	- - - - -	*	1	4
LD D,B	50	- - - - -	*	1	4
LD D,C	51	- - - - -	*	1	4
LD D,D	52	- - - - -	*	1	4
LD D,E	53	- - - - -	*	1	4
LD D,H	54	- - - - -	*	1	4
LD D,L	55	- - - - -	*	1	4
LD D,\$bb	16 bb	- - - - -	*	2	7
LD DE,(\$hhll)	ED 5B 11 hh	- - - - -	*	6	20
LD DE,\$hhll	11 11 hh	- - - - -	*	3	10

LD E,(HL)	SE	- - - - -	*	2	7
LD E,(IX+&oo)	DD SE oo	- - - - -	*	5	19
LD E,(IY+&oo)	FD SE oo	- - - - -	*	5	19
LD E,A	5F	- - - - -	*	1	4
LD E,B	58	- - - - -	*	1	4
LD E,C	59	- - - - -	*	1	4
LD E,D	5A	- - - - -	*	1	4
LD E,E	5B	- - - - -	*	1	4
LD E,H	5C	- - - - -	*	1	4
LD E,L	5D	- - - - -	*	1	4
LD E,\$bb	1E bb	- - - - -	*	2	7
LD H,(HL)	66	- - - - -	*	2	7
LD H,(IX+&oo)	DD 66 oo	- - - - -	*	5	19
LD H,(IY+&oo)	FD 66 oo	- - - - -	*	5	19
LD H,A	67	- - - - -	*	1	4
LD H,B	60	- - - - -	*	1	4
LD H,C	61	- - - - -	*	1	4
LD H,D	62	- - - - -	*	1	4
LD H,E	63	- - - - -	*	1	4
LD H,H	64	- - - - -	*	1	4
LD H,L	65	- - - - -	*	1	4
LD H,\$bb	26 bb	- - - - -	*	2	7
LD HL,(\$hhll)	2A 11 hh	- - - - -	*	6	20
LD HL,\$hhll	21 11 hh	- - - - -	*	3	10
LD I,A	ED 47	- - - - -	*	2	9
LD IX,(\$hhll)	DD 2A 11 hh	- - - - -	*	6	20
LD IX,\$hhll	DD 21 11 hh	- - - - -	*	4	14
LD IY,(\$hhll)	FD 2A 11 hh	- - - - -	*	6	20
LD IY,\$hhll	FD 21 11 hh	- - - - -	*	4	14
LD L,(HL)	6E	- - - - -	*	2	7
LD L,(IX+&oo)	DD 6E oo	- - - - -	*	5	19
LD L,(IY+&oo)	FD 6E oo	- - - - -	*	5	19
LD L,A	6F	- - - - -	*	1	4
LD L,B	68	- - - - -	*	1	4
LD L,C	69	- - - - -	*	1	4
LD L,D	6A	- - - - -	*	1	4
LD L,E	6B	- - - - -	*	1	4
LD L,H	6C	- - - - -	*	1	4
LD L,L	6D	- - - - -	*	1	4
LD L,\$bb	2E bb	- - - - -	*	2	7
LD R,A	ED 4F	- - - - -	*	2	9

LD SP,(\$hh11)	ED 7B 11 hh	- - - - -	*	6	20
LD SP,\$hh11	31 11 hh	- - - - -	*	3	10
LD SP,HL	F9	- - - - -	*	1	6
LD SP,IX	DD F9	- - - - -	*	2	10
LD SP,IY	FD F9	- - - - -	*	2	10
LDD	ED AB	- - 0 0 R -	*	4	16
LDDR	ED B8	- - 0 0 R -	*	5 (4)	21 (16)
LDI	ED A0	- - 0 0 R -	*	4	16
LDIR	ED B0	- - 0 0 R -	*	5 (4)	21 (16)
NEG	ED 44	R R R 1 R R	*	2	8
NDP	00	- - - - -	*	1	4
OR (HL)	B6	R R 0 0 R 0	*	2	7
OR (IX+\$oo)	DD B6 oo	R R 0 0 R 0	*	5	19
OR (IY+\$oo)	FD B6 oo	R R 0 0 R 0	*	5	19
OR A	B7	R R 0 0 R 0	*	1	4
OR B	B0	R R 0 0 R 0	*	1	4
OR C	B1	R R 0 0 R 0	*	1	4
OR D	B2	R R 0 0 R 0	*	1	4
OR E	B3	R R 0 0 R 0	*	1	4
OR H	B4	R R 0 0 R 0	*	1	4
OR L	B5	R R 0 0 R 0	*	1	4
OR \$bb	F6 bb	R R 0 0 R 0	*	2	7
OTDR	ED B8	- 1 - 1 - -	*	4 (5)	16 (21)
OTIR	ED B3	- 1 - 1 - -	*	4 (5)	16 (21)
OUT (C),A	ED 79	- - - - -	*	3	12
OUT (C),B	ED 41	- - - - -	*	3	12
OUT (C),C	ED 49	- - - - -	*	3	12
OUT (C),D	ED 51	- - - - -	*	3	12
OUT (C),E	ED 59	- - - - -	*	3	12
OUT (C),H	ED 61	- - - - -	*	3	12
OUT (C),L	ED 69	- - - - -	*	3	12

OUT (\$bb),A	D3 bb	- - - - -	*	3	11
OUTD	ED AB	- R - 1 - -	*	4	16
OUTI	ED A3	- R - 1 - -	*	4	16
POP AF	F1	R R R R R R	*	3	10
POP BC	C1	- - - - -	*	3	10
POP DE	D1	- - - - -	*	3	10
POP HL	E1	- - - - -	*	3	10
POP IX	DD E1	- - - - -	*	4	14
POP IY	FD E1	- - - - -	*	4	14
PUSH AF	F5	- - - - -	*	3	11
PUSH BC	C5	- - - - -	*	3	11
PUSH DE	D5	- - - - -	*	3	11
PUSH HL	E5	- - - - -	*	3	11
PUSH IX	DD E5	- - - - -	*	4	15
PUSH IY	FD E5	- - - - -	*	4	15
RES 0,(HL)	CB 86	- - - - -	*	4	15
RES 0,(IX+&oo)	DD CB oo 86	- - - - -	*	6	23
RES 0,(IY+&oo)	FD CB oo 86	- - - - -	*	6	23
RES 0,A	CB 87	- - - - -	*	2	8
RES 0,B	CB 80	- - - - -	*	2	8
RES 0,C	CB 81	- - - - -	*	2	8
RES 0,D	CB 82	- - - - -	*	2	8
RES 0,E	CB 83	- - - - -	*	2	8
RES 0,H	CB 84	- - - - -	*	2	8
RES 0,L	CB 85	- - - - -	*	2	8
RES 1,(HL)	CB 8E	- - - - -	*	4	15
RES 1,(IX+&oo)	DD CB oo 8E	- - - - -	*	6	23
RES 1,(IY+&oo)	FD CB oo 8E	- - - - -	*	6	23
RES 1,A	CB 8F	- - - - -	*	2	8
RES 1,B	CB 88	- - - - -	*	2	8
RES 1,C	CB 89	- - - - -	*	2	8
RES 1,D	CB 8A	- - - - -	*	2	8
RES 1,E	CB 8B	- - - - -	*	2	8
RES 1,H	CB 8C	- - - - -	*	2	8
RES 1,L	CB 8D	- - - - -	*	2	8

RES 2,(HL)	CB 96	- - - - -	*	4	15
RES 2,(IX+&oo)	DD CB oo 96	- - - - -	*	6	23
RES 2,(IY+&oo)	FD CB oo 96	- - - - -	*	6	23
RES 2,A	CB 97	- - - - -	*	2	8
RES 2,B	CB 90	- - - - -	*	2	8
RES 2,C	CB 91	- - - - -	*	2	8
RES 2,D	CB 92	- - - - -	*	2	8
RES 2,E	CB 93	- - - - -	*	2	8
RES 2,H	CB 94	- - - - -	*	2	8
RES 2,L	CB 95	- - - - -	*	2	8
RES 3,(HL)	CB 9E	- - - - -	*	4	15
RES 3,(IX+&oo)	DD CB oo 9E	- - - - -	*	6	23
RES 3,(IY+&oo)	FD CB oo 9E	- - - - -	*	6	23
RES 3,A	CB 9F	- - - - -	*	2	8
RES 3,B	CB 98	- - - - -	*	2	8
RES 3,C	CB 99	- - - - -	*	2	8
RES 3,D	CB 9A	- - - - -	*	2	8
RES 3,E	CB 9B	- - - - -	*	2	8
RES 3,H	CB 9C	- - - - -	*	2	8
RES 3,L	CB 9D	- - - - -	*	2	8
RES 4,(HL)	CB A6	- - - - -	*	4	15
RES 4,(IX+&oo)	DD CB oo A6	- - - - -	*	6	23
RES 4,(IY+&oo)	FD CB oo A6	- - - - -	*	6	23
RES 4,A	CB A7	- - - - -	*	2	8
RES 4,B	CB A0	- - - - -	*	2	8
RES 4,C	CB A1	- - - - -	*	2	8
RES 4,D	CB A2	- - - - -	*	2	8
RES 4,E	CB A3	- - - - -	*	2	8
RES 4,H	CB A4	- - - - -	*	2	8
RES 4,L	CB A5	- - - - -	*	2	8
RES 5,(HL)	CB AE	- - - - -	*	4	15
RES 5,(IX+&oo)	DD CB oo AE	- - - - -	*	6	23
RES 5,(IY+&oo)	FD CB oo AE	- - - - -	*	6	23
RES 5,A	CB AF	- - - - -	*	2	8
RES 5,B	CB A8	- - - - -	*	2	8
RES 5,C	CB A9	- - - - -	*	2	8
RES 5,D	CB AA	- - - - -	*	2	8
RES 5,E	CB AB	- - - - -	*	2	8

RES 5,H	CB AC	- - - - -	*	2	8
RES 5,L	CB AD	- - - - -	*	2	8
RES 6,(HL)	CB B6	- - - - -	*	4	15
RES 6,(IX+&oo)	DD CB oo B6	- - - - -	*	6	23
RES 6,(IY+&oo)	FD CB oo B6	- - - - -	*	6	23
RES 6,A	CB B7	- - - - -	*	2	8
RES 6,B	CB B0	- - - - -	*	2	8
RES 6,C	CB B1	- - - - -	*	2	8
RES 6,D	CB B2	- - - - -	*	2	8
RES 6,E	CB B3	- - - - -	*	2	8
RES 6,H	CB B4	- - - - -	*	2	8
RES 6,L	CB B5	- - - - -	*	2	8
RES 7,(HL)	CB BE	- - - - -	*	4	15
RES 7,(IX+&oo)	DD CB oo BE	- - - - -	*	6	23
RES 7,(IY+&oo)	FD CB oo BE	- - - - -	*	6	23
RES 7,A	CB BF	- - - - -	*	2	8
RES 7,B	CB B8	- - - - -	*	2	8
RES 7,C	CB B9	- - - - -	*	2	8
RES 7,D	CB BA	- - - - -	*	2	8
RES 7,E	CB BB	- - - - -	*	2	8
RES 7,H	CB BC	- - - - -	*	2	8
RES 7,L	CB BD	- - - - -	*	2	8
RET	C9	- - - - -	*	3	10
RET C	D8	- - - - -	*	3 (1)	11 (5)
RET M	FB	- - - - -	*	3 (1)	11 (5)
RET NC	D0	- - - - -	*	3 (1)	11 (5)
RET NZ	D0	- - - - -	*	3 (1)	11 (5)
RET P	F0	- - - - -	*	3 (1)	11 (5)
RET PE	E8	- - - - -	*	3 (1)	11 (5)
RET PO	E0	- - - - -	*	3 (1)	11 (5)
RET Z	D8	- - - - -	*	3 (1)	11 (5)
RETI	ED 4D	- - - - -	*	4	14
RETN	ED 45	- - - - -	*	4	14
RL (HL)	CB 16	R R 0 0 R R	*	4	15
RL (IX+&oo)	DD CB oo 16	R R 0 0 R R	*	6	23

RL (IY+&oo)	FD CB oo 16	R R O O R R	*	6	23
RL A	CB 17	R R O O R R	*	2	8
RL B	CB 10	R R O O R R	*	2	8
RL C	CB 11	R R O O R R	*	2	8
RL D	CB 12	R R O O R R	*	2	8
RL E	CB 13	R R O O R R	*	2	8
RL H	CB 14	R R O O R R	*	2	8
RL L	CB 15	R R O O R R	*	2	8
RLA	17	-- 0 0 - R	*	1	4
RLC (HL)	CB 06	R R O O R R	*	4	15
RLC (IX+&oo)	DD CB oo 06	R R O O R R	*	6	23
RLC (IY+&oo)	FD CB oo 06	R R O O R R	*	6	23
RLC A	CB 07	R R O O R R	*	2	8
RLC B	CB 00	R R O O R R	*	2	8
RLC C	CB 01	R R O O R R	*	2	8
RLC D	CB 02	R R O O R R	*	2	8
RLC E	CB 03	R R O O R R	*	2	8
RLC H	CB 04	R R O O R R	*	2	8
RLC L	CB 05	R R O O R R	*	2	8
RLCA	07	-- 0 0 - R	*	1	4
RLD	ED 6F	R R O O R R	*	5	18
RR (HL)	CB 1E	R R O O R R	*	4	15
RR (IX+&oo)	DD CB oo 1E	R R O O R R	*	6	23
RR (IY+&oo)	FD CB oo 1E	R R O O R R	*	6	23
RR A	CB 1F	R R O O R R	*	2	8
RR B	CB 18	R R O O R R	*	2	8
RR C	CB 19	R R O O R R	*	2	8
RR D	CB 1A	R R O O R R	*	2	8
RR E	CB 1B	R R O O R R	*	2	8
RR H	CB 1C	R R O O R R	*	2	8
RR L	CB 1D	R R O O R R	*	2	8
RRA	1F	-- 0 0 - R	*	1	4
RRC (HL)	CB 0E	R R O O R R	*	4	15
RRC (IX+&oo)	DD CB oo 0E	R R O O R R	*	6	23
RRC (IY+&oo)	FD CB oo 0E	R R O O R R	*	6	23

RRC A	CB 0F	R R O O R R	*	2	8
RRC B	CB 08	R R O O R R	*	2	8
RRC C	CB 09	R R O O R R	*	2	8
RRC D	CB 0A	R R O O R R	*	2	8
RRC E	CB 0B	R R O O R R	*	2	8
RRC H	CB 0C	R R O O R R	*	2	8
RRC L	CB 0D	R R O O R R	*	2	8
RRCA	OF	- - 0 0 - R	*	1	4
RRD	ED 67	R R O O R R	*	5	18
RST 00	C7	- - - - -	*	3	11
RST 08	CF	- - - - -	*	3	11
RST 10	D7	- - - - -	*	3	11
RST 18	DF	- - - - -	*	3	11
RST 20	E7	- - - - -	*	3	11
RST 28	EF	- - - - -	*	3	11
RST 30	F7	- - - - -	*	3	11
RST 38	FF	- - - - -	*	3	11
SBC A,(HL)	9E	R R R 1 R R	*	2	7
SBC A,(IX+&oo)	DD 9E oo	R R R 1 R R	*	5	19
SBC A,(IY+&oo)	FD 9E oo	R R R 1 R R	*	5	19
SBC A,A	9F	R R R 1 R R	*	1	4
SBC A,B	98	R R R 1 R R	*	1	4
SBC A,C	99	R R R 1 R R	*	1	4
SBC A,D	9A	R R R 1 R R	*	1	4
SBC A,E	9B	R R R 1 R R	*	1	4
SBC A,H	9C	R R R 1 R R	*	1	4
SBC A,L	9D	R R R 1 R R	*	1	4
SBC A,\$bb	DE bb	R R R 1 R R	*	2	7
SBC HL,BC	ED 42	R R R 1 R R	*	4	15
SBC HL,DE	ED 52	R R R 1 R R	*	4	15
SBC HL,HL	ED 62	R R R 1 R R	*	4	15
SBC HL,SP	ED 72	R R R 1 R R	*	4	15
SCF	37	- - 0 0 - 1	*	1	4
SET 0,(HL)	CB C6	- - - - -	*	4	15
SET 0,(IX+&oo)	DD CB oo C6	- - - - -	*	6	23

SET 0,(IY+&oo)	FD CB oo C6	- - - - -	*	6	23
SET 0,A	CB C7	- - - - -	*	2	8
SET 0,B	CB C0	- - - - -	*	2	8
SET 0,C	CB C1	- - - - -	*	2	8
SET 0,D	CB C2	- - - - -	*	2	8
SET 0,E	CB C3	- - - - -	*	2	8
SET 0,H	CB C4	- - - - -	*	2	8
SET 0,L	CB C5	- - - - -	*	2	8
SET 1,(HL)	CB CE	- - - - -	*	4	15
SET 1,(IX+&oo)	DD CB oo CE	- - - - -	*	6	23
SET 1,(IY+&oo)	FD CB oo CE	- - - - -	*	6	23
SET 1,A	CB CF	- - - - -	*	2	8
SET 1,B	CB C8	- - - - -	*	2	8
SET 1,C	CB C9	- - - - -	*	2	8
SET 1,D	CB CA	- - - - -	*	2	8
SET 1,E	CB CB	- - - - -	*	2	8
SET 1,H	CB CC	- - - - -	*	2	8
SET 1,L	CB CD	- - - - -	*	2	8
SET 2,(HL)	CB D6	- - - - -	*	4	15
SET 2,(IX+&oo)	DD CB oo D6	- - - - -	*	6	23
SET 2,(IY+&oo)	FD CB oo D6	- - - - -	*	6	23
SET 2,A	CB D7	- - - - -	*	2	8
SET 2,B	CB D0	- - - - -	*	2	8
SET 2,C	CB D1	- - - - -	*	2	8
SET 2,D	CB D2	- - - - -	*	2	8
SET 2,E	CB D3	- - - - -	*	2	8
SET 2,H	CB D4	- - - - -	*	2	8
SET 2,L	CB D5	- - - - -	*	2	8
SET 3,(HL)	CB DE	- - - - -	*	4	15
SET 3,(IX+&oo)	DD CB oo DE	- - - - -	*	6	23
SET 3,(IY+&oo)	FD CB oo DE	- - - - -	*	6	23
SET 3,A	CB DF	- - - - -	*	2	8
SET 3,B	CB D8	- - - - -	*	2	8
SET 3,C	CB D9	- - - - -	*	2	8
SET 3,D	CB DA	- - - - -	*	2	8
SET 3,E	CB DB	- - - - -	*	2	8
SET 3,H	CB DC	- - - - -	*	2	8
SET 3,L	CB DD	- - - - -	*	2	8

SET 4,(HL)	CB E6	- - - - -	*	4	15
SET 4,(IX+&oo)	DD CB oo E6	- - - - -	*	6	23
SET 4,(IY+&oo)	FD CB oo E6	- - - - -	*	6	23
SET 4,A	CB E7	- - - - -	*	2	8
SET 4,B	CB E0	- - - - -	*	2	8
SET 4,C	CB E1	- - - - -	*	2	8
SET 4,D	CB E2	- - - - -	*	2	8
SET 4,E	CB E3	- - - - -	*	2	8
SET 4,H	CB E4	- - - - -	*	2	8
SET 4,L	CB E5	- - - - -	*	2	8
SET 5,(HL)	CB EE	- - - - -	*	4	15
SET 5,(IX+&oo)	DD CB oo EE	- - - - -	*	6	23
SET 5,(IY+&oo)	FD CB oo EE	- - - - -	*	6	23
SET 5,A	CB EF	- - - - -	*	2	8
SET 5,B	CB E8	- - - - -	*	2	8
SET 5,C	CB E9	- - - - -	*	2	8
SET 5,D	CB EA	- - - - -	*	2	8
SET 5,E	CB EB	- - - - -	*	2	8
SET 5,H	CB EC	- - - - -	*	2	8
SET 5,L	CB ED	- - - - -	*	2	8
SET 6,(HL)	CB F6	- - - - -	*	4	15
SET 6,(IX+&oo)	DD CB oo F6	- - - - -	*	6	23
SET 6,(IY+&oo)	FD CB oo F6	- - - - -	*	6	23
SET 6,A	CB F7	- - - - -	*	2	8
SET 6,B	CB F0	- - - - -	*	2	8
SET 6,C	CB F1	- - - - -	*	2	8
SET 6,D	CB F2	- - - - -	*	2	8
SET 6,E	CB F3	- - - - -	*	2	8
SET 6,H	CB F4	- - - - -	*	2	8
SET 6,L	CB F5	- - - - -	*	2	8
SET 7,(HL)	CB FE	- - - - -	*	4	15
SET 7,(IX+&oo)	DD CB oo FE	- - - - -	*	6	23
SET 7,(IY+&oo)	FD CB oo FE	- - - - -	*	6	23
SET 7,A	CB FF	- - - - -	*	2	8
SET 7,B	CB FB	- - - - -	*	2	8
SET 7,C	CB F9	- - - - -	*	2	8
SET 7,D	CB FA	- - - - -	*	2	8
SET 7,E	CB FB	- - - - -	*	2	8

SET 7,H	CB FC	- - - - -	*	2	8
SET 7,L	CB FD	- - - - -	*	2	8
SLA (HL)	CB 26	R R O O R R	*	4	15
SLA (IX+&oo)	DD CB oo 26	R R O O R R	*	6	23
SLA (IY+&oo)	FD CB oo 26	R R O O R R	*	6	23
SLA A	CB 27	R R O O R R	*	2	8
SLA B	CB 20	R R O O R R	*	2	8
SLA C	CB 21	R R O O R R	*	2	8
SLA D	CB 22	R R O O R R	*	2	8
SLA E	CB 23	R R O O R R	*	2	8
SLA H	CB 24	R R O O R R	*	2	8
SLA L	CB 25	R R O O R R	*	2	8
SLI (HL)	CB 36	R R O O R R	?	4	15
SLI (IX+&oo)	DD CB oo 36	R R O O R R	?	6	23
SLI (IY+&oo)	FD CB oo 36	R R O O R R	?	6	23
SLI A	CB 37	R R O O R R	?	2	8
SLI B	CB 30	R R O O R R	?	2	8
SLI C	CB 31	R R O O R R	?	2	8
SLI D	CB 32	R R O O R R	?	2	8
SLI E	CB 33	R R O O R R	?	2	8
SLI H	CB 34	R R O O R R	?	2	8
SLI L	CB 35	R R O O R R	?	2	8
SRA (HL)	CB 2E	R R O O R R	*	4	15
SRA (IX+&oo)	DD CB oo 2E	R R O O R R	*	6	23
SRA (IY+&oo)	FD CB oo 2E	R R O O R R	*	6	23
SRA A	CB 2F	R R O O R R	*	2	8
SRA B	CB 28	R R O O R R	*	2	8
SRA C	CB 29	R R O O R R	*	2	8
SRA D	CB 2A	R R O O R R	*	2	8
SRA E	CB 2B	R R O O R R	*	2	8
SRA H	CB 2C	R R O O R R	*	2	8
SRA L	CB 2D	R R O O R R	*	2	8
SRL (HL)	CB 3E	R R O O R R	*	4	15
SRL (IX+&oo)	DD CB oo 3E	R R O O R R	*	6	23
SRL (IY+&oo)	FD CB oo 3E	R R O O R R	*	6	23
SRL A	CB 3F	R R O O R R	*	2	8
SRL B	CB 3B	R R O O R R	*	2	8

SRL C	CB 39	R R O O R R	*	2	8
SRL D	CB 3A	R R O O R R	*	2	8
SRL E	CB 3B	R R O O R R	*	2	8
SRL H	CB 3C	R R O O R R	*	2	8
SRL L	CB 3D	R R O O R R	*	2	8
SUB (HL)	96	R R R 1 R R	*	2	7
SUB (IX+\$oo)	DD 96 oo	R R R 1 R R	*	5	19
SUB (IY+\$oo)	FD 96 oo	R R R 1 R R	*	5	19
SUB A	97	R R R 1 R R	*	1	4
SUB B	90	R R R 1 R R	*	1	4
SUB C	91	R R R 1 R R	*	1	4
SUB D	92	R R R 1 R R	*	1	4
SUB E	93	R R R 1 R R	*	1	4
SUB H	94	R R R 1 R R	*	1	4
SUB L	95	R R R 1 R R	*	1	4
SUB \$bb	D6 bb	R R R 1 R R	*	2	7
XOR (HL)	AE	R R R 1 R R	*	2	7
XOR (IX+\$oo)	DD AE oo	R R R 1 R R	*	5	19
XOR (IY+\$oo)	FD AE oo	R R R 1 R R	*	5	19
XOR A	AF	R R R 1 R R	*	1	4
XOR B	A8	R R R 1 R R	*	1	4
XOR C	A9	R R R 1 R R	*	1	4
XOR D	AA	R R R 1 R R	*	1	4
XOR E	AB	R R R 1 R R	*	1	4
XOR H	AC	R R R 1 R R	*	1	4
XOR L	AD	R R R 1 R R	*	1	4
XOR \$bb	EE bb	R R R 1 R R	*	2	7

[cs]Verklaring van de vreemde tekens:[rm]

0 : flag wordt op nul gezet

1 : flag wordt op één gezet

- : flag wordt niet veranderd

C : carry wordt gekopieerd

R : flag wordt afhankelijk van het bewerkingresultaat veranderd

\* : legale Z80-instructie

? : illegale Z80-instructie (Wel gebruikt binnen de HERBYMON!)

\$bb : byte-waarde

&oo : index/offset-waarde

@rrrr : byte-waarde die een relatieve sprongcalculatie oplevert

\$hhll : word-waarde ('hh' is high-byte / 'll' is low-byte)

# Appendix B

## Karaktercodes

Decimaal	Hex	ASCII-code	RESULTAAT ('ChPut')	ESCAPE-code
0	00	NUL	'niets'	
1	01	SOH		
2	02	STX		
3	03	ETX		
4	04	EOT		
5	05	ENQ		
6	06	ACK		
7	07	BEL	doe 'BEEP'	
8	08	BS	cursor naar links	
9	09	HT	volgende TAB-positie	
10	0A	LF	cursor omhoog	
11	0B	VT		
12	0C	FF	scherm schoon en cursor HOME	
13	0D	CR	cursor zover mogelijk naar links	
14	0E	SO		
15	0F	SI		
16	10	DLE		
17	11	DC1		
18	12	DC2		
19	13	DC3		
20	14	DC4		
21	15	NAK		
22	16	SYN		
23	17	ETB		
24	18	CAN		
25	19	EM		
26	1A	SUB		
27	1B	ESC	naar ESCAPE-groep	
28	1C	FS	cursor naar rechts	
29	1D	GS	cursor naar links	
30	1E	RS	cursor omhoog	
31	1F	US	cursor omlaag	
32	20	SP	spatie	
65	41			cursor omhoog
66	42			cursor omlaag

67	43	cursor naar rechts
68	44	cursor naar links
69	45	wis scherm en cursor HOME
72	48	cursor HOME
74	4A	wis tot einde van scherm
75	4B	wis tot einde van regel
76	4C	voeg regel tussen
77	4D	wis regel
89	59	geef cursor-coördinaten
106	6A	wis scherm en cursor HOME
108	6C	wis regel
120	78	verander cursor
121	79	verander cursor
127	7F DEL	karakter wissen

<b>Symbol</b>									
<b>Code</b>	32 &H20	33 &H21	34 &H22	35 &H23	36 &H24	37 &H25	38 &H26	39 &H27	40 &H28
<b>Symbol</b>									
<b>Code</b>	41 &H29	42 &H2A	43 &H2B	44 &H2C	45 &H2D	46 &H2E	47 &H2F	48 &H30	49 &H31
<b>Symbol</b>									
<b>Code</b>	50 &H32	51 &H33	52 &H34	53 &H35	54 &H36	55 &H37	56 &H38	57 &H39	58 &H3A
<b>Symbol</b>									
<b>Code</b>	59 &H3B	60 &H3C	61 &H3D	62 &H3E	63 &H3F	64 &H40	65 &H41	66 &H42	67 &H43
<b>Symbol</b>									
<b>Code</b>	68 &H44	69 &H45	70 &H46	71 &H47	72 &H48	73 &H49	74 &H4A	75 &H4B	76 &H4C
<b>Symbol</b>									
<b>Code</b>	77 &H4D	78 &H4E	79 &H4F	80 &H50	81 &H51	82 &H52	83 &H53	84 &H54	85 &H55
<b>Symbol</b>									
<b>Code</b>	86 &H56	87 &H57	88 &H58	89 &H59	90 &H5A	91 &H5B	92 &H5C	93 &H5D	94 &H5E
<b>Symbol</b>									
<b>Code</b>	95 &H5F	96 &H60	97 &H61	98 &H62	99 &H63	100 &H64	101 &H65	102 &H66	103 &H67
<b>Symbol</b>									
<b>Code</b>	104 &H68	105 &H69	106 &H6A	107 &H6B	108 &H6C	109 &H6D	110 &H6E	111 &H6F	112 &H70
<b>Symbol</b>									
<b>Code</b>	113 &H71	114 &H72	115 &H73	116 &H74	117 &H75	118 &H76	119 &H77	120 &H78	121 &H79
<b>Symbol</b>									
<b>Code</b>	122 &H7A	123 &H7B	124 &H7C	125 &H7D	126 &H7E	127 &H7F	128 &H80	129 &H81	130 &H82
<b>Symbol</b>									
<b>Code</b>	131 &H83	132 &H84	133 &H85	134 &H86	135 &H87	136 &H88	137 &H89	138 &H8A	139 &H8B
<b>Symbol</b>									
<b>Code</b>	140 &H8C	141 &H8D	142 &H8E	143 &H8F	144 &H90	145 &H91	146 &H92	147 &H93	148 &H94

<b>Symbol</b>									
<b>Code</b>	149 &H95	150 &H96	151 &H97	152 &H98	153 &H99	154 &H9A	155 &H9B	156 &H9C	157 &H9D
<b>Symbol</b>									
<b>Code</b>	158 &H9E	159 &H9F	160 &HA0	161 &HA1	162 &HA2	163 &HA3	164 &HA4	165 &HA5	166 &HA6
<b>Symbol</b>									
<b>Code</b>	167 &HA7	168 &HA8	169 &HA9	170 &HAA	171 &HAB	172 &HAC	173 &HAD	174 &HAE	175 &HAF
<b>Symbol</b>									
<b>Code</b>	176 &HB0	177 &HB1	178 &HB2	179 &HB3	180 &HB4	181 &HB5	182 &HB6	183 &HB7	184 &HB8
<b>Symbol</b>									
<b>Code</b>	185 &HB9	186 &HBA	187 &HBB	188 &HBC	189 &HBD	190 &HBE	191 &HBF	192 &HC0	193 &HC1
<b>Symbol</b>									
<b>Code</b>	194 &HC2	195 &HC3	196 &HC4	197 &HC5	198 &HC6	199 &HC7	200 &HC8	201 &HC9	202 &HCA
<b>Symbol</b>									
<b>Code</b>	203 &HCB	204 &HCC	205 &HCD	206 &HCE	207 &HCF	208 &HD0	209 &HD1	210 &HD2	211 &HD3
<b>Symbol</b>									
<b>Code</b>	212 &HD4	213 &HD5	214 &HD6	215 &HD7	216 &HD8	217 &HD9	218 &HDA	219 &HDB	220 &HDC
<b>Symbol</b>									
<b>Code</b>	221 &HDD	222 &HDE	223 &HDF	224 &HE0	225 &HE1	226 &HE2	227 &HE3	228 &HE4	229 &HE5
<b>Symbol</b>									
<b>Code</b>	230 &HE6	231 &HE7	232 &HE8	233 &HE9	234 &HEA	235 &HEB	236 &HEC	237 &HED	238 &HEE
<b>Symbol</b>									
<b>Code</b>	239 &HEF	240 &HF0	241 &HF1	242 &HF2	243 &HF3	244 &HF4	245 &HF5	246 &HF6	247 &HF7
<b>Symbol</b>									
<b>Code</b>	248 &HF8	249 &HF9	250 &HFA	251 &HFB	252 &HFC	253 &HFD	254 &HFE	255 &HFF	

<b>Symbol</b>									
<b>Code</b>	65 &H41	66 &H42	67 &H43	68 &H44	69 &H45	70 &H46	71 &H47	72 &H48	73 &H49
<b>Symbol</b>									
<b>Code</b>	74 &H4A	75 &H4B	76 &H4C	77 &H4D	78 &H4E	79 &H4F	80 &H50	81 &H51	82 &H52
<b>Symbol</b>									
<b>Code</b>	83 &H53	84 &H54	85 &H55	86 &H56	87 &H57	88 &H58	89 &H59	90 &H5A	91 &H5B
<b>Symbol</b>									
<b>Code</b>	92 &H5C	93 &H5D	94 &H5E	95 &H5F					

# Appendix C

## De memory-map

*De memory-map*

*De memory-map onder HERBYMON, na CLEAR 200, &H9FF*

\$0000 ↗	Operating System & sub ROM
\$4000 ↗	BASIC
\$8000 ↗	BASIC-programma
	Variabelen
	Array-variabelen
	Niet in gebruik
	Z80-stack
\$A000 ↗	Strings
\$C000 ↗	HERBYMON
	Voor eigen gebruik
\$F168 ↗	IO-buffers
\$F380 ↗	Werkgebied

\$0000 ↗	Operating System & sub ROM
\$4000 ↗	BASIC
\$8000 ↗	BASIC-programma
	Variabelen
	Array-variabelen
	Niet in gebruik
	Z80-stack
\$F168 ↗	Strings
\$F380 ↗	IO-buffers
	Werkgebied

# Appendix D

## Kleurentabel

Nummer	Nybble-patroon	Kleur
0	0000	transparant
1	0001	zwart
2	0010	groen
3	0011	lichtgroen
4	0100	donkerblauw
5	0101	lichtblauw
6	0110	donkerrood
7	0111	hemelsblauw
8	1000	rood
9	1001	lichtrood
10	1010	donkergeel
11	1011	lichtgeel
12	1100	donkergroen
13	1101	magenta
14	1110	grijs
15	1111	wit

# Appendix E

## Video-RAM-tabel

	Tekst (48x24)	Tekst (32x24)	High-Res	Multi-Color
Schermtabel	\$0000	\$1800	\$1800	\$0800
Patroontabel	\$0800	\$0000	\$0000	\$0000
Kleurentabel	n.v.t.	\$2000	\$2000	n.v.t.
Sprite-attributentabel	n.v.t.	\$1800	\$1800	\$1800
Sprite-patroontabel	n.v.t.	\$3800	\$3800	\$3800

# Appendix F

## VDP-registers

Naam	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
<b>VDP-statusregister</b>	Frame-F	Sprite-F	Coind-F	Nummer van de '5de' sprite					
<b>Mode-register 0</b>	0	0	0	0	0	0	Mode3	Ext Video	
<b>Mode-register 1</b>	4/16K	Aan/Uit	IE	Mode1	Mode2	0	Sprt-ptr	Sprt-meet	
<b>Mode-register 2</b>	0	0	0	0	Bits 3 t/m 5 voor LSB-namenabel				
<b>Mode-register 3</b>	Basis voor Kleurentabel								
<b>Mode-register 4</b>	0	0	0	0	0	Bits 3-5 voor LSB-karakterset			
<b>Mode-register 5</b>	0	Bit 7-MSB en bit 0 t/m 5 van LSB voor sprite-attributen							
<b>Mode-register 6</b>	0	0	0	0	0	Bits 3-5 voor LSB-sprite-vorm			
<b>Mode-register 7</b>	tekstkleur screen mode 0				border-kleur (transparant)				

L = lees gegevens

S = schrijf gegevens

Overige IO-adresgebieden:

\$00-\$7F - niet gedefinieerd

\$B0-\$CF - systeem

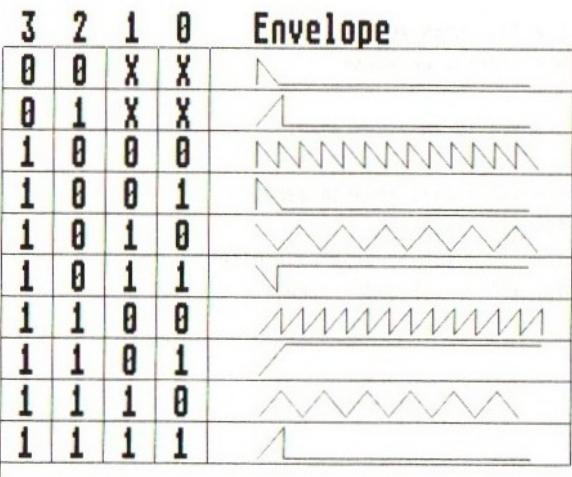
\$D0-D7 - floppy disk-controller

\$D8-\$FF - niet gedefinieerd

## Appendix G PSG-registers

Naam	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Register 0	Frequentie Kanaal A - LSB							
Register 1	n.g.	n.g.	n.g.	n.g.	Frequentie Kanaal A - MSB			
Register 2	Frequentie Kanaal B - LSB							
Register 3	n.g.	n.g.	n.g.	n.g.	Frequentie Kanaal B - MSB			
Register 4	Frequentie Kanaal C - LSB							
Register 5	n.g.	n.g.	n.g.	n.g.	Frequentie Kanaal C - MSB			
Register 6	n.g.	n.g.	n.g.	Ruisfrequentie				
Register 7	10-port A	10-port B	Ruis C	Ruis B	Ruis A	Toon C	Toon B	Toon A
Register 8	n.g.	n.g.	n.g.	Mode A	Amplitude Kanaal A			
Register 9	n.g.	n.g.	n.g.	Mode B	Amplitude Kanaal B			
Register 10	n.g.	n.g.	n.g.	Mode C	Amplitude Kanaal C			
Register 11	Envelope-frequentie LSB							
Register 12	Envelope-frequentie MSB							
Register 13	n.g.	n.g.	n.g.	n.g.	Norm van envelope			
Register 14	Cass-inp	Keyboard	Trigger B	Trigger A	Right-Joy	Left-Joy	Down-Joy	Up-Joy
Register 15	Kanaal-LED	Joystick	Puls Z	Puls 1	(1)	(1)	(1)	(1)

## *De PSG-registers*



## Golfvormen

# Appendix H

## IO-adressen

Device	Adres	Gebruik	Omschrijving	Opmerkingen
RS232C	\$80 \$81 \$81	L/S L S	Data Status Mode	8251A (USART)
Printer	\$90 \$90 \$91	S S S	Strobe Status Data	Centronics
VDP	\$98 \$99  \$99	L/S S L	Data Command/ Address set Status	TMS9129NL
PSG	\$A0 \$A1 \$A2	S S L	Address Latch Data Data	AY-3-8910
PPI	\$A8 \$A9 \$AA \$AB	L/S L/S L/S S	Data-Port A Data-Port B Data-Port C Mode set	8255A

# Appendix I

## De BIOS-sprongtabel

Adres	Naam	Functie	Invoer	Uitvoer
\$0000	ChkRm	Controleert het RAM en selecteert slots	(*)	n.v.t.n.v.t.
\$0008	SynChr	Controleert een BASIC-karakter	HL=adres van karakter	A=karakter / HL=volgend
\$000C	RdSlt	Leest RAM uit opgegeven slot	A=slot ID / HL=adres	A=byte
\$0010	ChrGtr	Verzorgt een volgend BASIC-karakter of -token	HL=adres van karakter	A=karakter / HL=volgend
\$0014	WrSlt	Schrijft door in RAM in opgegeven slot	A=slot ID / HL=adres / E=byte	n.v.t.
\$0018	DutDn	Verzorgt uitvoer naar het huidige device	A=karakter	n.v.t.
\$001C	CaiSlt	Springt naar een subroutine in opgegeven slot	IY=slot ID / II-adres	routine-afhankelijk
\$0020	DCompR	Vergelijkt de registerparen HL en DE	HL / DE	flags eventueel updated
\$0024	EnaSlt	Schakelt een geselecteerd slot permanent in	A=slot ID / HL=adres	n.v.t.
\$0028	GeTypR	Geeft het type van een BASIC-operand	n.v.t.	AF-type
\$0030	CallI0	Springt naar een subroutine in een 'aangelegd' slot	RST 30 / ID-byte / adres-word	routine-afhankelijk
\$0038	KeyInt	Hardware-interrupt en keyboard-scan	(*) n.v.t.	n.v.t.
\$003b	InitIO	Initialiseert IO-devices	(*) n.v.t.	n.v.t.
\$003e	IniFnk	Initialiseert functietoetsen	(*) n.v.t.	n.v.t.
\$0041	DisScr	Zet het scherm 'uit'	n.v.t.	n.v.t.
\$0044	EnaScr	Zet het scherm 'aan'	n.v.t.	n.v.t.
\$0047	WrtVDP	Schrijft in een VDP-register	(*) B=byte / C=VDP register	n.v.t.
\$004A	RdVrm	Haalt een byte uit het video-RAM	HL=adres	A=byte
\$004D	WrtVrm	Schrijft een byte naar het video-RAM	HL=adres / A=byte	n.v.t.
\$0050	SetRd	Initialiseert de VDP voor een leesopdracht	HL=adres	n.v.t.
\$0053	SetWrt	Initialiseert de VDP voor een schrijfopdracht	HL=adres	n.v.t.
\$0056	FilVra	Vult een stuk videogeheugen met een gegeven waarde	HL=adres / BC-lengte / A=byte	n.v.t.
\$0059	LDirVM	Kopieert een stuk videogeheugen naar RAM	HL=bron / DE=object / BC-lengte	n.v.t.
\$005C	LDirVM	Kopieert een stuk RAM naar het videogeheugen	HL=bron / DE=object / BC-lengte	n.v.t.
\$005F	ChgMod	Verzorgt de inschakeling van een gewenste VDP-mode	A=scherm-mode	n.v.t.
\$0062	ChgClr	Veranderd de VDP-kleuren	(*) n.v.t.	n.v.t.
\$0065	NMI	Verzorgt een Non Maskable Interrupt	n.v.t.	n.v.t.
\$0069	ClsSpr	Initialiseert alle sprites	n.v.t.	n.v.t.
\$006C	IniTxt	Initialiseert de VDP voor tekst-mode 40*124	(*) n.v.t.	n.v.t.
\$006F	Init32	Initialiseert de VDP voor tekst-mode 32*124	(*) n.v.t.	n.v.t.
\$0072	IniGrp	Initialiseert de VDP voor high resolution-mode	n.v.t.	n.v.t.
\$0075	IniMit	Initialiseert de VDP voor multicolor-code	n.v.t.	n.v.t.
\$0078	SetTxt	Stelt de VDP in op tekstaarde 40*124	n.v.t.	n.v.t.
\$007B	Set32	Stelt de VDP in op tekstaarde 40I*132	n.v.t.	n.v.t.
\$007E	SetGrp	Stelt de VDP in op high resolution-mode	n.v.t.	n.v.t.
\$0081	SetMit	Stelt de VDP in op multicolor-mode	n.v.t.	n.v.t.
\$0084	CalPat	Geeft het adres van een sprite-patroon	(*) A=sprite ID	HL=adres
\$0087	CalAtr	Geeft het adres van de sprite-attributen	(*) A=sprite ID	HL=adres
\$008A	SSpSiz	Geeft de huidige sprite-grootte	n.v.t.	A=grootte
\$008D	GrpPrt	Plaatst een karakter op het grafische scherm	(*) A=karakter	n.v.t.
\$0090	Giclni	Initialiseert de PSG	(*) n.v.t.	n.v.t.
\$0093	WrtPSG	Schrijft in een PSG-register	A=PSG register / E=byte	n.v.t.
\$0096	RdPSG	Leest een PSG-register uit	A=PSG register	A=byte
\$0099	StrtMx	Start het uitlezen van de muziek	(*) n.v.t.	n.v.t.
\$009C	ChsNS	Geeft de status van de keyboard-buffer	n.v.t.	zero updated
\$009F	ChSet	Wacht op het invoeren van een karakter	(*) n.v.t.	A=karakter (code)
\$00A2	ChPut	Voert een karakter uit naar het scherm	(*) A=karakter (code)	n.v.t.
\$00A5	LptOut	Voert een karakter uit naar de printer	A=karakter (code)	carry updated
\$00A8	LptStt	Controleert de printer	n.v.t.	A en zero updated
\$00Bb	CavChr	Zet karakter met 'grafische header' op	A=karakter	carry en zero updated

\$00AE	PinLin	Leest een bewerkte regel in	n.v.t.	HL=adres / carry updated
\$0081	InLin	Leest een bewerkte regel in	(*) n.v.t.	HL=adres / carry updated
\$0084	QinLin	Zet '?' op het scherm en springt naar INLIN	n.v.t.	HL=adres / carry updated
\$0087	BreakX	Controleert op CTRL-STOP	n.v.t.	carry updated
\$008A	IsCntr	Controleert op SHIFT-STOP	n.v.t.	n.v.t.
\$008D	CkCntr	IsCntr, maar nu door BASIC	n.v.t.	n.v.t.
\$00C0	Beep	Verzorgt een BEEP	n.v.t.	n.v.t.
\$00C3	Cls	Makst het scherm schoon	zero	n.v.t.
\$00C6	PosIt	Zet de cursor op een bepaalde plaats	(*) H-kolom / L=regel	n.v.t.
\$00C9	FnkSb	Kijkt of de tekst-functietoetsen 'aan' staan	n.v.t.	n.v.t.
\$00CC	ErfFnk	Halt de tekst-functietoetsen van het scherm	n.v.t.	n.v.t.
\$00CF	DspFnk	Zet de tekst-functietoetsen op het scherm	n.v.t.	n.v.t.
\$00D2	ToText	Zet het scherm in tekst-mode	n.v.t.	n.v.t.
\$00D5	GtStck	Leest de joystick-stand uit	A=joystick ID	A=richting
\$00D8	GtTrig	Leest de vuurknop-stand uit	A=vuurknop ID	A=vuur (0-niet)
\$00D8	GtPad	Leest de touch tablet-stand uit	A=tablet ID	A=waaarde
\$00DE	GtPdl	Leest de paddle-stand uit	A=paddle ID	A=waaarde
\$00E1	TapIn	Zet cassette-invoer aan en leest header	n.v.t.	carry updated
\$00E4	TapIn	Leest cassettesignaal in	n.v.t.	A=byte / carry updated
\$00E7	TapOf	Zet cassette-invoer uit	n.v.t.	n.v.t.
\$00EA	TapOn	Zet cassette-uivvoer aan en schrijft header	A=header type (0=kort)	carry updated
\$00ED	SchrJt	Schrijft cassettesignaal	A=byte	carry updated
\$00F0	TapOf	Zet cassette-uivvoer uit	n.v.t.	n.v.t.
\$00F3	StMotr	Zet cassette-motor aan/uit	A=actie (0=stop / 1=start)	n.v.t.
\$00F6	LftQ	Geefst de ruimte in de muziekstring	(*) A=arr-nummer	HL=ruimte
\$00F9	PutQ	Plaatst een byte in de muziekstring	(*) A=array-nummer / E=byte	zero updated
\$00FC	RightC	Beweegt 1 pixel naar rechts	n.v.t.	n.v.t.
\$00FF	LeftC	Beweegt 1 pixel naar links	n.v.t.	n.v.t.
\$0102	UpC	Beweegt 1 pixel omhoog	n.v.t.	n.v.t.
\$0105	UpC	UpC indien mogelijk	n.v.t.	carry updated
\$0108	DownC	Beweegt 1 pixel omlaag	n.v.t.	n.v.t.
\$010B	DownC	DownC indien mogelijk	n.v.t.	carry updated
\$010E	ScalXY	Zet de grafische coördinaten op schaal	BC X-coörd. / DE Y-coörd.	carry updated
\$0111	MapXYc	Berekent adres van grafische coördinaat	BC X-coörd. / DE Y-coörd.	HL=adres / A=patroon
\$0114	FetchC	Geefst het adres en maskerpatroon	n.v.t.	HL=adres / A=patroon
\$0117-	StoreC	Plaatst het adres en maskerpatroon	A=masker / HL=adres	n.v.t.
\$011A	SetAtr	Vult het attribuut-byte in	A=kleur	carry updated
\$011D	ReadC	Leest het attribuut-byte van een pixel	n.v.t.	A=kleur
\$0120	SetC	Vult het attribuut-byte van een pixel in	n.v.t.	n.v.t.
\$0123	NSetCX	Vult een aantal pixels horizontaal	HL=teller	n.v.t.
\$0126	GtAspc	Berekent een cirkelverhouding	(*) n.v.t. DE=A\$pc1 / HL=A\$pc2	n.v.t.
\$0129	PntInd	Initialiseert de PAINT-routine	(*) A=kleur	n.v.t.
\$012C	ScanR	Zoekt pixels naar rechts	B-flag / DE=teller	DE=teller / C-flag
\$012F	ScanL	Zoekt pixels naar links	B-flag / DE=teller	DE=teller / C-flag
\$0132	ChgCap	Verandert de status van de CAP-indicator	(*) A=aan/uit (0=uit)	n.v.t.
\$0135	ChgSnd	Wijzigst de key click	(*) A=aan/uit (0=uit)	n.v.t.
\$0138	RSIReg	Leest het primaire slot-register	n.v.t.	A=byte
\$013B	WSIReg	Schrijft naar het primaire slot-register	n.v.t.	n.v.t.
\$013E	RdVDP	Leest het VDP-statusregister	n.v.t.	A=status
\$0141	SnsMat	Leest een rij van de keyboard-matrix	A=regel	A=status
\$0144	PhyIO	Bestuurt de eventueel aanwezige diskdrive	ID-afhankelijk	ID-afhankelijk
\$0147	Format	Initialiseert de eventueel aanwezige diskdrive	ID-afhankelijk	ID-afhankelijk
\$014A	IsFIIO	Bekijkt het eventueel optreden van device-ID	n.v.t.	F=type IO
\$014D	OutDip	Verzorgt geformattede uivvoer naar de printer	(*) A=karakter (code)	n.v.t.
\$0150	GetVCP	Geefst de pointer van de muziek-string	(*) A=kanaal	HL=adres
\$0153	GetVc2	Geefst de pointer van een gewenste muziekvariabele	L=byte	HL=pointer
\$0156	KilBuf	Initialiseert de keyboard-buffer	(*) n.v.t.	n.v.t.
\$0159	CalBAS	Springt naar de BASIC-interpreter	IIX-adres	BASIC-afhankelijk

(\*): de functie informeert en/of verandert het (variabelen)werkgebied

# Appendix J

## Het werkgebied

Adres	Naam	Type	Functie
\$F380	RdPrim	R	Slot-schakeling en uitlezen (primair)
\$F385	WrPrim	R	Slot-schakeling en wegschrijven (primair)
\$F38C	C1Prim	R	Slot-schakeling voor CALSLT
\$F39A	Usr0	A	Adres Usr0 (default 'ILLEGAL FUNCTION CALL')
\$F39C	Usr1	A	Adres Usr1 (default 'ILLEGAL FUNCTION CALL')
\$F39E	Usr2	A	Adres Usr2 (default 'ILLEGAL FUNCTION CALL')
\$F3A0	Usr3	A	Adres Usr3 (default 'ILLEGAL FUNCTION CALL')
\$F3A2	Usr4	A	Adres Usr4 (default 'ILLEGAL FUNCTION CALL')
\$F3A4	Usr5	A	Adres Usr5 (default 'ILLEGAL FUNCTION CALL')
\$F3A6	Usr6	A	Adres Usr6 (default 'ILLEGAL FUNCTION CALL')
\$F3A8	Usr7	A	Adres Usr7 (default 'ILLEGAL FUNCTION CALL')
\$F3AA	Usr8	A	Adres Usr8 (default 'ILLESAL FUNCTION CALL')
\$F3AC	Usr9	A	Adres Usr9 (default 'ILLEGAL FUNCTION CALL')
\$F3AE	LinL40	V	Schermbreedte in 40*24 tekst-mode (default 37)
\$F3AF	LinL32	V	Schermbreedte in 32*24 tekst-mode (default 29)
\$F3B0	LinLen	V	Actuele schermbreedte (wordt tijdens scherm-inschakeling gekopieerd)
\$F3B1			
\$F3B2	ClmLst	V	Ruimte voor PRINT-functie (default 14)
\$F3B3	TxtNam	A	Namentabel (40*24) (default \$0000)
\$F3B5	TxtCol	A	Kleurentabel (40*24) (default \$0000)
\$F3B7	TxtCgp	A	Karakterset (40*24) (default \$0800)
\$F3B9	TxtAtr	A	Sprite-attributen (40*24) (default \$0000)
\$F3BB	TxtPat	A	Sprite-patronen (40*24) (default \$0000)
\$F3BD	T32Nam	A	Namentabel (32*24) (default \$1800)
\$F3BF	T32Col	A	Kleurentabel (32*24) (default \$2000)
\$F3C1	T32Cgp	A	Karakterset (32*24) (default \$0000)
\$F3C3	T32Atr	A	Sprite-attributen (32*24) (default \$1B00)
\$F3C5	T32Pat	A	Sprite-patronen (32*24) (default \$3800)
\$F3C7	GrpNam	A	Namentabel (hires) (default \$1800)
\$F3C9	GrpCol	A	Kleurentabel (hires) (default \$2000)
\$F3CB	GrpCgp	A	Karakterset (hires) (default \$0000)
\$F3CD	GrpAtr	A	Sprite-attributen (hires) (default \$1B00)
\$F3CF	GrpPat	A	Sprite-patronen (high res) (default \$3800)
\$F3D1	MltNam	A	NamenTabel (multicolor) (default \$0800)
\$F3D3	MltCol	A	KleurenTabel (multicolor) (default \$0000)
\$F3D5	MltCgp	A	Karakterset (multicolor) (default \$0000)
\$F3D7	MltAtr	A	Sprite-attributen (multicolor) (default \$1B00)

\$F3D9	MltPat	A	Sprite-patronen (multicolor) (default \$3800)
\$F3DB	ClikSw	V	'Klik' bij druk op toets
\$F3DC	CsrY	V	Verticaalcoördinaat van de cursor
\$F3DD	CsrX	V	Horizontaalcoördinaat van de cursor
\$F3DE	CnsDfg	V	Functietoetsen op het scherm? (0=niet)
\$F3DF	Rg0Sav	V	Kopie VDP-register 0 (default \$00)
\$F3E0	Rg1Sav	V	Kopie VDP-register 1 (default \$F0)
\$F3E1	Rg2Sav	V	Kopie VDP-register 2 (default \$00)
\$F3E2	Rg3Sav	V	Kopie VDP-register 3 (default \$00)
\$F3E3	Rg4Sav	V	Kopie VDP-register 4 (default \$01)
\$F3E4	Rg5Sav	V	Kopie VDP-register 5 (default \$00)
\$F3E5	Rg6Sav	V	Kopie VDP-register 6 (default \$00)
\$F3E6	Rg7Sav	V	Kopie VDP-register 7 (default \$F4)
\$F3E7	StatFl	V	VDP-statusregister (default \$CA)
\$F3E8	TrgFlg	V	Status van alle vuurknoppen
\$F3E9	ForClr	V	Huidige voorgrondkleur (default 15)
\$F3EA	BakClr	V	Huidige achtergrondkleur (default 4)
\$F3EB	BorClr	V	Huidige borderkleur (default 4)
\$F3EC	MaxUpd	S	Sprong voor LINE-routine
\$F3EF	MinUpd	S	Sprong voor LINE-routine
\$F3F2	AtrByt	V	Penkleur voor grafische routines
\$F3F3	Queues	A	Controleblokken voor de drie muziek-arrays
\$F3F5	FrcNew	V	Flag voor CLOAD (onderscheid tussen CLOAD en CLOAD?, 0=CLOAD)
\$F3F6	ScnCnt	V	Frequentie keyboard-scanning
\$F3F7	RepCnt	V	Snelheid keyboard-repeating
\$F3F8	PutPnt	A	Adres van de volgende 'plaats'-positie in de keyboard-buffer
\$F3FA	GetPnt	A	Adres van de volgende 'haal'-positie in de keyboard-buffer
\$F3FC	Csi1200	P	Parameters voor 1200 baud-cassette
\$F401	Cs2400	P	Parameters voor 2400 baud-cassette
\$F406	Low	V	LAAG-cyclus
\$F408	High	V	HOOG-cyclus
\$F40A	Header	V	Cycli in aanloopblok
\$F40B	AspCt1	V	Verhouding stralen van CIRCLE
\$F40D	AspCt2	V	Verhouding stralen van CIRCLE (default)
\$F40F	EndPrg	R	Lege BASIC-regel (voor 'ON ERROR...')
\$F414	ErrFlg	V	Nummer van de opgetreden fout
\$F415	LptPos	V	Positie van de printer-kop
\$F416	PrtFlg	V	Uitvoer naar scherm of printer? (0=scherm / 1=printer)
\$F417	NtMSXP	V	Hoe wordt de printer grafisch aangestuurd?
\$F418	RawPrt	V	Hoe worden de grafische karakters naar de printer gestuurd?
\$F419	VlzAdr	A	Adres van karakter voor VAL-functie
\$F41B	VlzDat	V	Waarde van karakter voor VAL-functie
\$F41C	CurLin	V	Het regelnummer dat nu wordt verwerkt door de interpreter
\$F41F	KBuf	B	Gecodeerde vorm van een ingetypte regel
\$F55E	Buf	B	Tekst die INLIN heeft ingelezen
\$F661	TtyPos	V	Positie op het scherm voor PRINT
\$F662	DimFlg	V	Zoekroutine voor array-variabelen

\$F663	ValTyp	V	Type van de variabele in DAC
\$F664	DoRes	V	Flag voor het scannen van een DATA-regel
\$F665	DoNum	V	Flag voor omzetten van regelinummer
\$F666	ConTxt	A	Adres voor eerstvolgende karakter NA een numerieke constante
\$F668	ConSav	V	Token-opslag van een numerieke constante
\$F669	ConTyp	V	Type-opslag van een numerieke constante
\$F66A	ConLo	B	Opslagruimte voor numerieke constanten
\$F672	MemSiz	A	Top van het stringgeheugen (wijzigen met CLEAR of MAXFILES)
\$F674	StkTop	A	Top van de stapel (wijzigen met CLEAR of MAXFILES)
\$F676	TxtTab	A	Bodem van het programma-geheugen (default \$8001)
\$F678	TempPt	A	Eerstvolgende vrije ruimte in TEMPST
\$F67A	TempSt	B	Stringstapel
\$F698	DscTmp	B	String-signalementbuffer
\$F69B	FreTop	A	Eerstvolgende vrije ruimte in het stringgeheugen
\$F69D	Temp3	V	Interpreter-ruimte (algemeen gebruik)
\$F69F	Temp8	V	Interpreter-ruimte (algemeen gebruik)
\$F6A1	EndFor	A	FOR...NEXT-werkpointer
\$F6A3	DatLin	A	Regelnummer van huidig DATA-statement
\$F6A5	SubFlg	V	Variabelenindex
\$F6A6	FigInp	V	Flag voor verschil tussen INPUT en READ (0=INPUT)
\$F6A7	Temp	V	Interpreter-ruimte (algemeen gebruik)
\$F6A9	PtrFlg	V	Flag regelnummeradres
\$F6AA	AutFlg	V	Flag voor AUTO (0=geen AUTO)
\$F6AB	AutLin	V	Huidige regelnummer tijdens uitvoering van AUTO
\$F6AD	AutInc	V	Regelverhoging tijdens AUTO
\$F6AF	SavTxt	A	Adreswijzer van interpreter
\$F6B1	SavStk	A	Adreswijzer van de stapel
\$F6B3	ErrLin	V	Regelnummer indien er een fout optreedt
\$F6B5	Dot	V	Nummer van de laatst behandelde regel
\$F3B7	ErrTxt	B	Buffer voor SAVTXT
\$F6B9	OnElIn	A	Adres voor ON...ERROR
\$F6BB	OnEFlg	V	Flag voor ON...ERROR (1=error opgetreden)
\$F6BC	Temp2	V	Interpreter-ruimte (algemeen gebruik)
\$F6BE	OldLin	V	Regelbuffer voor programma-onderbreking
\$F6C0	OldTxt	A	Statement-buffer voor programma-onderbreking
\$F6C2	VarTab	A	Eerste byte in het variabelengeheugen
\$F6C4	AryTab	A	Eerste byte in het array-geheugen
\$F6C6	StrEnd	A	Einde array-geheugen
\$F6C8	DatPtr	A	Adres van huidige DATA-statement
\$F6CA	DefTbl	V	Type van elke groep BASIC-variabelen (A...Z)
\$F6E4	PrmStk	A	Startadres van FN-parameters
\$F6E5	PrmLen	V	Lengte van het FN-blok dat in PARM1 staat
\$F6E8	Paral	B	Lokale variabelen van een gevalueerde FN-functie
\$F74C	PrmPrv	A	Adres van FN-blok
\$F74E	PrmLn2	V	Lengte van het FN-blok dat in PARM2 staat
\$F750	Param2	B	Variabelenconstructie voor FN-functie
\$F7B4	PrmFlg	V	Flag voor onderscheid tussen globale en lokale variabelen

\$F785	AryTa2	A	Eindadres van het thans doorzochte variabelengeheugen
\$F787	NoFuns	V	Signalering van lokale variabelen
\$F788	Temp9	V	Interpreter-ruimte (algemeen gebruik)
\$F78A	FunAct	V	Het aantal FN-functies thans in gebruik
\$F78C	SwpTmp	B	Buffer voor SWAP-statement
\$F7C4	TrcFlg	V	Flag voor de TRACE-functie (0=geen TRACE)
\$F7C5	FBuffr	B	Buffer voor numerieke output
\$F7F0	DecTmp	V	Tijdelijke opslagruimte voor deelroutine
\$F7F2	DecTm2	V	Tijdelijke opslagruimte voor deelroutine
\$F7F4	DecCnt	V	Teller voor deelroutine
\$F7F6	Dac	B	Primaire buffer voor expressie-evaluatie in de interpreter
\$F808	HoldB	B	Buffer voor verenigvuldigingsroutine
\$F847	Arg	B	Secundaire buffer voor expressie-evaluatie in de interpreter
\$F857	RndX	B	Buffer voor huidige random-waarde
\$F85F	MaxFil	V	Aantal mogelijke IO-buffers
\$F860	FilTab	A	Adres van pointer-tabel voor file-controleblokken
\$F862	NulBuf	A	Adres van het eerste byte van de databuffer van IO-buffer 0
\$F864	PrtFil	A	Adres van het file-controleblok van de huidige IO-buffer
\$F866	FilNam	B	Buffer voor bestandsnaam
\$F871	FilNm2	B	Buffer voor ingelezen bestandsnaam
\$F87C	MicNiy	V	Status voor IO-buffers tijdens LOAD
\$F87D	SavEnd	A	Adres voor het laatste byte tijdens BSAVE
\$F87F	FnkStr	B	Buffer voor functietoetsen-strings
\$F91F	CgPnt	P	Slot en adres voor de te kopiëren karakterset
\$F922	NamBas	A	Startadres van de huidige namentabel voor de VDP
\$F924	CgpBas	A	Startadres van de huidige karaktertabel voor de VDP
\$F926	PatBas	A	Startadres van de huidige sprite-patronentabel voor de VDP
\$F928	AtrBas	A	Startadres van de huidige sprite-attributentabel voor de VDP
\$F92A	Cloc	V	Positie van de laatst verwerkte pixel
\$F92C	CMAsk	V	Masker gebruikt tijdens de laatste pixel-operatie
\$F92D	MinDel	V	Minimaal verschil voor LINE-routine
\$F92F	MaxDel	V	Maximaal verschil voor LINE-routine
\$F931	Aspect	V	Stralenverhouding tijdens CIRCLE
\$F933	CenCnt	V	Teller voor plot-routine tijdens CIRCLE
\$F935	CLineF	V	Flag voor CIRCLE-routine
\$F936	CnPnts	V	Aantal segment-punten voor CIRCLE
\$F938	CplotF	V	Flag voor CIRCLE-routine
\$F939	CpCnt	V	Y-coördinaat tijdens CIRCLE
\$F93B	CpCnt8	V	Aantal punten thans geplot tijdens CIRCLE
\$F93D	CrcSum	V	Teller voor CIRCLE
\$F93F	CstCnt	V	Aantal punten binnen starthoek tijdens CIRCLE
\$F941	CsClXY	V	Afplattings-flag voor CIRCLE
\$F942	CSaveA	V	Tijdelijke ruimte voor SCANR
\$F944	CSaveM	V	Tijdelijke ruimte voor SCANR
\$F945	CXOff	V	Tijdelijke opslagruimte voor CIRCLE
\$F947	CYOff	V	Tijdelijke opslagruimte voor CIRCLE
\$F949	LohMsk	V	Tijdelijke opslagruimte voor PAINT

\$F94A	LohDir	V	Nieuwe scan-richting tijdens PAINT
\$F94B	LohAdr	A	Adres van meest linkse uitstulping tijdens PAINT
\$F94D	LohCnt	V	Grootte van linker uitstulping
\$F94F	SkpCnt	V	Aantal pixels over te slaan tijdens PAINT
\$F951	MovCnt	V	Aantal bewegingen tijdens PAINT
\$F953	PdiRec	V	Huidige richting tijdens PAINT
\$F954	LfProg	V	Flag voor beweging naar links tijdens PAINT
\$F955			
\$F956	MclTab	A	Adres voor macro-ontledertabel (DRAW of PLAY)
\$F958	MclFlg	V	Flag voor macro-ontleidergebruik (DRAW of PLAY)
\$F959	QueTab	P	Parameters voor afspelen van muziek (4*6)
\$F971	QueBak	V	Grootte van de muziek-arrays (4*1)
\$F975	VoicAQ	B	Muziek-array kanaal A
\$F976	VoicBQ	B	Muziek-array kanaal B
\$F977	VoicCQ	B	Muziek-array kanaal C
\$FAF5	Rz2IQ	B	Rij voor het RS232-kanaal
\$FB35	PrsCnt	V	Teller voor de PLAY-routine
\$FB36	SavSp	A	Opslag van de Z80-stapelwijzer tijdens PLAY
\$FB38	VoiceN	V	Nummer van het thans gebruikte kanaal tijdens PLAY
\$FB39	SavVol	V	Opslag van het volume tijdens een R-verwerking
\$FB3B	MclLen	V	Lengte van de huidige macro-ontlede string
\$FB3C	MclPtr	A	Adres van de huidige macro-ontlede string
\$FB3E	QueueN	V	Het nummer van de thans verwerkte muziek-array
\$FB3F	MusicF	V	Flags voor te verwerken muziek-array(s)
\$FB40	PlyCnt	V	Aantal opeenvolgende PLAY-statements
\$FB41	VcbA	P	Parameters voor PLAY-routine, kanaal A
\$FB66	VcbB	P	Parameters voor PLAY-routine, kanaal B
\$FB8B	VcbC	P	Parameters voor PLAY-routine, kanaal C
\$FB80	EnStop	V	Flag voor warme start interrupt (0=geen warme start)
\$FB81	BasRom	V	Flag voor CTRL/STOP-reactie (0=geen 'reactie')
\$FB82	LinTtb	V	Flags voor het 'overlopen' van een schermregel (24*)
\$FB8A	FstPos	V	De coördinaten van de cursors i.v.m. terugrekening bij 'overloop'
\$FBCC	CurSav	V	Opslag van het karakter dat werd vervangen door de cursor
\$FBCC	FnkSwi	V	Bijhouden welke groep functietoetsen wordt afgabbeeld
\$FBCE	FnkFig	V	Flags of functietoetsen in werking (10*)
\$FBDB	Onsbsf	V	Aantal interrupts bijhouden (bijv. intussen opgetreden ID e.d.)
\$FBDB	ClikFl	V	Flag om te zorgen dat er maar één 'klik' tegelijk wordt gegeven
\$FBDA	OldKey	B	Buffer voor het opslaan van de 'vorige' keyboard-matrix
\$F3E5	NewKey	B	Buffer voor het opslaan van de 'huidige' keyboard-matrix
\$FBF0	KeyBuf	B	Keyboard-buffer
\$FC18	LinWrk	B	BIOS-buffer om een hele schermregel op te kunnen slaan
\$FC40	PatWrk	B	BIOS-buffer voor het opslaan van een 8[+]8 pixel-patroon
\$FD48	Bottom	A	Laagste RAM-adres (default \$8000)
\$FC4A	HiMem	A	Hoogste RAM-adres (default \$F380)
\$FC4C	TrpTbl		Interrupt-veroorzakers
\$FC4C		V/A	Key 0
\$FC4F		V/A	Key 1

\$FC52	V/A	Key 2	
\$FC55	V/A	Key 3	
\$FC58	V/A	Key 4	
\$FC5B	V/A	Key 5	
\$FC5E	V/A	Key 6	
\$FC61	V/A	Key 7	
\$FC64	V/A	Key 8	
\$FC67	V/A	Key 9	
\$FC6A	V/A	STOP	
\$FC6D	V/A	Sprite	
\$FC70	V/A	Strig 0	
\$FC73	V/A	Strig 1	
\$FC76	V/A	Strig 2	
\$FC79	V/A	Strig 3	
\$FC7C	V/A	Strig 4	
\$FC7F	V/A	Interval	
\$FC82		n.g.	
\$FC85		n.g.	
\$FC88		n.g.	
\$FC8B		n.g.	
\$FC8E		n.g.	
\$FC91		n.g.	
\$FC94		n.g.	
\$FC97		n.g.	
\$FC9B	IntFlg	V	(CTRL/)STOP-detectie
\$FC9C	PadY	V	Y-positie touch tablet
\$FC9D	PadX	V	X-positie touch tablet
\$FC9E	Jiffy	V	Timer
\$FCA0	IntVal	V	Tijdsduur voor ON...INTERVAL
\$FCA2	IntCnt	V	Aftelling i.s.m. ON...INTERVAL
\$FCA4	LowLim	V	Duur van startbit bij lezen van cassette
\$FCA5	WinWid	V	Tijdsduur hoog/laag-cyclus bij lezen van cassette
\$FCA6	GrpHed	V	Flag voor grafische header (0=geen grafische header)
\$FCA7	EscCnt	V	Flag voor ESC-verwerking (0=geen escape-code)
\$FCA8	InsFlg	V	Flag voor INSERT-mode (0=geen INSERT)
\$FCA9	CsRSr	V	Flag voor 'cursor in beeld' (0=geen cursor)
\$FCAA	CStyle	V	Flag voor cursor-type (0=blok / anders=8[*]4)
\$FCAB	CapSt	V	Flag voor 'Caps Lock' (0=geen 'Caps Lock')
\$FCAC	KanaSt	V	Flag voor 'Kana Lock' (Op Japanse machines)
\$FCAD	KanaMd	V	Keyboard-mode op Japanse machines
\$FCAE	FlbMem	V	Flag voor onderdrukking IO-foutmeldingen
\$FCAF	ScrMod	V	Huidige scherm-mode
\$FCB0	OldScr	V	Vorige scherm-mode
\$FCB1	CasPrv	V	Vorig opgeslagen karakter voor cassette
\$FCB2	BrdAtr	V	Randkleur voor PAINT-routine
\$FCB3	GXPos	A	Opslag High-Res X-positie
\$FCB5	GYPos	A	Opslag High-Res Y-positie

\$FCB7	GrpAcX	A	Huidige X-positie voor GRPPRT
\$FCB9	GrpAcY	A	Huidige Y-positie voor GRPPRT
\$FCBB	DrwFlg	V	Flag voor DRAW-routine
\$FCBC	DrwScl	V	'Scale factor' voor DRAW
\$FCBD	DrwAng	V	Gebruikte hoek bij DRAW
\$FCBE	RunBnf	V	Flag voor BLOAD met R-parameter (0=geen auto RUN)
\$FCBF	SavEnt	A	Adres voor BSAVE (en BLOAD)
\$FCC1	ExpTbl	V	Flags voor expanded slots
\$FCC5	SltTbl	V	Huidige toestanden van de expanded slot-registers
\$FCC9	SltAtr	P	Parameters voor uitbreidings-ROM's
\$FD09	SltWrk	B	Mogelijk werkgebied voor uitbreidings-ROM's
\$FD89	ProcNm	B	Naam van een statement of apparaat i.v.m. uitbreidings-ROM's
\$FD99	Device	V	Device-ID voor een uitbreidings-ROM-adres

B: buffer

P: parameters

R: routine

S: sprong

V: variabele

# Appendix K

## HOOKS

Adres	Naae	Functie
\$FD9A		
\$FDF9	H.TimI	Interrupt, voornamelijk timer-gestuurd
\$FDA4	H.ChPu	Standaard CHPUT
\$FDA9	H.DspC	Plaats cursor
\$FDAE	H.EraC	Verwijder cursor
\$FDDB	H.DspF	Standaard DSPFNK
\$FD88	H.EraF	Standaard ERAFNK
\$FDDB	H.ToTe	Standaard TOTEXT
\$FDC2	H.ChGe	Standaard CHGET
\$FDC7	H.IniP	Kopieer karakterset naar VDP
\$FDCC	H.KeyC	Keyboard decoder
\$FDD1	H.KeyA	Keyboard decoder (snel)
\$FDD6	H.NMI	Standaard NMI
\$FDDB	H.PinL	Standaard PINLIN
\$FDE0	H.QinL	Standaard QINLIN
\$FDES	H.InLi	Standaard INLIN
\$FDEA	H.OnGo	ON device GOTO procedure
\$FDEF	H.DskO	Disk Output (DSKO\$)
\$FDF4	H.Sets	SET
\$FDF9	H.Name	RENAME
\$FDFE	H.Kill	KILL
\$FE03	H.Ipl	Initial Program Load (IPL)
\$FE08	H.Copy	COPY
\$FE0D	H.Cmd	CMD
\$FE12	H.DskF	Disk Free (DSKF)
\$FE17	H.DskI	Disk Input (DSKI\$)
\$FE1C	H.Attr	ATTR\$
\$FE21	H.Lset	Left Set (LSET)
\$FE26	H.Rset	Right Set (RSET)
\$FE2B	H.Fiel	FIELD
\$FE30	H.Mki\$	Make Int (MKI\$)
\$FE35	H.Mki\$	Make Single (MKI\$)
\$FE3A	H.Mkd\$	Make Double (MKD\$)
\$FE3F	H.Cvi	Convert Int (CVI)
\$FE44	H.Cvs	Convert Single (CVS)
\$FE49	H.Cvd	Convert Double (CVD)
\$FE4E	H.BetP	Standaard GETPTR (get file pointer)

\$FE53	H.SetF	Standaard SETF1L (set file pointer)
\$FE58	H.NoFo	Standaard NOFOR (no for clause) t.b.v. OPEN
\$FE5D	H.NuLo	Standaard NULOPN (null open file) t.b.v. OPEN
\$FE62	H.NtFl	Standaard NTFLD (not file number 0) t.b.v. CLOSE IO-buffer 0
\$FE67	H.Merg	MERGE
\$FE6C	H.Save	SAVE
\$FE71	H.BinS	SAVE (binair)
\$FE76	H.BinL	LOAD (binair)
\$FE7B	H.File	FILES
\$FE80	H.DGet	DGET
\$FE85	H.FilD	Standaard FILOUI (seq. uitvoer)
\$FE8A	H.InDs	Standaard INDSKC (seq. invoer)
\$FE8F	H.RsLF	Herselectie oude drive (INPUT\$)
\$FE94	H.SavD	Save huidige drive
\$FE99	H.Loc	LOC (lokatie-functie)
\$FE9E	H.Lof	LOF (lengte van file-functie)
\$FEA3	H.Eof	EOF (einde file-functie)
\$FEA8	H.FPos	FPOS (file positie-functie)
\$FEAD	H.BakU	Standaard BAKUPT (back up-routine)
\$FEB2	H.ParD	Standaard PARDEV (pass device name)
\$FEB7	H.NoDe	Standaard NODEVN (no device name)
\$FEBD	H.PosD	Standaard POSDSK (possible disk)
\$FEC1	H.DevN	Standaard DEVNAM (device name)
\$FED6	H.GenD	Standaard GENDSP (general dispatcher)
\$FECB	H.RunC	Standaard RUNC (run-clear)
\$FEDD	H.Clea	CLEAR
\$FED5	H.LopD	Standaard LOPDFT (loop and set default)
\$FEDA	H.StkE	Standaard STKERR (Z80-stack error )
\$FEDF	H.IsFl	Standaard ISFLIO (file ID)
\$FFE4	H.OutD	Standaard OUTDO
\$FFE9	H.CrDo	Standaard CRLF (CR+linefeed naar OUTDO)
\$FEEE	H.DskC	Standaard DSKCHI (disk character input)
\$FEF3	H.DoGr	Standaard DOGRPH
\$FEF8	H.PrgE	Standaard PRGEND (einde van programma)
\$FEFD	H.ErrP	Standaard ERRPRT (error uitvoer)
\$FF02	H.ErrF	Error verwerking
\$FF07	H.Read	'Alles OK'-gedeelte van de 'MAIN loop'
\$FF0C	H.Main	Standaard MAIN (hoofdlus)
\$FF11	H.DirD	Standaard DIRDO (direct statement)
\$FF16	H.FinI	Einde hoofdlus-invoer
\$FF1B	H.FinE	Einde hoofdlus-error (onjuist regelnummer)
\$FF20	H.Crun	(De)coderen BASIC-regels
\$FF25	H.Crus	(De)coderen BASIC-regels
\$FF2A	H.IsRe	(De)coderen BASIC-regels
\$FF2F	H.NtFn	(De)coderen BASIC-regels
\$FF34	H.SnGf	FOR
\$FF39	H.NewS	Nieuw statement verwerken (zelfde regel)

\$FF3E	H.Sone	Signalering uitvoerpunt in statement-verwerking
\$FF43	H.ChrG	Standaard CHGET
\$FF48	H.Retu	RETURN
\$FF4D	H.PrtF	PRINT
\$FF52	H.Comp	PRINT (zoeken naar exclusieve tekens)
\$FF57	H.FinP	PRINT (einde)
\$FF5C	H.TrMn	Invoer-error
\$FF61	H.FrMe	Expressie-evaluatie
\$FF66	H.NtpL	Expressie-evaluatie
\$FF6B	H.Eval	Factor-evaluatie
\$FF70	H.OkNo	Factor-evaluatie
\$FF75	H.FinG	Factor-evaluatie
\$FF7A	H.Ismi	Signalering uitvoerpunt in statement-verwerking
\$FF7F	H.Widt	WIDTH
\$FF84	H.List	LIST
\$FF8E	H.BufL	Standaard BUFLIN (buffer line) voor (de)coderen BASIC-regels
\$FF93	H.FrqI	Standaard FRQINT (integer-omrekening)
\$FF98	H.ScnE	Omzetting van regelnummer naar adres
\$FF9D	H.Fret	Standaard FRETM (free up to temporaries)
\$FFA2	H.Ptrg	Standaard PTRGET (pointer get voor variabelen)
\$FFA7	H.PhxD	Standaard PHYDIO (physical disk IO)
\$FFAC	H.Form	Standaard FORMAT
\$FFB1	H.Erra	ERROR-behandeling...
\$FFB6	H.Lpto	Standaard LPTOUT
\$FFBB	H.Lpts	Standaard LPTSTT
\$FFC0	H.Scre	SCREEN
\$FFC5	H.Play	PLAY

# Index

A-register 20  
absolute adressering 24  
accumulator 20  
adressering, geïndexeerde 25  
adressering, impliciete 24  
adressering, onmiddellijke 24  
adressering, zero page 25  
adressering, indirecte 26  
adressering, relatieve 25  
adressering, absolute 24  
adressering, bit- 27  
algemene registers 20  
assembler 15  
BASIC 14  
binaire getallen 56  
BIOS 61  
bit 56  
bitadressering 27  
byte 56  
carry 21  
EPROM 13  
flag-register 21  
geïndexeerde adressering 25  
half carry flag 21  
hexadecimale getallen 56  
impliciete adressering 24  
indexregisters 20

indirecte adressering 26  
interrupt, maskeerbare 48  
interrupt 35, 37  
interrupt-register 22  
maskeerbare interrupt 48  
monitor 15  
nybble 59  
onmiddellijke adressering 24  
overflow flag 21  
parity flag 21  
programmateller 18  
RAM 13  
refresh-register 22  
relatieve adressering 25  
ROM 13  
scherm-modes 115  
sign flag 22  
sprite 115, 159  
stack pointer 19  
stapelwijzer 18  
subtract flag 21  
VDP-processor 116  
verborgen registerset 22  
word 46  
Z80, opbouw van de 17  
zero page-adressering 25  
zero flag 21





# **Bestelkaart diskette/cassette**

Ondergetekende:

Naam: \_\_\_\_\_

Adres: \_\_\_\_\_

Postcode en woonplaats: \_\_\_\_\_

bestelt hierbij\*) de:

- |                                   |             |
|-----------------------------------|-------------|
| <input type="checkbox"/> diskette | ad. f 24,50 |
| <input type="checkbox"/> cassette | ad. f 19,50 |

behorende bij: **Machinetaal voor MSX-computers**

U kunt deze diskette of cassette in uw bezit krijgen door een ingevulde en ondertekende girobetaalkaart of bankcheque te zamen met deze bestelkaart te zenden aan:

**Kluwer Technische Boeken B.V.**  
t.a.v. afd. administratie  
**Brink 25**  
**7411 BS Deventer**

Desgewenst kunt u het verschuldigde bedrag ook overmaken op gironummer 3243819 t.n.v. Kluwer Technische boeken in Deventer onder vermelding van bestelnummer 2035 CAS voor de cassette, of:  
bestelnummer 2035 DIS voor de diskette

\* aankruisen wat u wenst.

Bezitters van MSX-computers weten dat hun machine over grandioze mogelijkheden beschikt. Helaas is BASIC, de ingebouwde programmeertaal van deze computers, vaak te langzaam om alles uit de computer te kunnen halen.

Enthousiaste MSX-bezitters zijn dan ook geneigd om te experimenteren met machinetaal, een taal die veel dichter bij de computer staat dan BASIC. Het is echter zeer moeilijk om zonder deskundige begeleiding in machinetaal thuis te raken.

Dit boek zorgt voor die deskundige begeleiding. Het is geschreven voor de MSX-bezitter die zich wil verdiepen in de geheimen van machinetaal. Enige BASIC-voorkennis is vereist.

Met dit boek komt u al zeer snel tot spectaculaire resultaten. Dankzij de gratis machinetaalmonitor, als programma-listing opgenomen, kunt u meteen met de vele voorbeelden experimenteren. Tegelijkertijd leert u spelenderwijs omgaan met de vele machinetaalinstructies die de MSX-computer kent. De vele overzichtelijke tabellen en tekeningen zult u steeds weer willen raadplegen.

John Vanderaart geniet grote bekendheid als programmeur en als auteur van vele artikelen. In dit boek bewijst hij op een prettig leesbare manier dat iedereen in machinetaal kan leren programmeren.