Liesert/Schieb

Peeks en pokes voor de CPC computers

CPC Bibliotheek 10

DATA BECKER
NEDERLANDS*

Peeks en pokes voor de CPC computers

Liesert/Schieb

Peeks en pokes voor de CPC computers

CPC Bibliotheek 10

DATA BECKER
NEDERLANDS*

Oorspronkelijke titel
Peeks & Pokes zum CPC

Vertaling
W. B. de Jong, Utrecht

Omslag
Martin van Keulen

Druk
Vonk, Zeist
ISBN 90 229 3363 6
D/1984/0939/283

INHOUD

5. GRAFIEK

6. TOEPASSING VAN GRAFIEKEN

7. PROGRAMMEREN MET INTERRUPTS

8. SOUND

9. BASIC EN HET BEDRIJFSSTEEM

10. HULPAPPARATEN EN HUN FUNCTIONEREN

VOORWOORD

Hebt u zich ooit wel eens de vraag gesteld hoe uw CPC eigenlijk werkt? Hebt u altijd al willen weten wat een bedrijfssysteem is en wat men ermee kan doen? Op zulke vragen kan een handboek, al is het nog zo uitvoerig geschreven, geen antwoord geven. Vele kneepjes en trucs die het leven van een programmeur aanmerkelijk gemakkelijker maken, worden echter pas mogelijk na een blik achter de schermen. Daarom willen we in dit boek uiteenzetten hoe uw computer functioneert, wat er gebeurt als u b.v. de RETURN-toets indrukt, enz.

De titel PEEKS & POKES laat gemakkelijk de indruk ontstaan dat er hier een boek voor u ligt dat slechts over deze twee commando's gaat. Gelukkig is dat niet zo (dat zou ook voor mij een vervelende zaak worden).
De titel is gekozen omdat de voorloper van dit boek 'PEEKS & POKES voor de Commodore-64' heette en we hetzelfde concept en de manier van laten verschijnen niet wilden veranderen. Bovendien wordt ook een Commodore-64 niet slechts met PEEKS en POKES geprogrammeerd. Ten slotte nog dit: deze twee commando's hebben in het algemeen de reputatie dat ze de toegang tot de begrippen van het bedrijfssysteem en de machinetaal vereenvoudigen. En precies dàt willen we bereiken.

Natuurlijk worden ook de bijbehorende trucs erbij geleverd, en wel op twee verschillende manieren. Nadat een tip uitvoerig aan u is voorgesteld, wordt hij aan het einde van het hoofdstuk in een samenvatting nog eens meegedeeld, waardoor men bij het terugzoeken van deze truc niet alles uit dat hoofdstuk hoeft over te lezen.

Ook de Spartaanse grafiek-commando's van de CPC worden uitgebreid, en wel zodanig dat ze gemakkelijk te begrijpen zijn. U hoeft de informatie niet geheel te doorgronden om alles te kunnen begrijpen. En als de machinetaal voor u nog steeds een ondoorzichtige materie is, vindt u in dit boek nog wel zo iets als een 'snuffelcursus' waarmee u een begin kunt maken met het programmeren. Later kunt u altijd nog de beslissing nemen of u met assembler of liever met Pascal of iets dergelijks verder wilt gaan.

Mij blijft nu nog slechts over om u veel plezier toe te wensen met deze lectuur en bij het toetsen ervan in de praktijk.

Hans Joachim Liesert
Munster, november 1984

1. HOE WERKT EEN COMPUTER?

In het hierna volgende leert u de CPC en zijn manier van functioneren kennen. Degenen onder u die al een beetje bedreven zijn in het programmeren, kunnen rustig verder bladeren. De 'cracks' onder u mogen mij de eventuele vereenvoudigingen niet kwalijk nemen, omdat ik die voor een beter begrip nodig vond.

1.1. OOK EEN MICROCOMPUTER HEEFT INGANGEN

Allereerst een beginsel. Elke microprocessor kan naar een bepaalde geheugenplaats adresseren, dat wil zeggen, hij kan een zeker aantal geheugenplaatsen of bytes oproepen. Dit is afhankelijk van het aantal adresleidingen dat de processor bezit. Elke adresleiding presenteert een bit (wat dat is, zult u zich uit het handboek nog wel herinneren) en deze kan slechts in twee vormen voorkomen: in 0 en 1.

De Z-80-microprocessor, waarin zich de 'hersenen' van uw CPC bevinden, heeft 16 van deze adresleidingen, die samen ook wel eens adresingang genoemd worden. Daarmee kan de CPC 216 (dat zijn 65.535, of ook wel 64K genoemd) geheugenplaatsen oproepen.

Misschien is het u al opgevallen dat de CPC over 64 RAM en 32 ROM, dat is 96K, geheugen beschikt, dus eigenlijk meer geheugenruimte heeft dan hij kan gebruiken. Hoe dat ondanks alles toch nog kan werken, zal ik later uiteenzetten.

Behalve over de adresingang beschikt de Z-80 over nog twee ingangen. Ten eerste is er de besturingsingang. Dat is eigenlijk alleen maar een naam voor het totaal aantal besturingen waarmee bijvoorbeeld het geheugen wordt omgeschakeld van invoer naar uitvoer e.d.

Bovendien is er nog een ingang die veel belangrijker is, en wel de data-ingang. Deze beschikt over 8 leidingen en heeft de opdracht om data (of, nauwkeuriger gezegd, 8 bits = 1 byte) in het inwendige van de computer te transporteren.

Zowel de data - als de adresingang is verbonden met alle delen van de computer die door de microprocessor kunnen worden opgeroepen - dus met RAM, ROM en andere chips.

Steeds als de microprocessor een opdracht uitvoert die gegevens betreft anders dan via de Z-80, geeft hij allereerst het geheugenadres (zogezegd het telefoonnummer) op de adresingang. De geheugenbouwsteen herkent welke byte er bedoeld wordt. Dan geeft de microprocessor de gegevens hetzij aan de adresingang, waar ze dan het geheugen ingaan, of het geheugen geeft de inhoud van de geheugenplaats via de ingang verder aan de Z-80. Zo eenvoudig is dat!

Dit geldt voor alle 8-bit-processors (8 is afkomstig van de ingangsbreedte van de data-ingang). Maar vanaf nu zullen we ons ook met de speciale eigenschappen van uw CPC-464 bezighouden.

1.2. DE HARDWARE-OPBOUW VAN DE CPC

Geen angst, want ook hierbij wordt het niet al te technisch. Het is voor het begrijpen van het volgende hoofdstuk echter wel belangrijk als men het een en ander van het inwendig functioneren van de CPC 464 weet.

Aan het einde van dit hoofdstuk vindt u een zeer vereenvoudigd blokschema van uw computer (fig. 1). Zoals u uit deze afbeelding kunt zien, liggen ROM en RAM gedeeltelijk naast elkaar, dat wil zeggen, ze hebben dezelfde adressen. Nu kan een byte geen twee verschillende waarden leveren (er bestaat immers slechts één data-ingang). Ergens moet men dus bepalen wat er bedoeld wordt: RAM of ROM. Afhankelijk van de wens van de programmeur wordt dan het ene of het andere deel van de data-ingang ingeschakeld. Dit gebeurt omdat de processor de verschillende delen ook voor verschillende doelen gebruikt. Altijd als er bepaalde data nodig zijn, wordt de Z-80 tijdig ingeschakeld.

Vanuit BASIC kunt u overigens alleen maar het RAM-gedeelte aanroepen - per PEEK of per POKE. Met een kleine truc kan men echter toch het ROM bereiken - hoe, dat verklaar ik later.

Een deel van het RAM bepaalt het beeldschermgeheugen. Hiervandaan haalt de zogenoemde 6845-chip zijn informatie. Deze IC heeft de opdracht om de gegevens uit het video-RAM om te zetten in een videosignaal voor de monitor.

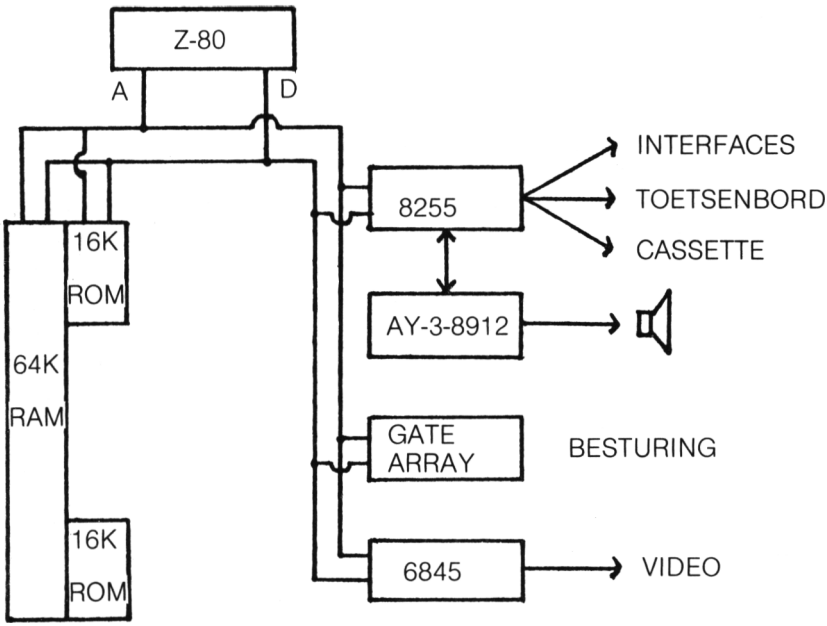
Behalve de 6845 zijn er nog meer chips. De AY-3-8812 (geen angst, u hoeft deze combinatie niet uit het hoofd te leren) wordt voor de SOUND gebruikt, dat wil zeggen, hij zet SOUND-commando's om in hoorbare tonen.

De 8812 is echter om technische redenen niet direct met de Z-80 verbonden, maar ontvangt zijn gegevens van de interface 8255 (zie afb. 1).

Deze geïntegreerde schakelkring geeft de gegevens van de processor door aan de diverse hulpapparaten, zoals de cassetterecorder, de interface en ook aan het toetsenbord (dat voor de Z-80 eveneens een afzonderlijk apparaat is dat zich toevallig ook in dezelfde installatie bevindt).

Als bijvoorbeeld het indrukken van een toets wordt onderzocht, dan wordt dat meegedeeld aan de interface. Deze 'ziet' dan of en welke toets er is ingedrukt, hetgeen dan via de data-ingang aan de Z-80 wordt meegedeeld. Als er een byte via een interface moet worden verwerkt, stuurt de processor die naar de 8255. Daar wordt hij dan bewaard tot het bijbehorende hulpapparaat gereed is, waarna hij wordt doorgestuurd.

Een andere chip regelt speciale, interne gebeurtenissen, die voor ons verder niet van belang zijn.



afb. 1 De opbouw van de CPC-464

1.3. INDELING VAN HET GEHEUGEN

Zoals u uit het laatste hoofdstuk te weten bent gekomen, bezit de CPC 64 64K RAM. Voor het programmeren in BASIC beschikken we echter slechts over 42,5K. Met recht kunt u zich afvragen waar de resterende 21,5K zijn gebleven.

Bij de eerste oogopslag schijnt het mogelijk te zijn om een veel groter geheugen voor BASIC te reserveren. Maar helaas heeft de computer zelf ook nogal wat geheutenruimte voor interne functies nodig. Allereerst kunnen we de video-RAM noemen. Deze heeft tot taak het beeld van het beeldscherm op te slaan. Elke keer als er een punt op het beeldscherm gezet of uitgewist moet worden, verandert de precessor de overeenkomstige waarde in de video-RAM. De chip die voor deze videosignalen zorgt, kijkt dan met regelmatige afstanden na welke punten er moeten oplichten. Op deze manier wordt de processor niet meer dan nodig is met de produktie van beelden belast.
De video-RAM heeft 16K byte nodig (die derhalve niet voor BASIC kunnen worden gebruikt) en ligt in het bereik van 49.152 tot 65.536. Daarmee bedekt hij ook dezelfde adressen als de BASIC-interpreter.
Trekken we deze 16K af van de 64K van de totale RAM, dan blijven er nog 48K over. Volgens onze berekening mankeert daar nog ca. 5,5K aan.

Elke processor gebruikt een zeker aantal bytes in het geheugen, dat de 'stack' (stapel) wordt genoemd, waar hij zijn 'persoonlijke' gegevens kan opslaan. Zo moet de Z-80 bij elke subroutine onthouden waarvandaan deze werd opgeroepen om aan het einde van de routine weer terug te kunnen springen. Dit gebeurt met behulp van de 'stack' (zie ook hoofdstuk 1.4.). Dit ligt in het gebied van 48.896 tot 49.151 en beslaat daarmee precies 256 bytes. U moet nooit met een POKE-opdracht naar dit gebied verwijzen. Het resultaat daarvan kan zijn dat uw computer zich 'ophangt', hetgeen slechts ongedaan kan worden gemaakt door de computer uit en vervolgens weer in te schakelen.

Om zowel in RAM als in ROM te kunnen werken, zijn de geheugenplaatsen in RAM van 0 tot 63 een kopie van de daar aanwezige ROM. Als de Z-80 op een zeker moment in RAM werkt en ROM-data nodig heeft, kan er door middel van een bijpassende routine naar deze 64 bytes worden overgeschakeld. Bovendien vindt u nog meer routines in RAM tussen het BASIC-geheugen en de video-RAM. U moet nooit proberen om in dit bereik een waarde door middel van POKE te veranderen. In de meeste gevallen zal de computer zich ook hier ophangen, dat wil zeggen, de computer reageert niet meer op het toetsenbord e.d. Ook dan blijft er niets anders over dan de computer uit te zetten!

Maar dat is nog niet alles. Het 'operating system' en de 'interpreter' hebben zelf ook een zeker aantal geheugenplaatsen nodig, bijvoorbeeld om tussentijdse uitkomsten van rekenkundige bewerkingen te kunnen 'onthouden', gegevens tussen de beide ROM-programma's te kunnen uitwisselen en invoer die via het toetsenbord plaatsvindt er tussen te kunnen schuiven, enz.

In dit verband is de toetsenbordbuffer heel interessant. Daarmee wordt het mogelijk om karakters in te voeren voordat BASIC deze in ontvangst neemt. Dit kunt u zelf testen. Tik eens het volgende programma(atje) in en start het daarna:

```
1 FOR i = 1 TO 10000:NEXT i
```

Als dit programma 'loopt', kunt u willekeurige toetsen indrukken. Zolang u de BREAK-toets niet aanraakt, gebeurt er niets op het beeldscherm. Zodra de lus echter zijn natuurlijk einde heeft gevonden, verschijnen de ingetoetste karakters op het beeldscherm. Terwijl het programma liep, heeft het bedrijfssyteem 20 tekens in het geheugen opgeslagen. Op deze manier kunt u bij lange berekeningen alvast de nodige voorbereidingen treffen voor de volgende INPUT-opdracht. Een 'BREAK' in het BASIC-programma zal echter de inhoud van de buffer geheel uitwissen.

Daarentegen worden deze tekens ook bij langdurige opdrachten (bijv. bij LOAD, enz., zij het met de nodige uitzonderingen) nog opgeslagen.

Als u een nauwkeuriger overzicht wilt hebben over de geheugen-opbouw van uw CPC464, kunt u de appendix van dit boek raadplegen. Daarin vindt u het gehele overzicht van het geheugen met de bijbehorende adressen.

1.4. POINTER EN STACKS

Twee vaktermen die u steeds opnieuw zult tegen komen, zijn 'pointer' en 'stack'.

Pointers (dat zijn wijzers) wijzen bepaalde delen van het geheugen aan en worden ook wel 'vectoren' genoemd. Daar kunnen informaties staan, maar ook een onderprogramma. De cursorpointer bijvoorbeeld wijst op de plaats van het beeldschermgeheugen waar de cursor staat en geeft daarmee aan waar het volgende teken komt te staan.

Een pointer in een subroutine wordt gebruikt om de werkwijze van een machinetaalprogramma flexibel te laten verlopen. Bij wijze van voorbeeld kan men een programma door middel van een tabel van subroutinepointers afhankelijk van de uitkomst de juiste vector kiezen, bij PRINT + 8 bijv. de achtste wijzer voor het aansturen van de printer (ook als dit niet precies zo werkt, is het toch een goed voorbeeld).

Pointers hebben een bepaald formaat. Ze bestaan in het algemeen uit twee bytes, waarvan de eerste een LOWBYTE (byte met een lage waarde) en de tweede een HIGHBYTE (byte met een hoge waarde) worden genoemd. Om de positie van de cursor of een adres te verkrijgen waarop gewezen wordt, gebruikt men de volgende formule:

Adres = LOWBYTE + 256 * HIGHBYTE

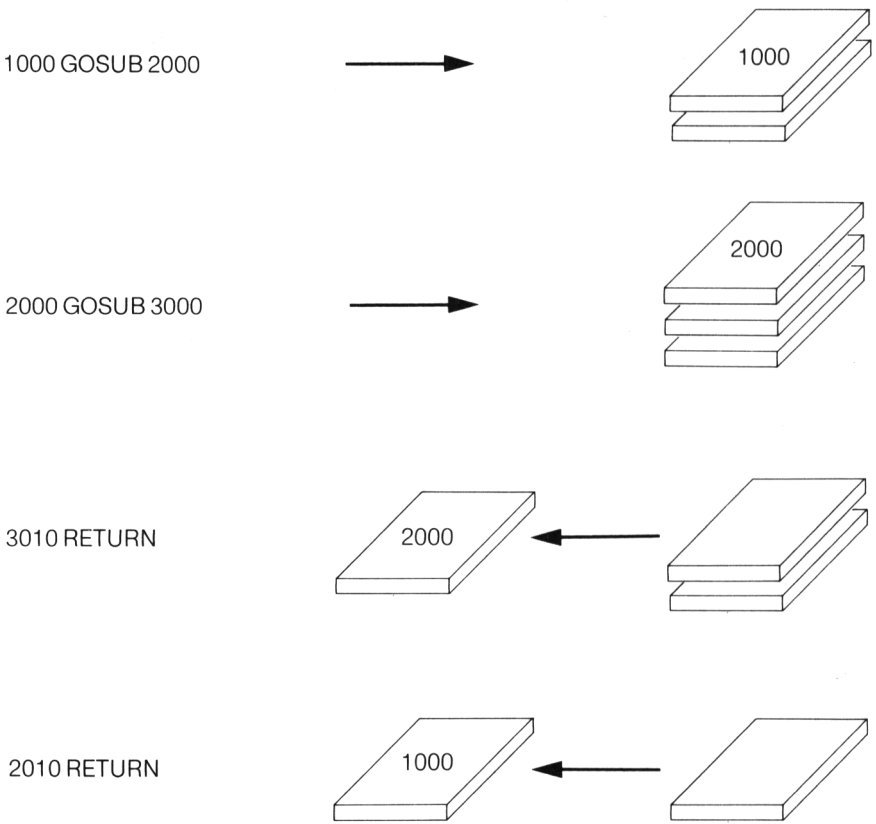
Als u in het hexadecimale systeem werkt, zet u eenvoudig de LOWBYTE op de rechterplaats en de HIGHBYTE op de linkerplaats, waardoor u een hexadecimaal getal krijgt van vier cijfers.

Het is een bijzonderheid van een computer dat de LOWBYTE altijd vóór de HIGHBYTE in het geheugen staat.

Een -byte-pointers wijzen binnen een bepaald bereik op een actuele positie (0-255) en worden bij een basisadres opgeteld.

Een stack (stapel) heeft de opdracht om gegevens tussen te schuiven en die dan in de omgekeerde volgorde er weer vandaan te halen als men ze nodig heeft. Zoals ook bij een echte stapel kan men in principe alleen het bovenste element wegnemen en er alleen maar een nieuw element bovenop leggen. Dit wordt in het bijzonder bij subroutines toegepast. Bij het aanroepen van een subroutine wordt de actuele plaats in het programma op de 'stack' geschoven en bij het commando 'RETURN' er weer afgehaald, waarbij het programma weer wordt voortgezet.

Vooropgesteld dat alle gegevens die op de stack zijn geplaatst er weer afgehaald worden voordat het nieuwe element aan de beurt is, springt elk onderprogramma gegarandeerd weer terug naar de juiste plaats. Een voorbeeld:



Bij machinetaalprogramma's werkt dat op dezelfde manier als bij BASIC. Overigens worden ook bij FOR...NEXT-lussen de plaatsen waarheen moet worden teruggesprongen op stacks geschoven. Dit is dan ook de reden waarom subroutines en lussen elkaar niet mogen overlappen. Als de terugsprong uit een subroutine 3000 voor de terugsprong van de subroutine 2000 plaatsvindt, geeft de computer een verkeerd adres op (2000 ligt immers bovenop!).

Samenvatting: Pointers
Adres = LOWBYTE + 256 * HIGHBYTE
LOWBYTE = Adres - INT(Adres/256)*256
HIGHBYTE = INT(Adres/256)
Pointers bestaan in het algemeen uit twee bytes, die altijd in de volgorde LOW/HIGH staan.

2. BEDRIJFSSYSTEEM EN INTERPRETER

Voordat we ons nader met de mogelijkheden van de CPC-464 gaan bezighouden, zullen we eerst een aantal vaktermen nader uiteenzetten. Tenslotte is het geen schande om tot nu toe niet te weten wat een bedrijfssysteem allemaal kan doen. Of wel?

Met deze en gelijksoortige thema's willen we ons in het volgende hoofdstuk gaan bezighouden.

2.1. BEHULPZAME GEESTEN VOOR HET KLEINE WERK

Elke computer heeft het nodig en diverse namen ervoor kent u al, maar slechts weinig beginners weten wat ermee bedoeld wordt - het bedrijfssysteem.

Zoals de naam al zegt, kan geen enkele computer buiten het bedrijfssysteem. Om bijvoorbeeld een byte van de processor naar de printer te sturen, is er steeds een klein machinetaalprogramma nodig. Weliswaar loopt deze overdracht automatisch, maar toch moet de interface geprogrammeerd en gestuurd worden, zodat fouten e.d. snel kunnen worden opgespoord.

In het algemeen stuurt een bedrijfssyteem zodoende de samenwerking tussen de computer en de diverse andere apparaten (zoals het toetsenbord, de printer, enz.). Voor elk apparaat bestaan er dus ook eigen subroutines, die voor de BASIC-interpreter of voor uw eigen programma's kunnen worden gebruikt. Om een karakter naar een dergelijk bereik te sturen, moet BASIC deze gegevens klaarzetten en dan de bijbehorende routine oproepen.

U hebt zeker wel eens van andere bedrijfssystemen, zoals CP/M of MS/DOS, gehoord (om de twee bekendste te noemen) en in dit verband misschien ook wel van 'standaardbedrijfssystemen'.

Deze systemen zijn zó opgebouwd dat men geen subroutines meer aanroept, maar dat in speciale registers de gegevens en commandonummers staan. Op deze manier kan een bedrijfssysteem, dat voor verschillende geconstrueerde computers steeds weer veranderd moet worden, met dezelfde codes op meerdere computers werken. Daarvoor moet men weten dat een machinetaal een onaangenaam neveneffect heeft. Heel vaak moeten, als een subroutine moet worden aangepast, ook de sprongadressen veranderd worden omdat het aantal bytes zelden gelijk blijft. Dat wil dus zeggen dat elke computer zijn eigen bedrijfssysteemroutines heeft en alle programma's die deze routines nodig hebben, moeten worden herschreven. Door deze commando's wordt derhalve het aantal opdrachten in de machinetaal met een aantal uitgebreid (en dat voor alle computers tegelijk). Dat betekent voor de softwarefabrikanten dat hun programma's niet voor elke nieuwe computer moeten worden herschreven.

2.2. DE INTERPRETER

De Engelse benaming voor dit machinetaalprogramma in ROM verraadt al het een en ander over de toepassingen ervan. De 'vertolker' (dat is de betekenis van 'interpreter') zet de BASIC-commando's om in acties van de processor. De Z-80 kan namelijk van huis uit alleen zijn speciale (voor elke microprocessor verschillende) machinetaal verwerken.

BASIC-commando's zijn voor hem in eerste instantie niet meer dan een aantal meer of minder samenhangende tekens. Volgt op deze rij tekens dan nog een ENTER, dan begint de BASIC-interpreter zijn eerste vertolkfase (voor science fiction-fans: red alert).

Vanaf nu heeft de processor zijn handen (dat wil zeggen: zijn ingangen) vol. De letters en cijfers moeten als BASIC-opdrachten, regelnummers en andere gegevens worden herkend. Commando's krijgen een speciaal codenummer waarmee de interpreter later bij het verwerken van het programma de juist subroutines eruit kan halen.

Tot nu toe zijn de diverse tekens slechts van een code voorzien en staan ze nog niet in het geheugen. Daarvoor moet misschien eerst nog ruimte worden gemaakt (wellicht is er later nog een regel aan toegevoegd), waarna ze eindelijk in het geheugen worden opgeslagen.

Dit alles gebeurt in een tijdsbestek dat zo gering is dat de programmeur het nauwelijks kan waarnemen. En dat alles tussen het indrukken van de ENTER-toets en het weer verschijnen van de cursor op de volgende regel.

Het tweede deel van het vertolken start met het RUNnen. De interpreter haalt dan achter elkaar de BASIC-opdrachten en de gegevens uit het geheugen en verwerkt die. Vindt hij bijvoorbeeld een PRINT-opdracht, dan wordt de machinetaalroutine 'PRINT' in werking gezet, waarmee de variabelen worden gelezen en het bedrijfssysteem de opdracht krijgt om ze af te drukken.

Omdat de commando's alle een speciale code hebben gekregen, hoeft na het intikken van RUN niet meer de volgorde P R I N T herkend te worden, maar moet de pointer het juiste onderprogramma uitzoeken. Hiermee wordt heel veel tijd bespaard.

2.3. SYSTEMATISCH ONDERBREKEN

Wat voor ons als onhoffelijk geldt, wordt door de computer als noodzakelijk ervaren. Want juist de 'interrupt' maakt vele toepassingen mogelijk die zonder dit detail onmogelijk zouden zijn.

Zowel de BASIC-interpreter alsook het bedrijfssyteem zijn machinetaalprogramma's. Beide worden door het inschakelen van de computer gestart en lopen zo lang door tot er een ander programma in de machinetaal wordt opgeroepen. Gebeurt dit laatste door het commando 'CALL', dan keert de computer na het beëindigen van de subroutine terug naar BASIC.

Zoals u van het programmeren in BASIC weet, kan een computer (met uitzondering van multiprocessorsystemen) slechts één bewerking tegelijk uitvoeren. De interpreter en het bedrijfssysteem zijn echter twee gescheiden programma's, die voor het uitvoeren van bepaalde opdrachten tegelijkertijd moeten lopen. Hoe wordt een dergelijk probleem opgelost?

De eenvoudigste mogelijkheid is om twee programma's bijna tegelijk te laten lopen. Steeds als het BASIC met een deel van zijn werk klaar is, wordt het bedrijfssyteem ingeschakeld en omgekeerd. Dit gebeurt bijvoorbeeld als er een hulpapparaat moet worden gebruikt. Het BASIC stelt daarbij de nodige informatie ter beschikking, die dan door het bedrijfssysteem wordt verwerkt. Dit betekent echter dat er via het toetsenbord niets meer gedaan kan worden, omdat daar geen informatie meer vandaan kan komen. Tijdens het werken van het bedrijfssysteem moet echter ten minste de ESC-toets werken (BREAK). Om dit probleem op te lossen, hebben de computerfabrikanten de interrupt (onderbreker) uitgevonden. Elke 1/50ste seconde onderbreekt de processor het lopende programma (interpreter, BASIC of andere programma's) en springt dan naar een routine die het toetsenbord onderzoekt of iets dergelijks. 'Ontdekt' de computer daarbij dat de ESC-toets is ingedrukt, dan wordt het lopende BASIC-programma afgebroken. Is er een andere toets indrukt, dan wordt dat in de buffer opgeslagen.

Voor de gebruiker lijkt het alsof het toetstenbord blijvend wordt onderzocht, want zelfs de snelste typist(e) kan nauwelijks 15 aanslagen per seconde maken. Voor de microprocessor lijkt de tijd tussen twee interrupts echter een eeuwigheid, omdat daarmee een snelheid wordt bereikt van zo'n 4.000.000 aanslagen per seconde en een machine-opdracht gemiddeld 8 tot 10 van zulke aanslagen nodig heeft voor het uitvoeren ervan. De processor kan dan duizenden van deze opdrachten uitvoeren voordat hij door een interrupt wordt gestoord. Na het onderzoek van het toetsenbord gaat de processor verder op de plaats waar hij werd onderbroken.

Bij een aantal commando's wordt de interrupt uitgezet, zoals bijvoorbeeld bij een cassettebewerking. Dit kan men zien omdat de eventuele alternerende kleuren ophouden met knipperen. Ook deze wisselende kleuren worden door de interrupt gestuurd, waarbij bij elke interrupt eenvoudig een andere kleur wordt ingeschakeld.

Ook in BASIC kan men deze interrupts toepassen, want met de commando's AFTER en EVERY kunt u uw eigen interruptroutines vervaardigen. Dit werkt overigens op dezelfde manier als in de machinetaal.

Komt de interpreter op een AFTER of op een EVERY, dan wordt er een 'timer' gestart, die na een bepaalde tijd, zoals een wekker, de interrupt uitzet. De interpreter laat dan alles staan en liggen en voert de interruptroutine uit.

Een interruptsignaal kan ook van een andere bouwsteen van de computer komen, om bijvoorbeeld mee te delen dat er van een andere interface gegevens moeten worden verwerkt. Het BASIC reageert echter alleen maar op de TIMER-signalen.

Een heel bijzondere vorm van onderbreking is het commando RESET. Hierdoor ontstaat een toestand waarmee de Z-80 reageert zoals bij het aanzetten van de computer. De computer begint met het programma op geheugenplaats 0, onafhankelijk van hetgeen er voordien in dat geheugen stond. Op geheugenplaats 0 staat echter niets anders dan de initialiseringsroutine, die het geheugen uitwist en andere belangrijke basisinstellingen aanzet. Behalve met SHIFT-CTRL-ESC kunt u dit vanuit BASIC met CALL 0 oproepen, maar wel opgepast! Belangrijke gegevens worden erdoor uitgewist en moeten daarom eerst op de cassetteband of op diskette worden opgeslagen.

2.4 NIET ALLEEN VOOR TOPPROGRAMMEURS: SPECIALE COMMANDO'S

Stelt u zich eens de volgende situatie voor: u vindt in een van de talrijke computertijdschriften een prima programma om in te tikken. U hebt inmiddels 20K LISTING ingetikt, maar bij de eerste poging eindigt het programma voortijdig met ERROR.
Er is niets aan te doen, u moet het functioneren van het programma goed begrijpen om de fout te vinden, als u althans niet elke ingetikte letter of elk cijfer afzonderlijk wilt vergelijken. Als die vreemde POKE-opdrachten er maar niet waren.! Deze worden toch niet door een normale programmeur gebruikt. Het wordt dus tijd om de geheimen van deze instructies te ontluiseren.

2.4.1. PEEK & POKE

Nemen we om mee te beginnen de POKE-opdracht. De syntax hiervan wordt bekend verondersteld: POKE adres, byte. Het adres kan een getal zijn tussen 0 en 65535, het byte ligt tussen 0 en 255. De bedoeling van deze opdracht is om de byte op het gegeven adres op te slaan. Dit kan voor veel toepassingen worden gebruikt en afhankelijk van het adres kan men ermee het beeldscherm vullen, een machinecode laten uitvoeren en nog veel meer. Men kan er mooi mee gaan knoeien in de computer, want zowel het bedrijfssysteem alsook de interpreter moet - gedwongen - op bepaalde data 'letten'. Hoe deze data eruitzien, kan men door PEEK te weten komen. Ook hiervan zal de syntax wel bekend zijn: PRINT PEEK (adres) 'toont' de byte die op het opgeroepen adres staat.

Belangrijk is, dat PEEK een functie inneemt en derhalve alleen maar in een opdracht kan werken (A = PEEK (..)) of iets dergelijks.

Deze twee commando's hebben de gemeenschappelijke eigenschap dat ze zich op de adressen richten. Het is derhalve de moeite waard bij deze commando's de opbouw van het geheugen te raadplegen, om te zien in welk gebied men bezig is. Meestal kan men daar ook te weten komen om welke functie het gaat.

2.4.2. CALL

We krijgen nu een commando dat eigenlijk alleen voor programmeurs in de machinetaal interessant is: CALL-adressen. Deze dienen voor het oproepen van programma's in de machinetaal.

Met het commando CALL wordt het adres opgeroepen waar het machinetaalprogramma gaat beginnen. Aan het einde van het machinetaalprogramma keert de interpreter vanzelf, zoals ook bij een subroutine, terug in het BASIC-programma.

Bovendien kan men ook nog gegevens aan het CALL-commando toevoegen waarmee slimme machinetaalprogramma's ook nog kunnen werken.

2.4.3. EEN UITSTAPJE NAAR DE BINAIRE REKENKUNDE

Bij de tot nu toe beschreven opdrachten behoren er nog enkele die u zeker al kent, maar waarvan de veelzijdigheid u tot nu toe verborgen bleef. Als eerste willen we hier AND, OR, XOR, en NOT noemen. Tot nu toe hebt u die alleen maar in de FOR...NEXT-constructie leren kennen, bijv. in de vorm:

IF A = 0 AND B = 0 THEN 100

In feite zijn ze echter bedoeld als logische verbindingen van variabelen en getallen. Hiervoor moet u weten dat de computer ook vergelijkingen als getallen behandelt. Probeer hiervoor maar eens het volgende:

PRINT (1 = 2)
PRINT (1 = 1)

Als de uitkomst van een dergelijke vergelijking 'waar' is, krijgen we als antwoord de waarde -1, als deze uitkomst echter 'onwaar' is, wordt het antwoord 0. In het binaire systeem ziet een '-1' eruit als:

1111 1111

Als u het meest links staande bit niet als voorteken ziet, staat hier eigenlijk 255. Maar wat heeft dit eigenlijk met BASIC te maken?

Een IF...THEN-constructie wordt altijd verlaten als de uitkomst van de term gelijk aan 0 is. Dit is ook met de volgende opdracht denkbaar:

IF 3*A THEN 110

De uitkomsten van dergelijke vergelijkingen worden eenvoudig met elkaar verbonden en de uitkomst daarvan bepaalt het verdere programma. Om de werkingswijze van deze verbindingen te begrijpen, maken we een klein uitstapje naar de binaire rekenkunde.

AND, OR, XOR en NOT zijn de zogenoemde BOOLE'SE bewerkingen, die voor het verbinden van de logische toestanden dienen. En zoals u weet, kunnen logische toestanden met bits heel eenvoudig uitgedrukt worden: 0 voor 'onwaar' en 1 voor 'waar'. Er worden steeds twee bits met elkaar verbonden. De uitkomsten hiervan ziet u in de hieronder staande tabellen.

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

XOR	0	1
0	0	1
1	1	0

Bij de operators AND en OR geldt: de uitkomsten zijn in elk geval 1 als de beide ingangsbits de waarde 1 hebben. Men kan beide functies als volgt onder woorden brengen: bij AND is de uitkomst alleen dan 1 als bit 1 EN bit 2 de waarde 1 hebben, bij OR: als bit 1 OF bit 2 de waarde 1 heeft.
Bij de operator XOR wordt de uitkomst alleen dan 1 als OF de ene bit de waarde 1 OF de ander bit de waarde 1 heeft.

Bij de operator NOT verandert deze functie eenvoudig de waarde van de ingangsbit.

NOT	0	1
	1	0

Tot zover gaat het goed. Helaas hebben wij als onbedorven BASIC-programmeurs nog een probleem. In BASIC hebben we weinig aan enkelvoudige bits. Hier hebben we met decimale getallen te doen. Om te berekenen moeten we als volgt te werk gaan:
1. Het getal in een binair getal omzetten.
Hoe dat gaat, staat in uw handboek.

De getallen 45 en 123 zien er binair geschreven zo uit:
45 = 0010 1101
123 = 0111 1011

2. Deze binaire getallen bit voor bit met elkaar verbinden.
In ons voorbeeld van 45 AND 123 gaat dat als volgt:

```
  0010 1101
AND 0111 1011
  0010 1001
```

3. De uitkomst hiervan moeten we dan weer in het decimale getallensysteem omzetten.
0010 1001 = 41

Men kan dit natuurlijk veel eenvoudiger doen door gewoon PRINT 45 AND 123 in te tikken. Maar op deze manier krijgt men wel veel meer inzicht.

Terecht zult u nu de vraag kunnen stellen waar dit allemaal toe dient. Behalve bij het verbinden van vergelijkingen worden deze commando's vaak gebruikt als men afzonderlijke bits wil beïnvloeden. Door de AND-verbinding met 254 wordt in elk geval de meest rechts staande bit uitgewist, met de OR-verbinding met 1 wordt deze bit in elk geval weer gezet. Probeert u dat eens met een aantal willekeurige getallen!

2.4.4. VERBINDINGEN MET 'BUITEN': POORT-OPDRACHTEN

Via de ingangen aan de achterzijde van uw CPC464 kunt u de computer met andere apparaten laten communiceren. Om dit ook vanuit BASIC mogelijk te maken, is er een aantal speciale opdrachten.

INP(poort) haalt een byte van de poort die tussen de haakjes staat. Het nummer van deze poort is niet identiek met de geheugenadressen. Hiervoor bestaat er een apart adresseringssysteem, dat men pas in samenwerking met een uitgebreide kennis van de machinetaal kan gaan gebruiken. Ondanks dat zal ik deze commando's toch behandelen, zodat u tenminste voor een deel weet wat daarachter steekt.

Met OUT poort, byte kan men een byte via de poort uitvoeren. Zoals u ziet, lijken deze commando's veel op de PEEK en POKE-opdrachten.

En nu hebben we nog de geheimzinnige opdracht:
WAIT poort,X,Y.

Hierbij is de opdracht dat de bits in het geheugen worden verwisseld. Hij moet namelijk wachten. En dat 'lust' een computer helemaal niet. Dit gebeurt door een voortdurende verbinding van de bytes. Komt de interpreter bij een WAIT-commando, dan leest hij allereerst een byte van de aangegeven poort. Dit getal wordt XOR met het getal Y verbonden. De uitkomst hiervan wordt dan AND met de waarde Y verbonden. Als deze uitkomst het getal 0 oplevert, wordt de hele procedure herhaald en anders gaat men met de volgende opdracht verder.

Er bestaat overigens nog een variant van deze WAIT-opdracht, waarbij de variabel Y geen betekenis heeft. Hierbij wacht de interpreter tot de byte van 'poort' AND X ongelijk 0 wordt.

2.5. COMMANDO'S DIE NIET IN HET HANDBOEK STAAN!

Wat men alleen bij de fabrikanten van andere computers verwachtte, is ook bij de CPC gebeurt. Daar is een programmeur met veel liefs en toewijding bezig geweest met een interpreter, heeft deze, na vele uren voor het zwarte kastje met het toetsenbord en het beeldscherm te hebben gezeten eindelijk zonder fouten en dan vergeet de auteur van het handboek om die noeste arbeid te beschrijven.

Dit commando heet MOD. MOD komt van 'modulo', een begrip dat bij de wiskundigen onder u wel bekend zal zijn. De modulo-functie levert de rest die ontstaat na een deling. Hier een paar voorbeelden:

10/4 = 2.5 of 2 rest 2
10 mod = 2
11 mod 4 = 3 (11/4 = 2 rest 3).

Men kan het woord MOD voor het deelteken '/' zetten en krijgt dan de rest van deze deling. Weliswaar functioneert MOD echter alleen maar met getallen en variabelen in het gebied (-32768 tot + 32767).

Met behulp van deze MOD kan het algoritme van Euclides (zie hoofdstuk 14) eenvoudig geprogrammeerd worden.

Het tegenovergestelde van MOD is de INTEGER-DIVISION. Deze wordt afgebeeld door de omgekeerde schuine streep en geeft uit de deling 11/4 als uitkomst 2. Ook deze INTEGER-deling is bedoeld voor het delen met gehele getallen.

Bij de bekendmaking van de CPC464 was er al sprake van dat er ROM PACK's konden worden aangesloten, waarbij deze programma's (bijv. spelletjes, programmeertalen, businessoftware e.d.) met behulp van een BASIC-opdracht konden worden gestart. Dit commando staat niet in het handboek. Maar ondanks dat bestaat het toch. In tegenstelling tot andere opdrachten bestaat het slechts uit een teken, dat u kunt bereiken door het 'slingeraapje' (dat is de kleine letter a met het staartje) samen met de SHIFT-toets in te drukken. Op het beeldscherm verschijnt dan een loodrechte streep. Deze streep deelt dan de computer mee dat er een ander deel van het geheugen moet worden opgeroepen. Hij weet dan echter nog niet welke ROM. Daarom bezitten deze een LABEL, dat is een soort naam waaraan het bedrijfssysteem het ROM herkent. Omdat u in uw CPC zonder uitbreiding slechts over het BASIC-ROM beschikt, is de enige naam die u mag gebruiken het woord 'BASIC'. Als u dit probeert, zult u zien dat de gehele procedure opnieuw begint met de mededeling 'BASIC 1.0'. Hierbij worden ook alle gegevens uitgewist.

Een ROM-bereik kan ook meer dan één LABEL bezitten, die dan, laten we zeggen, als een soort toegevoegde commando's werken. Zo bezit ook de 'Floppy-controler' (zie hoofdstuk 10.1) zo'n uitbreiding van de commando's.

Het BASIC van uw CPC is overigens zeer mededeelzaam. Er bestaat namelijk een functie die meedeelt op welk adres een variabele is opgeslagen. Voor het programmeren in BASIC is dit overigens niet zo interessant, maar bij het programmeren in de machinetaal kan dit toch wel heel nuttig zijn. Deze functie werkt overigens met de 'slinger a', PRINT a geeft het adres van de variabele 'a' (voor zover deze reeds bestaat, anders krijgt u de mededeling IMPROPER ARGUMENT)!

Een andere merkwaardigheid is de functie DEC$ (x,y). Deze wordt slechts een enkele keer vermeld en wel als commando dat bij BIN$ hoort. In ROM bestaat deze functie werkelijk, zoals bij het afLISTen van de commandowoordentabel te zien is. Er blijkt echter geen bijbehorende subroutine in de interpreter voor te zijn, waardoor alleen maar de mededeling SYNTAX ERROR verschijnt. Misschien had de programmeur eerst het idee om deze functie als tegenhanger voor BIN$ en HEX$ te gebruiken, maar koos hij later voor de uitdrukkingen & en &X.

3. HET GEHEUGEN

De CPC464 behoort wat zijn geheugencapaciteit betreft tot de uitblinkers onder de home-computers. Naast 64K RAM beschikt u ook nog over 32K ROM. Bovendien kan men deze geheugencapaciteit met speciale onderdelen nog immens vergroten. Hoe dit werkt en wat men ermee kan doen, willen we in de volgende hoofdstukken tonen.

3.1. BESCHERMEN VAN HET GEHEUGEN

Voor het BASIC staat, zoals al eerder is meegedeeld, 42, 5 kilobyte geheugen vrij ter beschikking. De machinetaalprogrammeurs onder u (en ook degenen die dat willen worden) hebben daar echt weinig aan. Want de BASIC-interpreter houdt helemaal niet van machinetaalprogramma's. Hij maakt een vrijmoedig gebruik van het geheugen. Als er een nieuwe BASIC-regel of variabele ontstaat, dan wordt eenvoudig het (voor hem) vrije gebied overschreven, waarbij echter niet de geheugenplaatsen van onder naar boven worden gevuld. Aan het begin van BASIC-geheugen staan de programmaregels, de strings staan aan het einde van de gereserveerde ruimte en de variabelen staan daartussenin.

Ergens daartussenin moeten deze plaatsen op elkaar 'stoten' en geeft de CPC een MEMORY FULL ERROR. Men kan dus nooit precies zeggen welke bytes in het geheugen nog vrij zijn en dat ook gedurende een BASIC-programma blijven.

Voor de machinetaalprogramma's moet men derhalve een bepaald aantal bytes reserveren. Hiertoe dient het MEMORY-commando, waarvan de werking zeer eenvoudig is. Het einde van het BASIC-geheugen wordt aangewezen door een wijzer, waaraan ook de interpreter zich tijdens zijn werk oriënteert. Het MEMORY-commando doet in feite niets anders dan deze wijzer naar onze wens te veranderen. Wordt dit op een lager adres gezet, dan bereikt het BASIC al veel eerder het einde van het geheugen.

Deze wijzer staat overigens in de geheugencellen &AE7B en &AE7C en u kunt hem, in plaats van met MEMORY, ook met POKE veranderen, indien er onder u mensen zijn die de redenering huldigen: 'Waarom makkelijk, als het ook moeilijk kan'.

Het commando '? HIMEM' voert de omgekeerde opgave uit. Hij noemt ons de stand van de tegenwoordige waarde van de pointer. Na het inschakelen van de computer staat die op 43903. Dit getal geeft het einde aan van het BASIC-geheugen, dat bij byte 368 begint.

Het MEMORY-commando wordt alleen bij waarden tussen 368 en 43903 uitgevoerd; op ander plaatsen ontstaat de foutmelding MEMORY FULL ERROR. Dit gebeurt om het overschrijven van bedrijfssytemen te verhinderen. Bovendien wordt hiermee voorkomen dat er een ongewild uitwissen van een BASIC-programma plaatsvindt. Wordt de geheugengrens dermate ver naar beneden gezet dat daardoor een reeds bestaand programma wordt aangetast, dan krijgen we eveneens een MEMORY FULL ERROR. Deze blokkade kan men gemakkelijk (maar niet erg zinvol) omzeilen door middel van een POKE-commando.

Als u 256 bytes voor een machinetaalprogramma wilt reserveren, voert u eenvoudig 'MEMORY 43647' in. Nu kunt u vanaf geheugenplaats 43648 uw machineprogramma opslaan (MEMORY en HIMEM hebben betrekking op de laatste byte van het BASIC-geheugen)!

Samenvatting beveiliging van het geheugen
Met het commando MEMORY kan men de bovengrens van het geheugen omhoog of omlaag brengen. Het BASIC-geheugen bevindt zich tussen de adressen 368 en 43903. Met HIHEM kan men de aanwezige bovendien testen.

3.2. HOE FUNCTIONEERT HET 'BANKSWITCHEN'?

Zoals u uit de voorgaande hoofdstukken weet, liggen op een aantal plaatsen het ROM- en het RAM-geheugen naast elkaar. De processor kan echter slechts een van deze twee geheugendelen oproepen. Er moet dus tussen RAM en ROM worden overgeschakeld. Met een speciale logische schakeling worden de geheugenadressen, die op de adresingang arriveren, geanalyseerd, waarna de bijbehorende geheugenbouwstenen worden ingeschakeld.

Deze schakeling kan men van buitenaf beïnvloeden, dat wil zeggen, de processor kan beslissen over het ingaan en het uitgaan (zoals ook bij INP en OUT), waar dan de adressen worden gedecodeerd. Soms worden dan de RAM-bouwstenen ingeschakeld en soms die van ROM. Bovendien kunnen de twee ROM-bereiken gescheiden worden geschakeld. Zo is het mogelijk dat alleen het bedrijfssysteem actief is, terwijl met het video-RAM gewerkt wordt. Deze logische schakeling werkt dus als een wissel.

Ook vanuit BASIC kan men overschakelen op het ROM; het resultaat is dan echter dat de computer zich in de meeste gevallen ophangt. Probeert u dat eens! Met OUT &7F82,&82 kunt u beide ROM-gedeelten inschakelen.

Zoals u hebt gezien, is het overschakelen van het geheugenbereik vanuit BASIC niet aan te bevelen. Dat is ook geen wonder, want met het 'bankswitchen' krijgt de Z-80 een heel ander geheugen voorgeschoteld. In vele gevallen werkt de interpreter in RAM. Schakelt u nu over naar ROM, dan vindt de interpreter zijn programma niet meer terug. Resultaat: ophanging.

3.3. ROM UITLEZEN

Soms kan het nuttig of zelfs noodzakelijk zijn gegevens uit het ROM te lezen, bijv. de tekens (die staan overigens in het gebied van &3800 tot &3FFF). Vanuit BASIC is dat niet mogelijk. De PEEK-functie werkt alleen in RAM en het omschakelen van RAM naar ROM eindigt meestal met het verlies van een programma dat met veel moeite is opgebouwd. Zuiver theoretische is het mogelijk om een BASIC-programma te schrijven voor het uitlezen van het ROM, maar de CPC reageert daar zeer allergisch op. Ondanks dat zal ik u toch dat programma geven.
a. omdat hierdoor het principe duidelijk wordt, en
b. het meegeleverde machineprogramma niet meer hoeft te worden verklaard omdat dit een (bijna) woordelijke vertaling is.

Het BASIC-programma luidt:

```
1 OUT &7F82,&82
2 A = PEEK (X): REM X = gewenst adres
```

Het equivalent in de machinetaal ziet er als volgt uit:

```
1 DATA &01,&82,&7F,&ED,&49
2 DATA &1A,&32,&7F,&AB,&C9
3 MEMORY &AB6F: FOR i = &AB70 TO &AB79
4 READ a: POKE i,a: NEXT: END
5 REM x = gewenst adres
6 CALL &AB70,x
7 a = PEEK(&AB7F): RETURN
```

Voor de toepassing van deze machineroutine wil ik nog toelichten dat de regels 1 tot 4 het programma initialiseren, dat wil zeggen, hier worden de machinetaalcommando's in het geheugen gezet (in de POKE-lus). Dit deel wordt alleen in het begin van het programma uitgevoerd, daarna staat het machinetaalprogramma in het geheugen en kan het worden toegepast. De eigenlijke commando's staan in de DATA-regels (10 bytes).

Als u nu een byte uit het ROM wilt lezen, moet u dit adres in de machineroutine meedelen. Het adres komt in de variabele x te staan en met GOSUB 6 wordt het tweede deel van het programma gestart.
In regel 6 wordt dit oproepen overgenomen door CALL &AB70,x. Omdat de machinetaalprogramma's niet zonder meer gegevens aan het BASIC kan doorgeven, lossen we dit op door de gezochte waarde uit het ROM op een speciale plaats in het geheugen te zetten, waar we het dan met PEEK(&AB7F) vandaan kunnen halen. Dit gebeurt in regel 7.
Vanzelfsprekend kunt u de namen van de variabelen en de regelnummers veranderen.

3.4 GEHEUGENUITBREDING

In de advertenties staat vaak de opmerking dat het geheugen van de CPC kan worden uitgebreid. Dat kunnen bijv. 'ROM PACK'S' zijn, waarop kant en klare programma's staan, zoals bijv. tekstverwerkers, maar het is ook mogelijk om het geheugen van RAM uit te breiden.

Belangrijk is echter dat deze geheugenkaarten via de logische schakeling aan de adresingang wordt aangesloten. Daardoor is het mogelijk deze geheugenplaatsen door middel van 'bankswitching' als een ingebouwde ROM op te roepen. Voor de Z-80 maakt dat in het geheel niets uit, hij moet alleen de getallen in de OUT-opdrachten op overeenkomstige wijze veranderen.

Ook de 'Disk-drive' bezit een dergelijk uitbreidings-ROM, waarin de commando's voor de bediening zijn opgeslagen.

4. TRUCS VOOR HET BEELDSCHERM

De CPC behoort ook bij het vervaardigen van grafische afbeeldingen tot de voorlopers. De ingebouwde BASIC-commando's kennen al een uitgebreide toepassing, maar er zijn nog een aantal verborgen mogelijkheden, die hierna verklaard worden.

4.1 BEELDSCHERMBESTURING DOOR CHR$-COMMANDO'S

Is de tabel met de stuurtekens in hoofdstuk 9 van het handboek al opgevallen? Al deze tekens kunt u als toevoeging op de beeldschermbesturing gebruiken, want hiermee wordt direct een aantal bedrijfssysteemroutines in werking gezet. Ook de BASIC-interpreter heeft niet alleen deze functies, maar geeft het bedrijfssysteem eenvoudig de bijbehorende stuurtekens door.

Hierna worden overigens alleen díe stuurtekens meegedeeld die ook werkelijk zinvol te gebruiken zijn. Wat heeft het voor zin om een tweede LOCATE-commando uiteen te zetten dat bovendien nog moeizamer te hanteren is dan de BASIC-opdracht?

Al deze stuurtekens hebben de eigenschap dat ze met een PRINT CHR$(x) kunnen worden opgeroepen. De PRINT-opdracht zorgt ervoor dat deze tekens bij de juiste bedrijfssyteemroutine terechtkomen, de CHR$-funcite helpt ons dan deze code als getal aan te geven, omdat hier anders een grafische teken zou moeten staan. Dit laatste werkt overigens ook niet altijd, omdat een bedrijfssysteem de grafische en de stuurtekens verschillend behandelt!

Met het stuurteken nr. 1 (SOH) kunt u andere stuurtekens zichtbaar maken. Zoals u weet, worden tekens waarvan de ASCII-code kleiner is dan 32 niet afgedrukt, maar worden ze als stuurteken naar het bedrijfssysteem geleid. Zet u echter eerst de code CHR$(1) ervoor, dan wordt er een teken afgedrukt dat bij de besturing hoort; bijv. voor LF (LINE FEED of doorschuiven van de regel) een pijl naar beneden. Probeert u eens

```
PRINT CHR$(1)CHR$(10)
```

SOH heeft betrekking op de navolgende code. U moet dus voor elk teken dat moet worden afgegeven een nieuw SOH tussenvoegen.

Ook het stuurteken met het nummer 2 heeft een zeer interessant effect. Met STX (dat is de naam hiervoor) kan de cursor worden uitgeschakeld. Dit werkt echter alleen als het programma loopt. Als het BASIC in de directe vorm werkt, wordt na elke mededeling 'READY' de cursor vanzelf weer ingeschakeld. Zoals hieronder is weergegeven, functioneert het echter:

```
10 PRINT CHR$(2)
20 INPUT ''TEXT;A$
30 PRINT CHR$(3)
```

Zoals u zult zien, verschijnt bij de INPUT-opdracht niet meer de normale cursor.

De PRINT-opdracht in regel 30 heeft precies het tegenovergestelde als resultaat; hiermee wordt de cursor weer aangezet (dit is in ons voorbeeld eigenlijk overbodig, omdat dit aan het einde van een programma al vanzelf gebeurt).

Het stuurteken nr.7 kent u al uit het handboek en daarom zal ik u dit gepiep verder besparen.

Komen we dan nu op de stuurtekens waarmee de cursor over het beeldscherm kan worden bewogen. CHR$(8) werkt als de toets met de pijl naar links, CHR$(9) als de toets met de pijl naar rechts, CHR$(10) naar beneden en CHR$(11) naar boven. Dit kan voor sommige toepassingen heel nuttig zijn, als u bijv. een index wilt afdrukken. Met de beschreven codes gaat u dan eenvoudig naar de bedoelde plaats, laat daar de index afdrukken en gaat dan weer op dezelfde manier terug naar de regel waar u zojuist vandaan komt.

Bij de eerste oogopslag lijkt het stuurteken 12 overbodig. Hiermee kan het beeldscherm zoals met CLS worden uitgewist. Er doen zich echter geheel nieuwe mogelijkheden voor als men een of meer stuurtekens in een normale BASIC-string opneemt. Probeert u het volgende maar eens:

```
A$ = CHR$(12) + ''OPSCHRIFT''
```

Elke keer als deze string (A$), wordt opgeroepen (eenvoudig door PRINT A$) wordt allereerst het beeldscherm uitgewist en dan verschijnt de tekst. In beginsel kan men alle hierboven vermelde stuurtekens op dezelfde manier in strings opnemen.

Het volgende stuurteken voor de cursor is CR (CHR$(13)). Hiermee kan men de cursor naar het begin van de regel schuiven. CR betekent CARRIAGE RETURN, wat zoveel wil zeggen als 'zet de cursor naar het begin'.

Ook de functie van CHR$(16) is u inmiddels bekend. Daarmee kunt u de CLR-toets vanuit BASIC nabootsen en het teken onder de cursor verwijderen. Probeert u zelf eens:

```
10 CLS
20 LOCATE 20,10
30 PRINT ''ABCDEFGHIJK'';:
40 WHILE INKEY$ = '' '':WEND
50 PRINT CHR$(8)CHR$(8)CHR$(8)CHR$(8)CHR$(8)CHR$(16)
```

Dit programma hoeft niet verklaard te worden; u start het eenvoudig, drukt daarna een willekeurige toets in en zie wat er met de stuurtekens in regel 50 gebeurt.

Bij de eerste oogopslag lijkt dit niet bepaald zinvol, maar ik wil verder aan u overlaten gebruik te maken van deze nieuwe mogelijkheden en nuttige toepassingen te bedenken.

De stuurtekens 17 tot 20 zijn bedoeld om het beeldscherm uit te wissen. Vervangt u eens, om er een idee van te krijgen, in regel 50 het laatste stuurteken door CHR$(17). Als u nu het programma start, worden alle tekens tot de cursorpositie uitgewist, wat u door het verdwijnen van de letters ABCDEFG kunt zien. Als u nu CHR$(17) vervangt door CHR$(18), gebeurt precies het tegenovergestelde. Nu worden de letters GHIJK en de rest van de regel (die bij ons leeg is) uitgewist.

De stuurtekens CHR$(19) en CHR$(20) werken op een vergelijkbare manier, alleen CHR$(19) wist alle tekens uit vanaf het begin tot de cursorpositie en CHR$(20) doet dat vanaf de cursorpositie tot het einde (de rechteronderhoek). Dit kunt u gemakkelijk testen door in dit programma meer PRINT-opdrachten op te nemen, waardoor ook op andere regels een tekst komt te staan.

Maar nu iets anders dan de cursorbesturingstekens. Het volgende stuurteken is wel iets heel bijzonders. Met PRINT CHR$(21) kunt u het beeldscherm uitschakelen, wat niets anders betekent dan, dat er op het beeldscherm geen enkele mededeling meer wordt gedaan. Dit kan heel nuttig worden toegepast bij het invoeren van een Password, zoals u in het hieronderstaande programma kunt zien:

```
10 PRINT''GEEF HET TOEGANGSWOORD''CHR$(21)
20 INPUT A$
30 IF A$ = ''XYZ12'' THEN GOTO 1000 ELSE NEW
1000 PRINT CHR$(6)
....
....
....
```

Nadat regel 10 is uitgevoerd, worden alle teksten onderdrukt, dat wil dus zeggen dat ook de tekst die hierna wordt ingetoetst, niet op het beeldscherm zal verschijnen (regel 20), en zo kan ook geen onbevoegde uw codewoord meelezen. Overigens werkt de INPUT-opdracht heel normaal. In regel 1000 wordt de werking van CHR$(21) weer ongedaan gemaakt.

Ook het teken SYN (code 22) heeft een interessant effect. Als daar een 1 op volgt, wordt de transparantvorm ingeschakeld. In deze vorm wordt de plaats op het beeldscherm, waar het nieuwe teken moet komen te staan, niet eerst uitgewist. Het oude teken blijft staan en het nieuwe teken wordt er gewoon overheen geschreven. Dit kan bijv. gebruikt worden als men een letter of iets dergelijks wil onderstrepen (wat bij de meeste computers niet mogelijk is). Als u na het afdrukken van de tekst de cursor terugzet, dan de transparantvorm door ''PRINT CHR$(22) CHR$(1);'' inschakelt en het teken afdrukt waarmee een letter wordt onderstreept (SHIFT 8). Met CHR$(22) CHR$(0) wordt dit weer uitgezet.

Heel nuttig is ook het stuurteken 24. Hiermee wordt eenvoudig de PAPER- en PEN-kleur verwisseld. De uitkomst hiervan is dan een inverse afbeelding van het karakter, dus als tot nu toe een letter geel op een blauwe achtergrond werd afgedrukt, verschijnt nu een blauwe letter op een gele achtergrond.

Het laatste stuurteken is weer een cursorstuurteken. Met PRINT CHR$(30) wordt de cursor in de linkerbovenhoek van het scherm gezet. Deze functie noemt men ook wel HOME.

Een kleine oneffenheid is echter ook hier aanwezig. Als de PRINT-opdrachten door TAG de grafische cursor aanzet, worden alleen maar de grafische tekens afgedrukt; de funkties van de codes blijven tot TAGOFF uitgeschakeld.

Samenvatting: CHR$-codes.

Code(s)	Functie
1	Afdrukken van het stuurteken
2/3	Tekstcursor uit/aan
7	Beep
8	Cursor een plaats terug
9	Cursor een plaats naar voren
10	Cursor een regel omlaag
11	Cursor een regel omhoog
12	Beelscherm uitwissen
13	Cursor naar begin van de regel
16	Zoals de CLR-toets
17	Regel tot de cursor uitwissen
18	Regel vanaf de cursor uitwissen
19	Beeld tot de cursor uitwissen
20	Beeld vanaf de cursor uitwissen
21/6	Tekstscherm in/uit
22	Transparantvorm in/uit
24	Inverse
30	HOME

4.2. VIDEO-RAM 'VANBINNEN'

Wat weet u van het video-RAM? Tot nu toe nog niet veel. Maar dat zal met de volgende regels wel veranderen. Uit de voorgaande hoofdstukken weet u al dat de bytes in het video-RAM de punten op het beeldscherm vormen. De eerste vraag is dan: Hoe staan deze punten in het geheugen? Om dit te kunnen beantwoorden, gaan we over tot een klein experiment. Zet daartoe eerst even uw computer uit en weer aan, zodat alle interne gegevens de stand krijgen die wij graag willen hebben. Voer dan in:

```
MODE 2
FOR i = 48152 to 65535: POKE i,255: NEXT
```

Met deze opdracht krijgen alle bits van het video-RAM de waarde -1, waarmee alle punten van het beeldscherm tot oplichten worden gebracht. Na het indrukken van de RETURN-toets zult u zien dat als eerste de bovenste punten van een tekstregel worden 'gezet', daarna alle punten daaronder, enz., tot het gehele beeldscherm één vlak voorstelt.

Bedenk eens dat we alle tekstregels op elkaar kunnen stapelen, dan krijgen we in het totaal 8 rijen met punten (omdat immers in de vorm MODE 2 elk teken uit 8 × 8 punten bestaat). Delen we 16 Kilobytes (zo groot is het beeldschermgeheugen) door 8, dan is de uitkomst hiervan 2K of 2048 bytes voor elke regel. Als u nu alle 8 puntenregels achter elkaar legt, dan hebt u de interne organisatie van het beeldschermgeheugen voor u. Hiermee is echter nog niet verklaard hoe men een enkele punt kan uitbeelden.

Als - zoals in MODE 2 - slechts twee kleuren (de punt licht op of hij blijft donker) worden onderscheiden, is een bit per punt voldoende (0 voor donker en 1 voor licht). Bij 640 × 200 punten zijn dat precies 128000 bits of 16000 bytes.

De 16 Kilobytes (dat zijn 16384 bits) zijn dan eigenlijk 384 bits meer dan we voor het beeldscherm nodig hebben. Daarvoor zijn er twee verklaringen. Ten eerste is het voor de processor en voor de TV-chip gemakkelijker om met gehele Kilobytes te werken dan met gebroken Kilobytes (zelf werkt u ook liever met gehele getallen dan met breuken). Bovendien heeft de processor voor een aantal bewerkingen ook een paar bytes nodig (zoals bijv. voor het scrollen, maar daarover later). Zodoende zijn er aan het eind van elk 2K-blok nog 48 bytes ongebruikt.

In MODE 1 staan er vier kleuren ter beschikking. Daarom worden er voor elk punt twee bits gebruikt. De mogelijke combinaties zijn hierbij 00, 01, 10 en 11. Omdat er nu twee keer zoveel bits nodig zijn, kunnen we slechts de helft van de punten bereiken en daarom zijn deze twee keer zo breed en worden ook voor deze 320 × 200 punten 16000 bytes gebruikt.

In MODE 0 bestaan er 16 kleuren. Om nu 16 verschillende soorten punten te kunnen onderscheiden, hebben we slechts een halve byte nodig. Omdat ook hier het aantal bits is verdubbeld moet ook het aantal punten worden gehalveerd. We hebben dan nog 'slechts' 160 × 200 punten.

Maar alle geheimen van het video-RAM zijn nog niet opgelost. In MODE 2 is het duidelijk welke bits bij welke punten horen. Het eerste byte van het video-RAM bepaalt het uiterlijk van de eerste 8 punten. Met POKE 49152, x kunt u een willekeurig bitraster in het geheugen opslaan, dat dan ook op dezelfde wijze op het beeldscherm komt te staan (zolang de CPC nog in de MODE 2 vorm staat). Probeert u dat eens met de waarden 1, 2, 4, 8, 16, 32, 64, 128, 240, 15 en 170 en neem daarbij het plaatje voor ogen van de opstelling van de nullen en de enen van het bijbehorende binaire getal. U zult dan zien dat de punten die oplichten in overeenstemming zijn met het binaire getal.

In MODE 1 heeft elk punt twee bits nodig. Het zou het meest voor de hand liggen als hiervoor twee naast elkaar liggende bits zouden bepalen welke kleur eraan wordt gegeven. Maar helaas is dat niet zo. Allereerst maakt de video-chip een onderscheid tussen het linkerdeel en het rechterdeel van een byte. Een half byte heet ook wel Nibble. Schrijf eerst de rechternibble onder de linker (In het voorbeeld hieronder staan alleen de bits); de waarden moet u zelf invoeren:

7 6 5 4
3 2 1 0

De onder elkaar liggende bits bepalen de kleur van een punt. Met byte 1111 0000 hebben alle vier de punten de kleur 01 (dit wordt van onder naar boven gelezen). Met byte 1111 1010 hebben het eerste en het derde punt de kleur 3 (binair 11), en de twee andere punten de kleur 1 (binair 01). Probeer dat eens met de bijbehorende POKE-opdrachten.

MODE 0 wordt op een vergelijkbare manier tot stand gebracht. Schrijf hiervoor de bits zoals in het onderstaande voorbeeld onder elkaar:

7 6
5 4
3 2
1 0

Zoals u weet, wordt door een byte in deze vorm een tweetal punten aangeduid. Lees de bitcombinatie ook hier weer van onder naar boven en u krijgt dan het kleurnummer van die punt. Welke kleuren er verschijnen, kunt u in de tabel in uw handboek zien (voorzover u dat niet met INK hebt veranderd).

4.3. VERBORGEN GRAFIEKEN

Bij het programmeren van grafieken, teksten en dergelijke wil men vaak eerst het totale beeld opbouwen voordat het zichtbaar wordt. Of u wilt twee volledig onafhankelijke grafieken afbeelden. Hiervoor zijn er twee mogelijkheden. Zo kan men de grafiek in een donkere kleur opbouwen (bijv. met INK 2,0) en als deze helemaal klaar is, de kleur overschakelen naar helder (bijv. met INK 2,24). Het andere beeld kan men dan tegelijkertijd op dezelfde manier laten verdwijnen.

Dit kan men echter alleen doen als de beelden niet met elkaar botsen en als de MODE niet 2 is (in MODE 2 kan men geen reservekleuren toepassen).

Gemakkelijker kan het zijn om de CPC een ander beeldschermgeheugen toe te wijzen. Dit kost echter heel veel BASIC-geheugenruimte. Het is ook vanzelfsprekend dat dit beeldschermgeheugen alleen maar op een zodanige plaats kan komen te staan dat reeds aanwezige informatie, zoals bijv. het bedrijfssysteem, niet wordt beschadigd. Bovendien kan het video-RAM alleen maar in stappen van 16K worden verschoven, dus naar de beginadressen 0, 16384, 32768 en 49152. Deze laatste mogelijkheid ontstaat al bij het inschakelen van de computer. In de gebieden 0 - 16383 en 32768 - 49151 hebben zowel het bedrijfssysteem als de BASIC-interpreter belangrijke gegevens ondergebracht. Derhalve blijft er voor ons doel alleen nog maar het gebied van 16383 - 32768 over.

Helaas moeten we dan ons BASIC-gebied met MEMORY 16383 begrenzen, waardoor we minder dan 16K overhouden voor het programma en deze methode alleen maar voor kleine programma's gebruikt kan worden.

Met CALL &BC06,&40 kan men het beeldschermgeheugen naar 16384 brengen. CALL &BC06,&C0 brengt ons weer naar de uitgangspositie terug.

Alle grafiekopdrachten hebben slechts betrekking op het beeldschermgeheugen dat zojuist is ingeschakeld. U kunt dus heel normaal werken.

Maar ook dat kan veranderd worden. Het kan gebeuren dat men in het beeld dat nog verborgen is, wil werken waarbij het andere beeld nog zichtbaar blijft. Gelukkig is er een geheugencel in het RAM waar het bedrijfssysteem het beeldschermadres waarneemt. Moet er een tekst of een punt ontstaan, dan houdt de computer zich niet aan het werkelijke adres maar aan de waarde van deze geheugencel. Zo kan men het bedrijfssysteem een nog niet bestaand beeldschermgeheugen voorschotelen. Deze geheugencel heeft het adres &B1CB. Zolang daar de waarde staat die met de CALL-opdracht daar is neergezet, functioneert het uitvoeren van het beeldscherm normaal.

CALL &BC06,&40: POKE &B1CB,&C0 schakelt het video-RAM vanaf adres 16343 in; het uitvoeren vindt echter in het oorspronkelijke geheugen plaats.

De POKE-opdracht kan natuurlijk ook alleen worden gegeven, belangrijk is dat alleen de waarden &40 of &Co gePOKED worden, omdat de computer zich anders ophangt.

Helaas moeten we ook nog een andere merkwaardigheid van de CPC in acht nemen. Als het beeldscherm geSCROLLd moet worden, wordt niet de inhoud van het video-RAM byte voor byte in het geheugen verschoven, maar wordt het begin van het video-RAM voor de TV-bouwsteen veranderd en begint bij 53248 in plaats van bij 49152 (om eens een willekeurige waarde te noemen). Door de logica van het adresseren worden de bytes die er aan de 'achterzijde uitvallen ', er aan de 'voorzijde' bij 49152 weer ingeschoven, zodat er zich voor ons op het eerste gezicht geen verandering heeft voorgedaan. Als we echter in dat beeldscherm POKEn, dan kunnen we zien dat een punt niet linksboven maar op een andere plaats verschijnt. Voor de computer een daarmee ook voor u) heeft deze methode het voordeel dat het scrollen veel sneller gaat.

Vanwege deze eigenzinnige techniek is het aan te bevelen om aan het begin van een BASIC-programma in elk beeldschermgeheugen een MODE-commando in te voeren. Daarmee worden de pointers weer teruggezet. Het is niet voldoende om met CLS of met CLG uit te wissen. In het verdere verloop van uw programma moet u er nog zorgvuldig voor zorgen dat er geen regels gescrolld worden.

Samenvatting: beeldscherm verplaatsen

CALL &BC06,&40 schakelt het bereik van 16384 tot 32767 als nieuw video-RAM in (eerst met MEMORY).
CALL &BC06,&C0 schakelt weer terug.
Op de geheugencel &B1CB 'ziet' het bedrijfssysteem welk video-RAM is ingeschakeld.

4.4 OPSLAAN VAN HET BEELDSCHERM

Zoals u weet, heeft het CPC-BASIC een speciale opdracht om verschillende delen van het geheugen te SAVEn. Hiermee kan men ook de inhoud van het beeldschermgeheugen op een cassette opslaan en weer teruglezen. De commando's hiervoor zijn:

```
SAVE "!TV",B,49152,16384
LOAD "!TV",49152
```

U kunt natuurlijk ook een andere naam voor het bestand kiezen, maar u kunt het uitroepteken beter niet weglaten, omdat anders alle mededelingen van het bedrijfssysteem ook op het beeldscherm verschijnen.

Het is ook mogelijk om slechts delen van het bereik op te slaan. Daarvoor moet voor elke regel het begin- en het eind-adres worden berekend. Met de kennis uit hoofdstuk 4.2. zal dat echter geen al te grote problemen voor u opleveren. Elke regel wordt dan met zijn eigen SAVE-commando op de band gezet en kan daar met LOAD weer worden verwijderd. Dit werkt natuurlijk alleen maar als er niet gescrolld is (dit probleem kent u al uit hoofdstuk 4.3.).

4.5. SCROLLEN

Elke bezitter van een computer heeft het fenomeen 'scrollen' wel eens opgemerkt. Als de cursor op de onderste regel van het beeldscherm is gearriveerd, wordt de totale beeldscherminhoud een plaats omhooggeschoven om plaats te maken voor een nieuwe regel. Daarbij zijn er tegelijk twee dingen die voor de BASIC-programmeur interessant kunnen zijn. Ten eerste is er de manier waarop het scrollen tot stand wordt gebracht, ten tweede wordt scrollen niet alleen maar in de verticale richting uitgevoerd, maar kan het ook in de horizontale richting worden gedaan. Hiermee kan men dan zeer interessante effecten bereiken.

Analoog aan de bestaande situatie kan de bestaande tekst naar beneden gescrolld worden als op de bovenste regel het stuurteken 'Cursor omhoog' (CHR$(11)) staat. U kunt dan de ontstane regel eenvoudig met PRINT opvullen. Maar ook in de verticale richting kan men het gehele beeldscherm een plaats verschuiven. Daarvoor heeft men twee OUT-commando's nodig, waarmee de OFFSET van het begin van het video-RAM wordt veranderd. Zoals u uit het vorige hoofdstuk weet, heeft het scrollen geen verandering van het beeldschermgeheugen ten gevolge maar wordt alleen het beginadres met de lengte van een beeldschermregel verschoven. Deze verschuiving noemt men 'offset'. Gelukkig kan men deze offset ook in kleinere stappen dan een regel veranderen. De kleinst mogelijke verandering bestaat uit twee bytes. Daarbij behoren dan, afhankelijk van de beeldscherm-MODE, een halve, een hele of twee tekens. En zo is ook het horizontale scrollen mogelijk.

De 6845-videocontroler bezit diverse registers waarin deze offset kan worden opgeslagen. Met een OUTPUT-commando kan men in deze registers schrijven, maar met INP lezen is niet mogelijk. De opdracht daarvoor luidt:

```
OUT &BC00,13:OUT &BD00,offset
```

In het normale geval (als er dus geen scrollen aanwezig was) heeft de offset de waarde 0. Door het vergroten van dit getal wordt naar links geschoven en door het verkleinen van dat getal naar rechts door. Als u vanuit het midden naar links en naar rechts wilt verschuiven, is het aan te bevelen om voor de eerste PRINT-opdracht het beeldscherm uit te wissen en dan met behulp van OUT &BC00,13: OUT &BD00,20 alles naar het midden te scrollen.

Een hele regel doorschuiven kan ook met de volgende lus worden gedaan:

```
OUT &BC00,13: FOR i = 0 TO 40: OUT &BD00,1: NEXT
```

Het eerste OUT-commando dient voor de juiste keuze van het desbetreffende 6845-register. Daarom moet dat in de lus niet steeds opnieuw herhaald worden. Omdat echter ook het bedrijfssysteem in dit register staat, kan het zijn dat het registernummer niet meer klopt. U kunt daarom niet buiten het OUT &BC00,13 voor elk 'do-it-yourself-scrollen'!

Samenvatting: scrollen

Scrollen naar beneden kan met CHR$(11) op de bovenste plaats van het beeldscherm worden gedaan.
Voor het horizontale scrollen geldt:
OUT &BC00,13: OUT &BD00, offset
waarbij 'offset' het aantal bytes gedeeld door twee is dat men wil verschuiven.

4.6. 'SCROLLEN', MAAR NU ANDERS

In de volgende regels zal ik voor u een eigenschap van de CPC beschrijven die de bezitters van andere computers wel eens de 'moker' noemen. Naast het normale scrollen, waarbij eenvoudig het beeldschermgeheugen wordt veranderd, bestaat er nog een andere manier waarbij men de monitor ertoe kan brengen om het gehele beeld (inclusief de rand!) in alle vier de windrichtingen te verschuiven (hier is een WOW! wel op zijn plaats, want de constructeurs hebben hieraan veel werk gehad).

Ook hierbij spelen de 6845-registers weer een belangrijke rol. Hier vindt evenwel geen beïnvloeding van het beeldschermgeheugen plaats, maar van het videosignaal voor de monitor. Dit videosignaal bevat niet alleen informatie over het beeld, maar ook nog synchronisatiesignalen die het einde van een regel aangeven. Als dit signaal te laat wordt verstuurd, kan de monitor de beeldinformatie niet meer op de juiste plaats op het scherm zetten. Het resultaat hiervan is dan een verschoven beeld.

Met de 6845 hebt u de mogelijkheid om het tijdstip van deze synchronisatiesignalen met een OUT direct te bepalen. Voor het verschuiven in de horizontale richting moet OUT &BC00,2 worden toegepast. Met een tweede commando (OUT &BD00,x) wordt het synchronisatietijdstip in werking gezet. Normaal heeft x de waarde 46. U kunt voor x elk getal tussen 0 en 63 gebruiken, bij grotere getallen zal de computer zich weer ophangen. Ook hieruit kunt u zich alleen maar verlossen door de computer uit en weer aan te zetten.

Worden de getallen kleiner, dan vindt er een verschuiving naar rechts plaats en bij getallen die groter zijn, wordt er naar links geschoven. De volledige opdrachten voor het verschuiven van een plaats naar links luidt dus:

OUT &BC00,2: OUT &BD00,47

Op dezelfde manier wordt het verschuiven in de verticale richting gedaan. Hierbij luiden dan de opdrachten:

OUT &BC00,7: OUT &BD00,x

X kan hierbij in het gebied van 0 tot 38 liggen, terwijl de normale waarde 30 is. Wordt x kleiner, dan wordt er naar beneden geschoven. Met deze trucs kan men natuurlijk fraaie effecten verkrijgen; men kan bijv. bij een spel de 'strafpunten' door dit veranderen buiten het beeld zetten. Natuurlijk kan men deze twee manieren om te verschuiven ook met elkaar combineren.

Samenvatting: Beeldverschuiven

Horizontale verschuiving:
    OUT &BC00,2: OUT &BD00,x
Verticale verschuiving:
    OUT &BC00,7: OUT &BD00,y

4.7. NOG EEN KEER: CURSORBESTURING

De INKEY$ heeft tegenover het INPUT-commando het nadeel dat de cursor niet meer op het beeldscherm verschijnt. Maar gelukkig kan men de cursor vanuit BASIC in elke willekeurige situatie aan- en uitzetten. Daarvoor moet men gewoon twee bedrijfssysteemroutines oproepen. Hier als voorbeeld een programma om het hiervoor staande duidelijk te maken:

```
10 CALL &BB81: REMcursor inschakelen
20 WHILE INKEY$ = '''':WEND: REM wacht op indrukken van toets
30 CALL &BB84: REMcursor uitzetten
40 .....
50 .....
60 .....
```

In de directmodus werkt dit niet omdat het bedrijfssysteem bij elke afgifte van 'READY' de besturing van de cursor weer zelf overneemt.

Samenvatting: Cursor aan-/uitzetten

Cursor zichtbaar: CALL &BB81
Cursor onzichtbaar: CALL &BB84

5. GRAFIEK

Als u meer dan alleen maar rechte lijnen en punten op het beeldscherm wilt zetten, laat de CPC u aardig in de steek. De BASIC-interpreter kent helaas geen functies voor cirkels, rechthoeken e.d. Gelukkig kan men deze functies met eenvoudige routines maken.

5.1. HET GRAFIEK-STUURTEKEN

In hoofdstuk 4.1. heb ik u bewust een stuurteken onthouden dat met high-resolution te maken heeft. Met CHR$(23) kan men verschillende grafiek-kleurmodi aanzetten. In het normale geval (als u CHR$(23) nog niet hebt gebruikt) worden de punten uit een grafische opdracht eenvoudig op het scherm gezet, onafhankelijk of daar nou wel of niet iets stond. De CPC beheerst echter ook nog een andere techniek. Als u voor het zetten van een punt het commando PRINT CHR$(23) CHR$(1) heeft gegeven, dan worden de nieuwe punten met de oude XOR verbonden. Deze XOR-verbinding heeft interessante effecten, waarvan de uitkomst alleen maar 1 is als de twee ingangsbits verschillend zijn. Met andere woorden: XOR zet alleen dàn een punt als het oude en het nieuwe beeldschermpunt verschillend zijn. Als u een lijn van punt (0,0) tot punt (100,100) had getrokken en u doet dat nu nog een keer in de XOR-modus, dan wordt de lijn uitgewist. Dat komt omdat de punten die de lijn vormen en de punten die u met de tweede opdracht wilt maken, steeds dezelfde zijn. Nemen we voor een zichtbare punt de waarde 1, dan worden steeds 1 met 1 XOR verbonden, waarvan de uitkomst steeds 0 is en dus worden de punten uitgewist. Moet echter de lijn op een plaats komen waar nog geen andere punten staan, dan worden de punten juist zichtbaar omdat de oorspronkelijke waarde en de nieuwe waarde verschillend zijn en de XOR-verbinding de waarde 1 oplevert.

Bekijk eens het volgende programma als voorbeeld:

```
10 MODE 2: PRINT CHR$(23)CHR$(1):REM XOR-MODUS
20 FOR i = 100 TO 200 STEP 2
30 MOVE 10,i:DRAW 100,i: NEXT: REM VLAK TEKENEN
40 FOR j = 1 to 2
50 WHILE INKEY$ = '' '':WEND
60 FOR i = 130 TO 180 STEP 2
70 MOVE 60,i:DRAW 150,i: NEXT i,j: REM TWEEDE VLAK
```

In eerste instantie zult u zich over STEP 2 verwonderd hebben. Deze opdracht is echter belangrijk, omdat u in de loodrechte richting slechts over 200 verschillende punten kunt beschikken, maar de coördinaten daarvoor echter in het gebied van 0 tot 399 liggen. En dus zullen DRAW 100,100 en DRAW 101,101 dezelfde lijn bepalen. Maar precies dàt willen we in de XOR-modus voorkomen, omdat dan de twee rechte lijnen die we trekken elkaar zullen uitwissen.

De rest van het programma kunt u voor uzelf plausibel maken. Er worden in totaal drie vlakken getekend, een groot en twee kleine, waarbij het laatste zijn voorganger uitwist. Bekijk dat zelf eens.

Er zijn nog twee andere mogelijkheden, namelijk die met AND en met OR. Beide werken zoals de vorm met XOR, alleen wordt er een andere verbinding tot stand gebracht. Bij AND wordt er alleen maar een punt gezet als er voordien al een punt stond. Dit kan men toepassen als men met verschillende kleuren wil werken en deze kleuren dan wil omwisselen. U zet dan een punt in de nieuwe kleur op de plaats waar al een punt stond, waarna deze nieuwe kleur alleen maar daar verschijnt waar voordien een andere kleur dan 0 (zwart) stond. Deze modus kan men met PRINT CHR$(23) CHR$(2) aanzetten.

De OR-modus (PRINT/CHR$(23)/CHR$(3))is vergelijkbaar met de transparantmodus voor teksten. Nieuwe punten worden gewoon over de reeds bestaande punten heen gezet. Ook als de nieuwe punt de kleur 0 heeft, wordt de oude niet uitgewist.

Samenvatting: Grafiek modi.

Normaal: CHR$(23) CHR$(0)
XOR : CHR$(23) CHR$(1)
AND : CHR$(23) CHR$(2)
OR : CHR$(23) CHR$(3)

5.2. KAST EN RECHTHOEK

Hierna zal ik een aantal subroutines behandelen die met de bijbehorende grafische commando's nuttig te gebruiken zijn. We beginnen met de eenvoudigste figuren: het vlak en de rechthoek.

Een kast is volgens onze definitie eenvoudig een rechthoek die 'leeg' is. We moeten dus met DRAW vier rechte lijnen tekenen en onze kast is eigenlijk al klaar. Dat kan met het volgende programma worden gedaan:

```
65500 REM KAST: x1,y1,x2,y2,f
65501 MOVE x1,y1: DRAW x1,y1,f
65502 DRAW x2,y2,f: DRAW x2,y1,f
65503 DRAW x1,y1,f: RETURN
```

Dit programma, en ook de volgende programma's zijn zodanig geconstrueerd, dat ze met MERGE achter uw bestaande programma's kunnen worden 'gehangen'. Bovendien worden variabelen gebruikt die in overeenstemming zijn met de gebruikelijke parameter.

Om de plaats van de kast vast te leggen, moeten we alleen de linkerbovenhoek en de rechteronderhoek bepalen. Voor we dan deze routine aanroepen, moeten eerst de waarden van de variabelen worden benoemd. Hierbij geldt voor de coördinaten van de linkerbovenhoek (x1, y1) en voor de coördinaten van de
rechteronderhoek (x1, y1), en trekt daarvandaan de eerste lijn. Bovendien moeten we ook de kleur in de variabele 'f' vastleggen.
De eerste regel van ons onderprogramma zet de grafiekcursor op de plaats met de coördinaten (x1,y1), en trekt daarvandaan de eerste lijn. Met de volgende drie DRAW-opdrachten worden dan de andere drie lijnen getekend. Met RETURN springen we dan weer terug in het hoofdprogramma.

Het oproepen van deze subroutine kan op de volgende manieren plaatsvinden:

```
x1 = 100:y1 = 200:x2 = 200:y2 = 100:f = 1:GOSUB 65500
```

Met het volgende programma willen we een rechthoek tekenen, waarvan ook de vlakken tussen de lijnen zijn opgevuld. Hiervoor zetten we eenvoudig diverse lijnen direct onder elkaar tot de rechthoek de gewenste grootte heeft. Hier is de LISTING:

```
65505 REM RECHTHOEK: x1,y1,x2,y2,f
65506 FOR ii = y1 TO y2 STEP 2*SGN(y2-y1)
65507 MOVE x1,ii: DRAW x2,ii,f
65508 NEXT ii: RETURN
```

Ook hierbij hebben we steeds twee punten nodig; dit gebeurt op dezelfde manier als bij het vorige programma. In de FOR-opdracht vindt u weer STEP 2, die voor de XOR-modus belangrijk is.

Bovendien moet de stapgrootte met -1 worden vermenigvuldigd als y2 kleiner is dan y1. Dit gebeurt met SGN(y2-y1).

In regel 65507 wordt allereerst de grafiekcursor op de linkerkant van de rechthoek gezet, waarvandaan dan een lijn naar de andere kant wordt getrokken. Bij elke lusdoorgang komt de grafiekcursor een plaats lager te staan. Dit gaat net zo lang door tot het vlak geheel gevuld is.

5.3. ALLERLEI MET SINUS EN COSINUS

De twee volgende routines zullen u uit het handboek al een beetje bekend voorkomen. Maar uit het tekenen van cirkels en schijven is nog veel meer te halen.

Blijven we eerst nog even bij de cirkelroutines. In beide gevallen moeten we het middelpunt (x1,y1) en de straal (r1) maar ook de kleur (f) bepalen. Na het aanroepen van deze subroutine doorloopt de FOR...NEXT-lus de hele omtrek van de cirkel (0-360 graden) en berekent daarbij door middel van de sinus- en cosinusfunctie de afstand van dat punt tot het middelpunt.
Er ontstaat geen schijf als PLOTR wordt toegepast, maar telkens als er een lijn, vanuit het middeplunt naar de rand wordt getrokken.

Als men de straal heel groot kiest, kan het gebeuren dat er enkele punten binnen de cirkel niet in de gewenste kleur oplichten, maar donker blijven. Dit kan men voorkomen door in de FOR...NEXT-lus een stapgrootte te kiezen van 0.5 (desnoods kan men nog kleinere stappen nemen).

Hieronder staan dan deze twee routines:

```
65510 REM CIRKEL: x1,y1,r1,f
65511 DEG: r2 = r1: FOR ii = 0 TO 359
65512 xx = COS(ii)*r1:yy = SIN(ii)*r2
65513 MOVE x1,y1: PLOTR xx,y,y,f
65514 NEXT ii: RETURN

65515 REM SCHIJF: x1,y1,r1,f
65516 DEG: r2 = r1: FOR ii = 0 TO 359
65517 xx = COS(ii)*r1: yy = SIN(ii)*r2
65518 MOVE x1,y1: DRAWR xx,yy,f
65519 NEXT ii: RETURN
```

Een typisch begin kan zijn:

```
x1 = 320;y1 = 200:r1 = 100:f = 1:GOSUB 65510
x1 = 100:y1 = 100:r1 = 30:f = 1:GOSUB 65515
```

Weet u dat het, door een paar kleine veranderingen, ook mogelijk is om een ellips te tekenen? Het is u zeker al opgevallen dat er in de regel 65512 twee verschillende variabelen staan. De eerste aanblik zegt u misschien dat dit ook niet nuttig is. Maar als r1 en r2 verschillende waarden hebben, ontstaat er een ellips. Probeert u dat maar eens! R1 is de straal in de X-richting en r2 de straal in de Y-richting.

Om het geheel wat gemakkelijk te laten verlopen, voegen we de volgende regels toe:

```
65520 REM ELLIPS:x1,y1,r1,r2,f
65521 DEG: FOR ii = 0 TO 359
65522 GOTO 65512
```

Door het 'GOTO 65512' in regel 65522 wordt ook het onderprogramma in regel 65512 gebruikt, omdat dit, behalve de eerste twee regels, identiek is. Zo kan men geheugenruimte (en ook tikwerk) uitsparen.

Met een kleine verandering in het 'schijf'programma kunnen we sterren met een willekeurig aantal stralen maken. In beginsel is een schijf ook een soort ster, waarbij vanuit het middelpunt naar elke van de 360 randpunten een lijn wordt getrokken (zie afbeelding 2). Maar omdat de stralen zo dicht op elkaar staan, lijkt het alsof het een vlak is. We moeten derhalve het aantal stralen reduceren. Dit kunnen we bereiken met een STEP-commando, waarbij we bijv. elke tiende straal tekenen. Om de stralen gelijkmatig over de cirkel te verdelen, wordt het getal 360 door dit aantal gedeeld en de uitkomst als STEP-grootte ingevoerd.

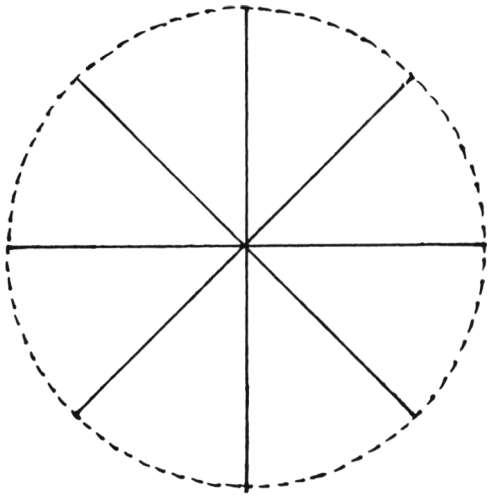
Ook in regel 65527 vindt u weer een GOTO. Opdat ook werkelijk alle stralen verschijnen, moet de FOR...NEXT-lus tot 360 worden verlengd.

Ook voor de ster moeten het middelpunt, de straal en de kleur in de variabelen worden uitgedrukt. Bovendien bestuurt de variabele EC het gewenste aantal stralen. Ook van dit programma krijgt u een LISTING:

```
65525 REM STER: x1,y1,r1,ec,f
65526 DEG: r2 = r1: FOR ii = 0 TO 360 STEP 360/ec
65527 GOTO 65517
```

Waarbij we kunnen beginnen met:

```
x1 = 100:y1 = 100:r1 = 30:ec = 12:f = 1:GOSUB 65525
```

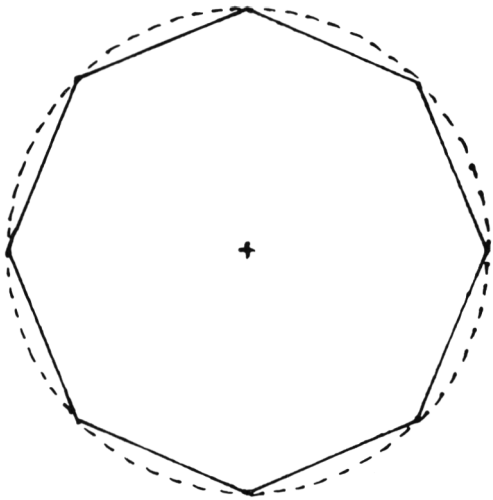


afb. 2 ster

De laatste subroutine tekent een willekeurige veelhoek (ook wel polygoon genoemd). Voor het principe van de functie moeten we nog een keer naar de ster kijken. Verbind hierin alle buitenste punten met rechte lijnen en u krijgt dan een veelhoek. Hierbij kunnen we onze routine gebruiken. Zoals ook bij de sterren worden er in meer of minder grote afstanden punten op de cirkelrand berekend en worden deze punten door middel van rechte lijnen met elkaar verbonden. Bovendien moet de grafiek-cursor laten we zeggen 'op het spoor gezet worden', omdat er anders hoogst ondecoratieve lijnen getrokken worden. Ook hier moet de lus weer verlengd worden, deze keer tot 370.

```
65530 REM POLYGOON:x1,y1,r1,ec,f
65531 DEG: MOVE x1,+ r1,y: FOR ii = OTO 370 STEP 360/ec
65532 xx = COS(ii)*r1: yy = SIN(ii)*r1
65533 DRAW x1 + xx,y1 + yy,f: NEXT ii: RETURN
```

Als u dat zou willen, kunt u ook ei-achtige sterren en polygonen maken, waarbij u alleen maar een tweede straal hoeft in te voeren. U kunt ook halve cirkels of andere delen van de cirkel maken als u de FOR...NEXT-lus b.v. van 180-360 laat lopen. Aan uw fantasie zijn geen beperkingen opgelegd.



afb. 3 polygoon

5.4. WAAROM PIXELTESTS?

Zo op het eerste gezicht lijken de commando's TEST en TESTR relatief zinloos. Als we alleen maar willen weten welke kleur een punt heeft, hoeven we slechts naar het beeldscherm te kijken. Maar er zijn ook nog toepassingen die zonder TEST niet denkbaar zijn.

In dit verband denk ik aan de Commodore-sprites of aan de Shapes van de Applecomputer. Voor de beginners onder u die nog niet zo goed thuis zijn in de wereld van de computer, zal ik dit een beetje toelichten. Het gaat hierbij om figuren die de programmeur zelf kan definiëren en door hem met een enkele opdracht op het beeldscherm kunnen verschijnen. Iets dergelijks kunt u ook met de CPC in BASIC programmeren.

Ons programma moet hiervoor twee functies hebben. Met de ene moeten we een willekeurig deel van het beeldscherm op een andere plaats kopiëren. Het principe van deze subroutine is bepaald eenvoudig. Zoals bij het tekenen van de kast en van de rechthoek uit 5.1. worden er twee punten aangegeven die de nieuwe plaats bepalen. Dan worden er door middel van twee FOR...NEXT-lussen alle pixels uit deze rechthoek onderzocht en wordt de uitkomst van deze TEST als kleurnummer voor het nieuw te zetten punt gebruikt. Als de TEST-uitkomst bijvoorbeeld een 0 is, wordt er een punt met de kleur 0 (zwart) gezet, waarbij de uitkomst een donkere pixel zal zijn. Op deze manier wordt het hele gebied pixel voor pixel gekopiëerd.

De tweede routine moet een beeld op het scherm zetten dat er nog niet eerder was. Dit wordt dus niet gekopiëerd, maar nieuw gemaakt. Daarvoor moet men natuurlijk op de een of andere manier de gegevens vasthouden. In BASIC-programma's gaat dat het beste met DATA-regels. Voor elke puntregel die door ons onderprogramma op het beeldscherm moet worden gezet, moeten we een string maken waarin de kleuren van de pixel moeten staan. De lengte van de string geeft aan hoeveel punten er op een rij komen. Voor elk punt staat er een teken in de string. Daarbij moet het teken een hexadecimaal getal zijn, waarmee het getal met de kleur wordt aangegeven. Ook moet de routine nog weten wanneer de figuur klaar is. Daarvoor nemen we eenvoudig een string met de lengte 0. Komt de subroutine deze lege string tegen, dan is bekend dat de figuur is voltooid.
Tenslotte hebben we ook nog de positie nodig waar onze figuur moet worden neergezet. De hiervoor noodzakelijke coördinaten definiëren we als variabelen.

```
999 REMGEBIED KOPIËREN;x1,y1,x2,y2,xs,ys
1000 FOR ii = y1 TO y2 STEP -2: FOR jj = x1 TO x2
1010 PLOT xs + jj-x1,ys + ii-y1, TEST (jj, ii)
1020 NEXT jj,ii: RETURN

1049 REMGEBIED OP HET BEELDSCHERM PLOTTEN: xs, ys
1050 zz = 0
1060 READ aa$: 11 = LEN(aa$): IF 11 = 0 THEN RETUN
1070 FOR ii = 0 TO 11-1: aa = VAL("&" + MID$(aa$,ii,1))
1080 PLOT xs + ii*m,ys-zz*2,aa: NEXT ii: zz = zz + 1: GOTO 1060
```

In de eerste subroutine hebben we x1, x2, y1 en y2 nodig als coördinaten van het gebied dat we willen kopiëren. Xs en ys geven de coördinaten van de linkerbovenhoek weer van de plaats waar deze kopie komt te staan.

Bij het tweede routine hebben we niet zoveel variabelen nodig. Met xs en ys geven we de coördinaten van de linkerbovenhoek van ons doel op het beeldscherm weer. In het PLOT-commando moet u even goed op de verdubbeling van zz letten, want deze variabele bepaalt het nummer van de lijn die we doorlopen.
Doen we deze verdubbeling niet, dan worden onze lijnen twee keer getekend (de CPC verdubbelt de Y-coördinaten). Hiermee moet men ook bij het bepalen van de hoogte rekening houden. Dus: voor elke puntenregel die we willen PLOTten, moeten we twee punten in de Y-richting verder tellen. Iets dergelijks geldt ook voor de X-richting. Hier wordt met de variabele 'm' vermenigvuldigd, waar, afhankelijk van de modus, een ander getal komt te staan. Als de modus 0 is, wordt de waarde van m gelijk aan 4, omdat dan 4 X-coördinaten hetzelfde punt aangeven. Bij modus 1 is dat 2 en 1 bij modus 2.

Om het gebruik van deze routine wat duidelijker te maken, hebben we hier nog een voorbeeldprogramma:

```
100 MODE 2
110 xs = 320:ys = 200:m = 1:GOSUB 1050:END
120 DATA"10000100001100010000100000011110"
130 DATA"10000100010010010000100000100001"
140 DATA"10000100100001010000100000100001"
150 DATA"10000100100001010000100000100001"
160 DATA"11111100111111010000100000100001"
170 DATA"11000100110001011000011000011001"
180 DATA"11000100110001011000011000011001"
190 DATA"11000100110001011000011000011001"
200 DATA"11000100110001011111101111110011110"
210 DATA""
```

Deze regels moeten aan de vorige listing worden toegevoegd.

In de DATA-regels staat een puntraster voor het woord HALLO. Zoals u gemakkelijk kunt zien, stelt het getal 1 een zichtbare pixel voor en het getal 0 een onzichtbare. Als u in een ander MODUS werkt, kunt u tegelijkertijd met meerdere kleuren werken, bijv. 'f' voor de kleurstift 15.

5.5. COÖRDINATENSYSTEMEN

Dit gedeelte van het boek is in het bijzonder voor de wiskundigen en scholieren onder u interesssant. De eersten moeten het mij niet kwalijk nemen als ik zulke eenvoudige zaken als coördinatensystemen op een volledig onvakkundige manier behandel, de tweeden hebben daarentegen misschien nog veel problemen met dit nog onbekende thema.

Elke keer als u een functie grafisch wilt weergeven, moeten de functiewaarden worden omgerekend in beeldschermcoördinaten. Daarbij kan het commando ORIGIN een deel van deze arbeid overnemen. Laten we dit aan de hand van het voorbeeld 'sinuskromme' eens doornemen.

Zoals u weet, liggen de functiewaarden van een sinus tussen -1 en + 1. Daarom willen we de X-as precies in het midden van het beeldscherm hebben.

De Y-as wordt bij de trigonometrische functies meestal aan de linkerkant getekend. Daarmee krijgt het nulpunt van het coördinatensysteem de beeldschermcoördinaten (0,199). Geven we in ORIGIN 0,199, dan hebben alle volgende grafiekcommando's alleen nog betrekking op dit nieuwe nulpunt. PLOT a,-1 zet dus voor elke toegestane waarde van a een punt direct onder de X-as.

Als u eenmaal wat met het commando ORIGIN hebt geëxperimenteerd en dan de daarbij behorende waarde bent vergeten, kunt u dat met de volgende regels weer op het scherm krijgen:

```
PRINT UNT(PEEK(&B328) + 256*PEEK(&B329))
```

Daarmee krijgt u dan de waarde voor de X-richting; voor de Y-as moeten dan de adressen in &B32A en &B32B veranderd worden.

Als u nu gewoon de waarden van de sinusfunctie in de grafiekopdrachten zet (bijv. met PLOT x,SIN(x)), dan zou de sinuskromme slecht één pixel hoog worden. Daarom ziet de goede opdracht er als volgt uit:

```
PLOT x,SIN(x)*199
```

Vanzelfsprekend kunt u ook de coördinaten van de X-as met een dergelijke factor vermenigvuldigen, als het bereik niet met de grafiekcoördinaten overeenstemt.

Hieronder staat een subroutine die aan de hand van uw eigen opgaven van het bereik van de X- en de Y-as een assenstelsel op het beeldscherm tekent.

```
1000 INPUT''X-MINIMUM'';XA
1010 INPUT''X-MAXIMUM'';XE
1020 INPUT''Y-MINIMUM'';YA
1030 INPUT''Y-MAXIMUM'';YE
1040 CLG
1050 MX = (XE-XA)/639:X = -XA/MX
1060 MY = (YE-YA)/399:Y = -YA/MY
1070 ORIGIN X,Y
1080 MOVE XA/MX,0:DRAW XE/MX,0:REM Y-AS
1090 MOVE 0, YA/MY:DRAW 0,YE/MY:REM X-AS
1100 RETURN
```

De eerste vijf regels zijn duidelijk en daarom ga ik daar niet verder op in.

In de twee daaropvolgende regels worden de maten voor de X- en Y-as en de coördinaten van het nulpunt berekend (X,Y).
De maten worden berekend door het verschil van het minimum en het maximum te delen door het mogelijke aantal punten. Dit verschil geeft aan hoeveel punten er gewenst worden; deze worden door de deling op het mogelijke aantal punten geprojecteerd.

Elke keer als uit een functiewaarde of uit een X-waarde de beeldschermcoördinaten moeten worden berekend, moet men door de bijbehorende maat delen. Dit gebeurt ook met de nieuwe coördinaten van het nulpunt.

Regel 1080 tekent dan de Y-as en regel 1090 is bedoeld voor het tekenen van de X-as. Ook hierbij vindt u de maat weer terug.

Dit programma kan natuurlijk ook nog verfijnd worden als u door middel van TAG en PRINT de diverse betekenissen erbij schrijft. Ook de deelstreepjes van de X- en de Y-as kan men eraan toe PLOTten.

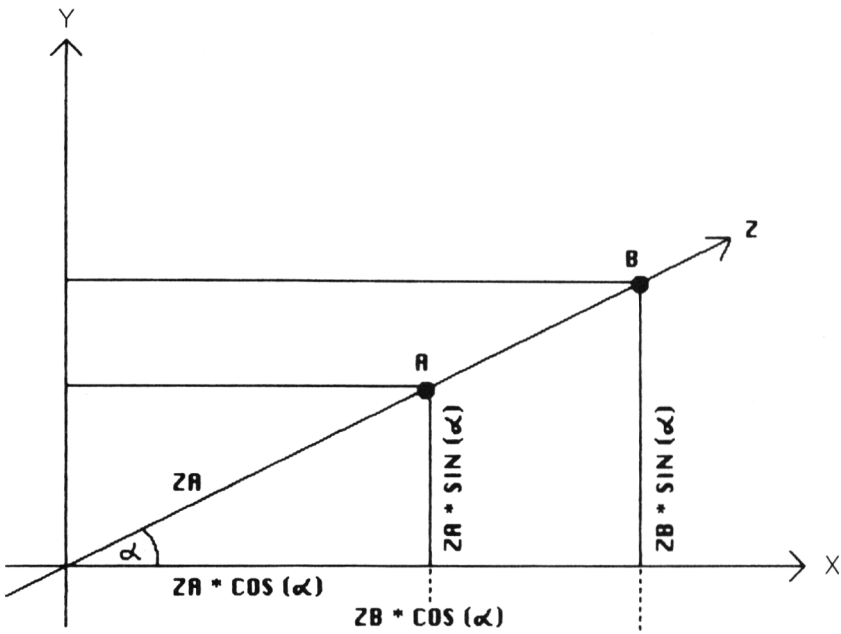
5.6. 3D-GRAFIEKEN

De grafische voorstellingen van driedimensionale functies zijn ongetwijfeld de interessantste grafieken, maar helaas ook de lastigste. Maar ondanks dat zal ik u toch, met een klein programma, in deze techniek inwijden.

Bekijken we allereerst het coördinatensysteem. Anders dan bij de normale functies hebben we hierbij 3 assen: X, Y, en Z. De Z-as moet noodgedwongen schuin worden getekend (zie de afbeelding). De hoek (en daarmee de perspectiviteit) kunnen we zelf bepalen.

Omdat we met het commando PLOT geen drie coördinaten kunnen geven, moeten deze worden omgerekend naar 2 dimensies. Dat noemt men een 'projectie'. Wij houden ons hier bezig met de eenvoudigste vorm van projecteren; de parallelprojectie, waarbij er geen 'vluchtpunt' bestaat. Als in werkelijkheid evenwijdig lopende lijnen worden geprojecteerd, zullen ook hun projecties evenwijdig lopen en geen snijpunt bezitten.

Zoals u in afbeelding 4 kunt zien, moet een punt, afhankelijk van de Z-as, meer of minder ver naar rechts worden geschoven. Op de Z-as zijn twee punten met de coördinaten (0,0,za) en (0,0,zb) getekend. Zoals u ziet, kan men de projectiecoördinaten door optelling van za*cos(a) en zb*sin(a) berekenen. De projectieformules luiden dus:



Afb. 4 Coördinatensysteem voor 3D-grafieken

x = 0 + za*cos(a) y = 0 + za*sin(a)

of in de algemene vorm:

x = xa + za*cos(a) y = ya + za*sin(a)

Omdat deze 2D-coördinaten niet altijd in overeenstemming zijn met de coördinaten van het beeldscherm, moeten we nog de strekkingsfactoren SX en SY invoeren (zie programmalisting, regel 100). Ook de oorsprong van het coördinatenstelsel wordt met ORIGIN 320,200 verplaatst, zodat het beeld in het midden van ons beeldscherm verschijnt.

Omdat er van de drie coördinaten slechts een wordt berekend en de twee andere op een andere manier worden bepaald, hebben we in ons programma twee geneste lussen nodig. Het is ook aan te raden, om redenen die ik verderop nog zal verklaren, de functie van achteren naar voren te PLOTten.
In het onderstaande programma worden alle waarden die bij het experimenteren vaak moeten worden veranderd, via een INPUT-commando ingelezen. Bovendien werkt het programma in MODE 2, omdat we niet zoveel mogelijk kleuren willen hebben, maar een zo goed mogelijke oplossing. Tik het programma in en start het met de volgende waarden: h = 45:sx = 0.5:sy = 0.5:xs = 2:zs = 20

```
10 INPUT"HOEK";h:DEG:z1 = COS(h):z2 = SIN(h)
20 INPUT"UITREKKING-X";sx
30 INPUT"UITREKKING-Y";sy
40 INPUT"STAP GROOTTE-X";xs
50 INPUT"STAP GROOTTE-Z";zs
60 MODE 2:ORIGIN 320,200
70 FOR z = 180 TO -180 STEP -zs
80 FOR x = -180 TO 180 STEP xs
90 y = SIN(x)*cos(z)*100: REM FUNKTIETERM
100 bx = sx*(x + z*z1):by = sy*(y + z*z2)
110 PLOT bx,by,1
120 NEXT x,z
```

Misschien is het u bij het afdraaien van dit programma opgevallen dat er punten zijn die eigenlijk aan de achterzijde liggen en daarom niet gezien kunnen worden. Dit probleem is op te lossen door alle punten weg te wissen die onder het laatst gezette punt liggen. Verander daarom regel 100 in:

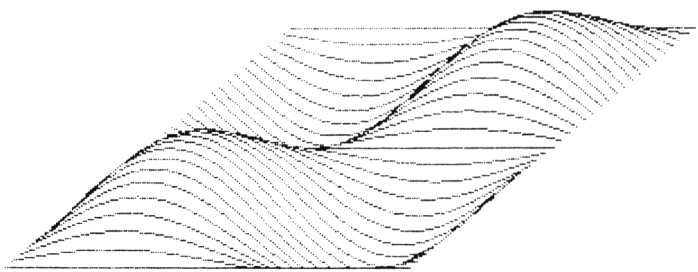
```
110 PLOT bx,by.1: MOVE bx,by-2: DRAW bx,-200,0
```

Aan de hand van dit voorbeeld wil ik het een en ander uitleggen. Gaan we ervan uit dat onze functie een volledig plat vlak is, dan verdwijnt dit vlak schuin van onder naar boven in de ruimte. Het voorste punt ligt dus het laagst en kan daarom geen ander punt bedekken. Als we nu aannemen dat we de Y-coördinaten van het voorste punt met de waarde 1 vergroten, dan wordt de daarachter liggende regel precies bedekt. Vergroten we de Y-coördinaten nu steeds meer, dan worden er steeds meer regels afgedekt. Hieruit volgt dat alle punten die onder de eerste liggen, moeten worden uitgewist.

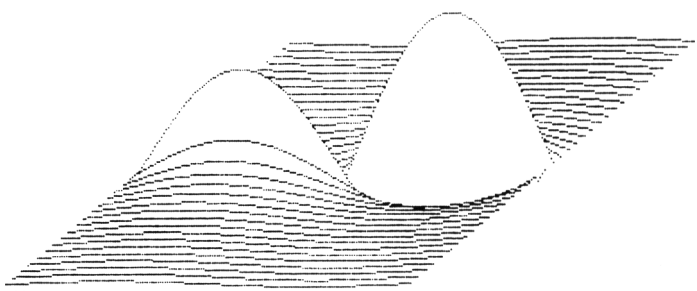
Dit geldt ook als de onderste regel geen rechte maar een kromme is. Het is het beste als u dit uitwissen bij het ontstaan van de grafiek nauwkeurig bekijkt, dan zult u het direct begrijpen.

Nog een paar toelichtingen van de diverse programmaregels. In regel 10 worden de cosinus- en de sinuswaarden van de hoek h berekend. Omdat deze waarden niet meer veranderen, kunnen we ze als constanten in het geheugen opslaan. Dat bespaart ons veel rekenwerk. De waarden in de FOR...NEXT-lussen in de regels 70 en 80 kunt u desgewenst veranderen om daarmee grote of kleine fragmenten van de functie te laten zien. In regel 90 is u waarschijnlijk de uitdrukking '*100' opgevallen. Omdat de functiewaarden altijd kleiner zijn dan 1, moeten we deze wel uitrekken.

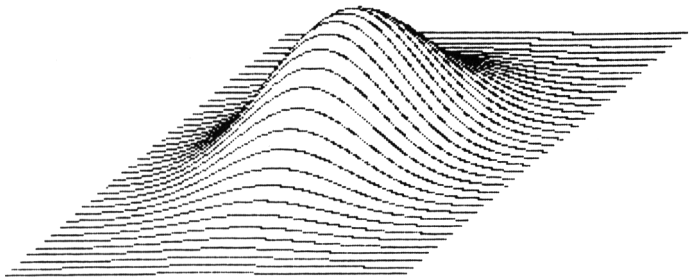
Probeert u het nu eens met verschillende uitrekfactoren en verschillende stapgrootten. Ook kunt u nu diverse functies proberen. Veel plezier ermee!



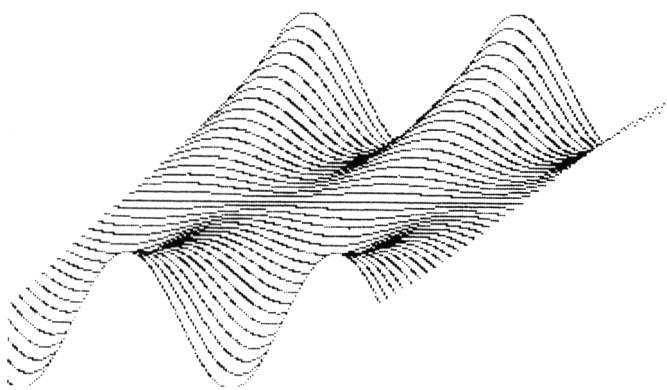
Afb. 5. y = sin(x)*sin(z)*140



Afb. 6. y = sin(x)*2/(z/8 + 0.5)*80



Afb. 7. y = exp(-(x*x + z*z)/2/sqr(2*P1)*300



Afb. 8. y = sin(x/30)*(6exp(z/90)-1/exp(z/90)*10

6. TOEPASSINGEN VAN GRAFIEKEN

Met de hulpmiddelen van het laatste hoofdstuk kan men al aardige grafieken opbouwen. Echte gebruikswaarde hebben ze echter nog niet. Dat wil ik dan nu veranderen. Als u tot de gebruikers behoort die hun CPC ook in het bedrijf willen inzetten, zal hoofdstuk 6.1. u bijzonder interesseren. Voor de hobbyisten en de kunstenaars is in het bijzonder hoofdstuk 6.2. bedoeld.

6.1. DIVERSE DIAGRAMMEN

Er zullen beslist veel computerbezitters zijn die op de avond van de verkiezingen de wens zullen uitspreken om uit de veelheid van getallen een grafische voorstelling te kunnen maken. Of wilt u misschien liever de verkoopcijfers of dergelijke waarden in een staafdiagram onderbrengen? Dat alles is met de CPC gemakkelijk te realiseren. In feite is een staaf niets anders dan een rechthoek, die we al in hoofdstuk 5.1. hebben geprogrammeerd. We moeten dus alleen nog maar de getallen in coördinaten omzetten. De omrekening van de werkelijke waarden in beeldschermcoördinaten heb ik in hoofdstuk 5.5. al uitgelegd en ik zal hier dezelfde methode toepassen.

Voor het X-bereik hoeven we niet meer een minimum en een maximum te bepalen, maar alleen nog het aantal "staven". Dit getal wordt dan met 10 vermenigvuldigd om de punten voor de staafbreedte te reserveren (een staaf moet immers breder zijn dan 1 pixel; zie regel 60040).

In een array W(0,1,2,...XE) moeten de enkele waarden staan. Het maximum wordt in de eerste FOR...NEXT-lus berekend. Omdat de nummering van een array met 0 begint en wij in het algemeen met 1 beginnen te tellen, moet van het getal xe nog 1 worden afgetrokken.

Bij de maatformules vallen u zeker de kleine puntconstanten op en de daardoor verkleinde vlakken voor ons diagram. Deze zijn nodig om nog wat ruimte over te houden voor een eventueel onderschrift.

Omdat we ervan uitgaan dat verkoopcijfers e.d. nooit negatief worden, kunnen we de minima van xa en ya verwaarlozen.

Regel 60040 wist het beeldscherm uit en zet dan de oorsprong van het coördinatenstelsel op het punt (50,30). Daarmee blijft er aan de onder- en de linkerkant van het staafdiagraam nog ruimte over voor een commentaar. In regel 60050 worden de grenzen van ons diagram getekend; dat werkt op dezelfde manier als bij het assenstelsel.

In de regels 60060 en 60070 vindt u het rechthoekprogramma in een iets gewijzigde vorm. Omdat we loodrechte staven willen tekenen, is het voordeliger en verloopt alles ook veel sneller als we ook de lijnen voor deze rechthoek loodrecht DRAWen.

Bovendien heb ik ook nog een aantal variabelen door rekenkundige uitdrukkingen vervangen. Daarbij moet ik nog zeggen dat de constante 10 het aantal maateenheden aangeeft die voor een staaf plus de tussenruimte ter beschikking staat. De werkelijke staafbreedte wordt door ' + 8' in regel 60070 (alweer in maateenheden en niet in pixels) aangegeven. Als u een grotere of een kleinere tussenruimte wilt hebben, hoeft u alleen maar deze waarden te veranderen.

```
60000 REM STAAFDIAGRAM: xe, w(0.....xe-1),f
60010 xe = xe-1:ye = 0
60020 FOR ii = 0TO xe:ye = MAX(ye,w(ii)):NEXT
60030 mx = (xe + 1)*10/584:my = ze/364
60040 CLG:ORIGIN 50,30
60050 MOVE -5,0:DRAW -5,ye/my + 5:MOVE -5,0:DRAW (xe + 1)*10/mx + 5,0
60060 FOR ii = 0TO xe:FOR jj = ii*10/mx TO (ii*10 + 8)/mx
60070 MOVE jj,0:DRAW jj,w(ii)/my,f:NEXT jj,ii
60080 RETURN
```

Het tweede diagramtype lijkt veel gecompliceerder, maar ook hierbij kunnen we weer gebruik maken van oudere onderprogramma's en deze een beetje aanpassen. Bedoeld worden hiermee de 'taartdiagrammen', waarbij aan de bijbehorende waarden een meer of minder groot deel van het cirkelsegment wordt toegewezen.

De opbouw van deze diagrammen is heel eenvoudig. Achter elkaar worden voor elk getal delen van de cirkel getekend, waarbij alle delen samen een volle schijf vormen. Om deze delen goed van elkaar te kunnen onderscheiden, worden verschillende kleuren gebruikt.

U hebt zich misschien afgevraagd hoe men een cirkelsegment moet programmeren. Tot nu toe hebben we alleen maar met hele cirkels en schijven gewerkt. Voor de verklaring hiervan kunt u nog eens hoofdstuk 5.3. opzoeken. In de LISTING van regel 65515 ziet u staan 'FOR i = 0 TO 359'. Zoals ik daar al heb opgemerkt, wordt daarmee de hele cirkel van 0 tot 359 graden beschreven. Als u echter intikt 'FOR i = 180 to 270', krijgt u slechts een kwart cirkel. U ziet dus dat we het begin en het einde van de cirkelboog vrij kunnen kiezen.

Er is echter nog een ander probleem. Opmerkzame lezers hebben vast en zeker vastgesteld dat onze oorspronkelijke routine de cirkel mathematisch correct tegen de wijzers van de klok in heeft getekend. Onze leesgewoonte is echter precies andersom. Dit kan heel gemakkelijk worden opgelost als we de SIN- en Cos-functies verwisselen voor de coördinaten. Daarmee begint de omloop bij '12 uur' (dus in de bovenste positie) en draait dan met de klok mee.

De LISTING luidt:

```
60100 REM TAARTEN DIAGRAM :x1:y1:r:xe:w(0...xe-1)
60110 wb = 0:DEG:FOR ii = 0TO xe-1
60120 wa = wb:wb = wb + w(ii)*3.6:s(ii) = (wb-wa)/2
60130 FOR jj = wa TO wb
60140 xx = SIN(jj)*r:yy = COS(ii)*r
60150 MOVE x1,y1:DRAWR xx,yy,f(ii):PLOTR 0,0,1
60160 NEXT jj,ii:RETURN
```

Ook dit onderprogramma verwacht een aantal variabelen (zie de lijst in regel 60100). X1 en x2 bepalen het middelpunt van de cirkel en r de straal. Ook de functie van xe is dezelfde gebleven. De afzonderlijke waarden die de grootte van de cirkelsegmenten bepalen, worden als procenten in de array w(0...xe-1) opgeslagen. Als een veld 20% van de cirkel moet bedekken, dan wordt de variabele 20. Bovendien kan voor elk deel bepaald worden welke kleur het moet hebben (f(ii)).

Ook dit programma bestaat uit twee geneste lussen. De buitenste zorgt ervoor dat alle waarden worden gekozen en de binnenste dat alle hoeken worden doorlopen.

In regel 60120 worden de begin- en de eindwaarden van het cirkelsegment berekend. Bovendien willen we onszelf een beetje verwennen. In de array s(*...xe-1) (dat voor het gebruik van dit programma geDIMensioneerd moet worden) wordt steeds de hoek opgeslagen die het segment halveert. Als u dan de afzonderlijke delen wilt beschrijven, kunt u de grafiekcursor in deze hoek zetten en met TAG een tekst op de gewenste plaats laten ontstaan. Ook hierbij moeten de coördinaten met SIN en COS worden berekend, terwijl we ook de straal groter moeten maken om te voorkomen dat de tekst in de cirkel komt te staan.

In deze schijfroutine is alleen het PLOTR-commando nieuw. Het tekent een rand om het diagram heen voor het geval de segmentkleur zwart is. Als we dat niet doen, lijkt het erop dat we een ster tekenen.

De rest van het programma is duidelijk en hoef ik daarom niet meer te verduidelijken.

U moet er nog wel op toezien dat de som van de waarden in het array gelijk is aan honderd, daar het anders kan gebeuren dat de cirkel niet geheel wordt getekend en daarmee op een ster gaat lijken of dat er een deel van de cirkel wordt overschreven.

6.2. HET PROGRAMMA VOOR DE 'KUNSTENAARS'

Het volgende programma is bedoeld voor degenen die veel met grafieken experimenteren en de vooraf pasklaar gemaakte beelden dan in hun programma's willen opnemen. Het is veel te vermoeiend om hele beelden met PLOT en DRAW te vervaardigen; eenvoudiger is het om met de cursortoetsen rondom het beeldscherm te gaan en dan naar wens een punt neer te zetten of uit te wissen. Dat kan met het onderstaande programma:

```
10 MEMORY 16383:CALL &BC06,&40:MODE2:x = 320:y = 200:m = 1
20 a$ = '''':WHILE a$ = '''':a$ = INKEY$:WEND:PLOT x,y,f
30 IF a$ = CHR$(240) AND y<398 THEN y = y + 2
40 IF a$ = CHR$(243) AND x<639 THEN x = x + 1
50 IF a$ = CHR$(241) AND y>1 THEN y = y-2
60 IF a$ = CHR$(242) AND x>1 THEN x = x + 1
70 IF a$ = CHR$(224) THEN f = 1:BORDER 24:m = 0
80 IF a$ = CHR$(13) THEN f = 0:BORDER 24,0:m = 0
90 IF a$ = '' '' THEN m = 1:BORDER 0
100 IF a$ = ''s'' THEN CALL &BC06,&C0:MODE 2:INPUT''filename'';f$:
    SAVE f$,b,16384,16384:CALL &BC06,&40:GOTO 20
110 IF a$ = ''1'' THEN CALL &BC06,&C0:MODE 2:INPUT''filename'';f$: LOAD f$,
    16384:CALL &BC06,&40:GOTO 20
120 IF a$ = ''c'' THEN CLG:GOTO 20
130 IF a$ = ''x'' THEN CALL&BC06,&&CO:END
140 IF m = 1 THEN f = TEST(x,y)
150 PLOT x,y,1:GOTO 20
```

Voor de grafiek wordt het tweede beeldschermgeheugen vanaf 16384 gebruikt, opdat eventuele teksten onze moeizame arbeid niet zullen verstoren.

Bij het tekenen bestaan er drie modi, die we aan de verschillende randkleuren kunnen herkennen. De eerste modus wil ik de 'cursor-modus' noemen, omdat we hiermee de cursor over het beeldscherm kunnen bewegen zonder dat de daaronder liggende punten worden uitgewist. Hierbij is de rand donker. Is de rand helder, dan wordt een punt onder de grafiekcursor gezet en als de rand knippert, dan wordt er uitgewist. Bij elk van deze modi behoort een toets waarmee ze worden opgeroepen. Voor de cursor-modus is dat de spatiebalk, punten kunnen worden gezet door het indrukken van COPY, en ENTER brengt het programma in de wis-modus.

Maar dat is nog niet alles. Met de 'S'-toets kan men het beeld op een cassette opslaan. Daartoe wordt overgeschakeld naar het oorspronkelijke beeldschermgeheugen, waar dan de naam van het beeld kan worden ingetoetst. Dan wordt er geSAVEd (zie regel 100). Op een vergelijkbare manier werkt het laden van een beeld dat inmiddels al klaar is met 'L' (regel 110). In beide regels bevinden zich MODE-commando's, waarmee het beeldscherm wordt uitgewist en het scrollen bij het overschakelen wordt vermeden.

De 'C'-toets wist het beeldscherm uit. Tenslotte is er nog de toets 'X' waarmee men het programma kan beëindigen, zonder dat het beeld daardoor verloren gaat. Als u dan nog iets wilt veranderen en het oude beeld wilt terughalen, tikt u gewoon 'CALL &BC06,&40: GOTO 20' in.

Nu nog een aantal verklaringen van de afzonderlijke programmaregels. Regel 10 initieert het beeldscherm en zet de startcoördinaten. In de daaropvolgende regel wordt een teken van het toetsenbord gehaald en de nieuwe pixel (uitkomst van de vorige lus) gezet. Verder wordt deze toets in een aantal verschillende IF...THEN-constructies benut.

De variabele 'f' bepaalt de kleur voor het PLOT-commando uit regel 20. Is de cursormodus ingeschakeld (m = 1), dan geeft f de kleur weer van de oorspronkelijke pixel. Op deze manier blijft het beeld onveranderd behouden. Het PLOT-commando staat achter de INKEY$ om de grafiek-cursor zo lang mogelijk op het beeldscherm te houden. De cursor zelf wordt door het PLOT-commando in regel 150 opgeroepen.

Natuurlijk is dit kleine programma geen super-software; dat zou ook buiten het doel van dit boek liggen. Vaak is het een begin voor uw eigen ontwikkeling en een klein hulpmiddel voor 'er even tussendoor'. Overigens kunt u de beelden die met dit programma zijn opgeslagen via LOAD''!naam'',49152 weer terughalen in het normale beeldschermgeheugen.

7. PROGRAMMEREN MET INTERRUPTS

Het programmeren met interrupts is een van de voortreffelijkste eigenschappen van het CPC-BASIC. Helaas wordt hierover in het handboek weinig vermeld, terwijl ook de bijbehorende commando's nauwelijks worden verklaard. Dat moet hier dan maar gedaan worden.

7.1 HOE WERKT EEN BASIC-INTERRUPT?

In beginsel is er geen verschil tussen de machinetaal- en BASIC-interrupts. In beide gevallen levert een bouwsteen (meestal de timer) een interruptsignaal, waarmee het lopende programma wordt onderbroken nadat het commando dat zojuist wordt uitgevoerd is beëindigd. Daarna wordt er een sprong naar de betreffende subroutine gemaakt. Tot zover gaat het nog wel. Bij de Z-80 gaat dit allemaal op hardware-basis, hetgeen betekent dat het niet via een programma maar door bepaalde schakelingen wordt uitgevoerd. In BASIC daarentegen zijn er uitgekiende machineprogramma's waarmee een correcte uitvoering hiervan kan worden gemaakt. De BASIC-interrupts kunnen slechts een van de vier timers (0-3) gebruiken en niet bijv. een interface.

Als er een dergelijk commando verschijnt, onderzoekt het BASIC allereerst of er nòg een opdracht moet worden uitgevoerd. Het is dus niet mogelijk om INPUT-opdrachten of dergelijke tussentijds te onderbreken.

Het kan ook zijn dat de interruptroutine door een DI-commando (DI = disable interrupt) wordt tegengehouden. In de beide laatste gevallen zal de BASIC-interpreter de interruptroutine tijdelijk vasthouden en na het vrijgeven alsnog uitvoeren.

Ten slotte wordt er nog bepaald welke routine er door de timer moet worden uitgevoerd en wordt dit als een normale subroutine gestart.

Dit zijn nog niet alle opgaven die de CPC in verband met de interrupts moet waarnemen. Als u met INK wisselende kleuren hebt gedefinieerd, worden deze kleurwisselingen door een speciale timer uitgevoerd. U kunt deze timer met SPEED INK instellen. Steeds als er van kleur moet worden gewisseld, verandert het bedrijfssysteem een byte in de chip die voor de produktie van deze kleuren word gebruikt.

Bovendien vindt er een onderzoek van het toetsenbord plaats. Ook dit gebeurt door de inmiddels bekende manier door een interrupt die er elke 1/50ste seconde voor zorgt, dat de interface de code van de zojuist ingedrukte toets naar de processor stuurt.
In uw CPC is er zo te zien heel wat aan de hand!

7.2. De interruptcommando's

Na zoveel theorie moeten we nu de afzonderlijke interruptcommando's nauwkeuriger onder de loep nemen. Als eerste willen we de commando's noemen waarmee de interrupt kan worden ingeschakeld: AFTER en EVERY.

Als u de Engelse taal niet zo best beheerst: AFTER betekent 'na' en EVERY betekent 'elke'. Met deze twee commando's krijgt men een vergelijkbaar resultaat, ook de syntax is gelijk:

AFTER x,y,GOSUB z
EVERY x,y,GOSUB z

Zowel AFTER als EVERY stellen een uurwerk in op een bepaalde tijd, zoals bij een wekker. Anders dan onze wekker moet hier een tijd worden ingesteld na hoeveel 50ste seconden de interrupt moet worden ingeschakeld. Deze waarde komt in de eerste parameter (x) te staan. Met AFTER 200... wordt dus na 200/50 = 4 seconden de interrupt op gang gebracht.

Hiermee hebben we dan ook tegelijk het enige verschil dat er tussen AFTER en EVERY bestaat. Het commando AFTER wordt slechts een enkele interrupt opgeroepen, terwijl bij het commando EVERY steeds, als de tijd verstreken is, een nieuwe interrupt gaat werken. Zo kan men bijvoorbeeld elke 10 seconden het toetsenbord onderzoeken terwijl de berekeningen gewoon doorgaan.

De tweede parameter (y) geeft aan welke van de vier timers gebruikt wordt. Hoe hoger het nummer, des te meer betekenis heeft de interrupt. Timer 3 kan timer 2 onderbreken, maar niet omgekeerd. De rest van de opdracht is overigens vanzelfsprekend, omdat daar alleen wordt aangegeven naar welke regel er moet worden gesprongen.

De twee opdrachten DI en EI zijn snel uitgeleged. DI is een afkorting voor 'disable interrupt', waarmee wordt bedoeld dat er geen interrupts meer worden uitgevoerd. Ook de timer van een hogere prioriteit kan nu niets meer beginnen. De interpreter wordt alleen nog maar gebruikt om de interrupt-opdrachten tot een later tijdstip uit te stellen. Dat tijdstip is aangebroken als een RETURN of een EI wordt uitgevoerd. EI staat voor 'enable interrupt', waarmee wordt bereikt (zoals met RETURN) dat alle interrupts weer worden uitgevoerd.

DI en EI worden bij alle grafiekcommando's gebruikt om te voorkomen dat de grafiekcursor door een tweede interrupt ongewild wordt veranderd.

Voor een beter begrip van de prioriteiten en interruptbegrenzingen volgt hier een klein voorbeeldprogramma. Het bevat twee interruptroutines: het hoofdprogramma (regel 30) bestaat uit een eindeloze lus:

```
1 CLS:PRINT t;''Seconden''
2 LOCATE 6,5: PRINT''Seconden''
10 EVERY 50,0 GOSUB 100
20 EVERY 50,1 GOSUB 200
30 GOTO 30
100 t = t + 1: LOCATE 1,1: PRINT t
101 WHILE INKEY$ = '''':WEND
102 RETURN
200 x = x + 1:LOCATE 1,5:PRINT x
201 RETURN
```

Beide interruptroutines verhogen elke volle seconde de teller met 1. De routine van regel 200 heeft een grotere prioriteit omdat hierbij de timer 1 wordt gebruikt. Het onderprogramma vanaf regel 100 heeft daarentegen meer tijd nodig, omdat hierbij op het indrukken van een toets wordt gewacht (regel 101). Als u dit programma laat lopen, zult u zien dat de eerste routine regelmatig door de tweede wordt onderbroken (de bovenste secondenteller verandert alleen na het indrukken van een toets, de tweede laat zich hierdoor echter niet beïnvloeden).

Voegt u nu in regel 100 voor de uitdrukking 't = t + 1' het commando DI toe. Daarmee worden, voor de duur van de eerste routine, alle verdere interrupts afgebroken. Als u het programma nu weer start, wordt ook de tweede secondenteller pas na het indrukken van een toetst verwerkelijkt, omdat de tweede interruptroutine moet wachten op het vrijgegeven door RETURN aan het einde van de eerste routine.

Het laatste commando in verband met de interrupts heet REMAIN. Dit leidt een dubbel leven, want het heeft twee mogelijke vormen:

a. REMAIN (Y)
b. PRINT REMAIN (Y) of A = REMAIN (Y) enz.

Hiermee wordt in elk geval bereikt dat de timer wordt teruggezet, dat wil zeggen, dat hij geen interrupts meer uitwisselt.

Wordt REMAIN als functie ingeschakeld (vorm b.), dan geeft hij ook nog aan hoeveel 1/50ste seconden er nog zijn tot de volgende onderbreking.

7.3. IDEEËN VOOR HET PROGRAMMEREN MET INTERRUPTS

Het meest genoemde voorbeeld voor een interruptprogramma is de klok, die regelmatig op een of andere plaats op het beeldscherm wordt geprojecteerd. Daartoe kan men elke seconde (EVERY 50 ...) een subroutine oproepen waarbij gemakkelijk de secondenteller met de waarde 1 wordt verhoogd. Een probleem hierbij is dat de klok na elke toepassing van de cassetterecorder achterloopt, omdat deze de timer vasthoudt.

Maar hoe is het met een raadspel, waarbij het niet alleen aankomt op het goede antwoord, maar ook op de tijd. Het verloop zou als volgt kunnen zijn:
1. U stelt de kandidaat een vraag.
2. U geeft diverse mogelijke antwoorden die allemaal een kengetal bezitten. Dit kengetal kan met INKEY$ worden opgevraagd.
3. Via een AFTER-commando wordt na een bepaalde tijd (bijv. na 10 seconden) de INKEY$-lus afgebroken. De kandidaat krijgt dan, afhankelijk van het antwoord en de tijd, plus- of minpunten.

Punt 2 is overigens niet door een INPUT-opdracht te programmeren, omdat een interrupt nooit een lopende opdracht onderbreekt, dus ook niet een INPUT.

Een andere toepassing zou kunnen zijn om twee tekens op het beeldscherm periodiek met elkaar te verwisselen. Zo kan men bijvoorbeeld bewegingen nabootsen. Het eerste teken laat bijvoorbeeld een PAC-MAN met open mond zien en het tweede de PAC-MAN met zijn mond dicht. Als door een interruptroutine deze twee tekens regelmatig worden omgewisseld, ontstaat de indruk van een kauwbeweging.

Toepassingen van deze soort zijn er vele. Het komt op uw eigen fantasie aan wat u ervan kunt maken.

8. SOUND

Vele vaklieden noemen de CPC464 een uitstekende computer. Ook het oproepen van geluiden hoort tot de gelukte details. Omdat ik van mening ben dat men de SOUND-programmering zelf moet horen om dat te kunnen beoordelen, vindt u hiervan geen verdere toelichting. In plaats daarvan wil ik wijzen op de uitleg in het handboek en alleen de toepassingen beschrijven die men daaruit kan halen.

8.1. MINI-SYNTHESIZER

De ENT- en ENV-toevoegingen bieden de programmeur een onbegrensd aantal mogelijkheden om de toon te veranderen en zelfs om instrumenten na te bootsen. Met deze twee opdrachten kan men alle noodzakelijke factoren, behalve de klankkleur, beinvloeden. Hieronder staat een programma, waarbij door het indrukken van een toets alle parameters kunnen worden veranderd, waardoor het mogelijk is met de omhullingskrommen te experimenteren.

```
10 MODE 2:WINDOW #1,2,46,2,7:REM MENU-& WERKVENSTER
20 WINDOW #2,1,40,9,23:REM ENV-VENSTER
30 WINDOW #3,42,80,9,23:REM ENT-VENSTER
40 MOVE 0,272:DRAW 639,272
50 MOVE 0,32:DRAW 639,32
60 MOVE 320,32:DRAW 320,272
70 MOVE 363,272:DRAW 368,399
80 LOCATE 49,2:PRINT"MINI-SYNTHESIZER VERSIE 1.0"
90 LOCATE 49,4:PRINT CHR$(164)" 1985 DATA-BECKER GmbH"
100 LOCATE 49,6:PRINT "AUTEUR: HANS JOACHIM LIESERT"
110 LOCATE 2,25:PRINT"1 = AANTAL STAPPEN     2 = STAPGROOTTE
    3 = DUUR VAN DE PAUZE
120 LOCATE #2,3,2:PRINT #2,"ENVELOPE VOLUME"CHR$(10)
130 PRINT #2,"              1           2           3"CHR$(10)
140 PRINT #2,"      1       0           0           0"CHR$(10)
150 PRINT #2,"      2       0           0           0"CHR$(10)
160 PRINT #2,"      3       0           0           0"CHR$(10)
170 PRINT #2,"      4       0           0           0"CHR$(10)
180 PRINT #2,"      5       0           0           0"
190 LOCATE #3,3,2:PRINT #3,"ENVELOPE TOON"CHR$(10)
200 PRINT #3,"              1           2           3"CHR$(10)
210 PRINT #3,"      1       0           0           0"CHR$(10)
220 PRINT #3,"      2       0           0           0"CHR$(10)
230 PRINT #3,"      3       0           0           0"CHR$(10)
240 PRINT #3,"      4       0           0           0"CHR$(10)
250 PRINT #3,"      5       0           0           0"
260 DATA "q","2","w","3","e","r","5","5","t","6","y","7","u","i"
270 DIM T$(12):FOR i=0 TO 12:READ t$(i):NEXT i
280 DIM v(5,3),t(5,3)
290 SPEED KEY 255,10:ENV 1:ENT 1
300 LOCATE 1,1:c=1
1000 CLS #1:PRINT #1,"HOOFDMENU"CHR$(10)
1010 PRINT #1,"SPATIEBALK = MELODIE SPELEN"
1020 PRINT #1,"V          = ENV VERANDEREN"
1030 PRINT #1,"T          = ENT VERANDEREN"
```

```
1040 a$="":WHILE a$="":a$=INKEY$:WEND
1050 IF a$=" " THEN 1100
1060 IF a$="v" THEN 1200
1070 IF a$="t" THEN 1300
1080 GOTO 1040
1100 CLS#1:PRINT #1,"KLAVIER"CHR$(10)
1110 PRINT #1," 2 3   5 6 7"
1120 PRINT #1,"q w e r t y u i"CHR$(10)
1125 PRINT #1,"SPATIEBALK = MENU"
1130 a$="":WHILE a$="":a$=INKEY$:WEND
1140 IF a$=" " THEN 1000
1150 FOR i=1 TO 12:IF a$<>t$(i) THEN NEXT i:GOTO 1130
1160 per=ROUND(125000/440/2↑(i/12))
1165 c=c*2:IF c=8 THEN c=1
1170 SOUND c,per,0,0,1,1
1180GOTO 1130
1200 CLS #1:PRINT #1,"ENV VERANDEREN"CHR$(10)
1210 PRINT #1,"EINDE IS 0"CHR$(10)
1220 INPUT #1,"WELK ELEMENT ( REGEL, KOLOM";r,k
1230 IF r*k=0 THEN 1000
1240 INPUT #1,"WAARDE";v(r,k)
1250 LOCATE #2,k*10,4+2*r:PRINT #2,v(r,k);"    "
1260 ENV 1,v(1,1),v(1,2),v(1,3),v(2,1),v(2,2),v(2,3),
     v(3,1),v(3,2),v(3,3),v(4,1),v(4,2),v(4,3),v(5,1),v(5,2),v(5,3)
1270 GOTO 1230
1300 CLS #1:PRINT #1,"ENT VERANDEREN"CHR$(10)
1310 INPUT #1,"EINDE IS 0"CHR$(10)
1320 INPUT #1,"WELK ELEMENT ( REGEL, KOLOM";r,k
1330 IF r*k=0 THEN 1000
1340 INPUT #1,"WAARDE";t(r,k)
1350 LOCATE #3,k*10,4+2*r:PRINT #3,t(r,k);"    "
1360 ENT -1,t(1,1),t(1,2),t(1,3),t(2,1),t(2,2),t(2,3)
     t(3,1),t(3,2),t(3,3),t(4,1),t(4,2),t(4,3),t(5,1),t(5,2),t(5,3)
1370 GOTO 1300
```

Na het starten verschijnt in de linkerbovenhoek van het beeldscherm het menu. Het gewenste deel van het programma kan gewoon door het indrukken van een toets worden gekozen. Zolang alle parameters nog op nul staan, heeft het spelen van een melodie nog weinig nut. Daarvoor moet minstens de geluidssterkte ingevoerd worden. Ook moet het element dat veranderd moet worden de nieuwe waarde krijgen. Meteen wordt de inhoud van de eronder staande matrix veranderd. Alle parameters worden rechtstreeks aan de diverse commando's meegedeeld en u kunt ze dan ook gemakkelijk in uw programma's toepassen.

Ook bij dit relatief omvangrijke programma nog een aantal aantekeningen:
De regels 10 tot 270 bouwen het beeldscherm en de drie vensters op. Twee van deze vensters worden gebruikt voor de uitvoer van de ENV- en ENT-parameters. In het venster dat overblijft, komen alle aanwijzingen te staan die voor het bedienen noodzakelijk zijn. In regel 260 komen de afzonderlijke toetsen te staan die we voor de diverse tonen nodig hebben. Deze worden in de array t$ gezet. Door dit te vergelijken, kan aan elke toets de veldindex als 'toonnummer' worden toegevoegd. Dit nummer wordt gebruikt voor het berekenen van de toonperiode.

In regel 280 worden de twee paramaterarray's gedimensioneerd. Regel 290 schakelt tenslotte de REPEAT-functie uit en wist alle vorige gegevens uit.

Bij regel 1000 begint het eigenlijke programma. Allereerst wordt het menu verstrekt. In de daaropvolgende regels wordt een teken van het toetsenbord gehaald (a$), op grond waarvan daarna de verschillende programma-onderdelen worden gestart.

In de regels 1100 tot 1180 staat het deelprogramma 'melodie spelen'. Interessant is hier in het bijzonder de FOR...NEXT-lus in regel 1150. Deze wordt slechts dan voortgezet als er geen overeenstemming is tussen a$ en t$(i). Anders wordt uit i (= toets- resp. toonnummer) de toonperiode (per) voor het SOUND-commando berekend.

In regel 1200 staat het deel voor het veranderen van het ENVelope-volume. Hier is eigenlijk niets bijzonders aan de hand: dankzij de venstertechniek kunnen de nodige gegevens gemakkelijk met INPUT worden ingelezen. Het laatste deel dient voor het veranderen van de ENT-parameter en is identiek met het deel voor ENV.

8.2. HOE WORDT EEN TOON ONTWORPEN?

Zoals u uit het handboek van de CPC464 al hebt kunnen opmaken, kan men met de omhullingskrommen de klankkarakteristieken van verschillende instrumenten nabootsen. Om dit te bereiken, is niet alleen veel geduld bij het experimenteren nodig, maar toch ook wel een zekere kennis omtrent de omhullingskrommen van 'echte' tonen. Zo is het mogelijk om uit het geheugen vast te stellen dat de toon van een trompet net zo snel ophoudt als hij begint, terwijl dat bij een gong anders is; die begint plotseling en galmt daarna heel lang door. Een piano heeft ook een zeer abrupt begin en klinkt eveneens lang na, maar er is een verschil in toonaard, omdat de laatste complexer klinkt. We willen in de volgende regels proberen diverse instrumenten door verandering van de omhullingskromme te simuleren.

Een glazen klok heeft een zeer zuivere toon, dat wil zeggen dat de toon niet verandert door de frequentie. Daarom wordt de toonhoogte door de ENT-parameter niet veranderd. De geluidssterkte moet daarentegen direct op de hoogste waarde worden gebracht om daarna heel langzaam naar 0 terug te zakken. De twee parametermatrices moeten er daarom als volgt uitzien:

1	15	1	1	0	1
1	0	1	1	0	1
1	0	1	1	0	1
12	-1	8	1	0	1
2	-1	20	1	0	1

of in de ENV-opdracht als:

ENV 1,1,15,1,1,0,1,1,0,1,12,-1,8,2,-1,20

Een metalen klok klinkt ietwat ratelend. U kunt dit met uw CPC bereiken indien u door de omhullingskromme de frequentie voortdurend laat toe- en afnemen. Het commando hiervoor is:

```
ENT -1,1,1,3,1,-1,3,1,0,1,1,1,3,1,-1,3
```

Helaas heeft de voortbrenging van de tonen van uw computer een nadeel doordat er geen mogelijkheid bestaat om de klankkleur te veranderen. In tegenstelling tot andere computers kunt u met uw CPC geen doffe fluittonen laten horen, maar slechts heldere klanken. Daarom bestaan er voor diverse instrumenten (in het bijzonder voor de houten blaasinstrumenten) geen manieren om die na te bootsen. Ook de typische klank van een trompet is nauwelijks uit uw computer te halen (in ieder geval niet uit BASIC). Ondanks dat zal ik hier toch een aantal omhullingskrommen beschrijven waarmee toch minstens sterke klankverwantschappen zijn te herkennen.

Een accordeon heeft, afhankelijk van de manier waarop men die bespeelt, een relatief langzame verandering in de geluidssterkte en ook een lange nagalm. Bovendien klinkt de toon bepaald niet zuiver en 'snort' hij een beetje. Met:

```
ENV 1,7,2,1,1,1,1,1,0,1,1,0,1,15,-1,1,1,
ENT -1,1,0,3,1,-1,1,1,0,2,1,0,1,1,1,1,
```

kan men dit bereiken.

Alle instrumenten waarbij de geluiden door middel van snaren ontstaan, hebben een karakteristieke geluidskromme. Na een zeer snel begin zakt de sterkte iets af, om daarna langzaam weg te sterven. Het commando hiervoor luidt:

```
ENV 1,1,15,1,1,-3,2,1,0,1,1,0,1,12,-1,4
```

Met verschillende omhullingskrommen kunnen nu diverse snaarinstrumenten worden nagebootst. Een pianoachtige toon krijgt u door:

```
ENT -1,1,1,3,1,-1,3,1,0,3,1,1,3,1,1,3,1,-1,3
```

De ietwat vervormde klank van de banjo is met het volgende te imiteren (toegegeven, het is meer slecht dan echt):

```
ENT -1,1,2,1,1,0,2,1,0,2,1,-2,1,1,0,4
```

Dit is slechts een klein fragment van alle mogelijkheden. Maar tenslotte kan men nog andere dingen dan alleen maar muziekinstrumenten nadoen. Hoe zou het gaan met een drumstel, waarbij een kort geruis met een harde aanslag en een korte nagalmtijd geprogrammeerd moet worden? Ook kunt u fantasietonen bedenken. Uw creativiteit kan hier alle kanten uit.

Samenvatting: omhullingskrommen:

Klok:	ENV 1,1,15,1,1,0,1,1,0,1,12,-1,8,2,-1,20
Metalen klok:	ENT -1,1,1,3,1,-1,3,1,0,1,1,1,3,1,-1,3
Accordeon:	ENV 1,7,2,1,1,1,1,1,0,1,1,0,1,15,-1,1
	ENT -1,1,0,3,1,-1,1,1,0,2,1,0,1,1,1,1
Snaarinstrument:	ENV 1,1,15,1,1,-3,2,1,0,1,1,0,1,12,-1,4
Piano:	ENT -1,1,1,3,1,-1,3,1,0,3,1,1,3,1,-1,3
Banjo:	ENT -1,1,2,1,1,0,2,1,0,2,1,-2,1,1,0,4

9. BASIC EN HET BEDRIJFSSYSTEEM

De vaardigheden van de interpreter en van het bedrijfssysteem kunnen heel nuttig zijn. Een deel daarvan hebt u inmiddels leren kennen.

9.1. HOE WORDEN BASIC-REGELS OPGESLAGEN?

U hebt zich zeker al eens afgevraagd hoe een BASIC-regel in het geheugen komt te staan. Zoals u reeds uit hoofdstuk 2 weet, hebben de commando's zoals PRINT, SIN, SAVE enz. een speciale code. Deze code noemt men ook wel TOKEN. Met deze methode wordt heel veel geheugenruimte uitgespaard (voor PRINT bijv. geen 5 lettercodes maar slechts 1 TOKEN-byte). Bovendien hoeft de interpreter tijdens de duur van het programma deze letters niet te analyseren.

De namen van variabelen en tekens worden daarentegen als ASCII-codes opgeslagen. Verbindingen zoals *, / maar ook AND enz. en ook de relatietekens bezitten eveneens een TOKEN-code. Getallen worden in een zeer gecompliceerd format opgeslagen, dat afhankelijk van het type (geheel getal of breuk) en de grootte van dat getal verandert.

In het geheugen kan men TOKEN herkennen omdat de bytes een grotere waarde hebben dan 127. De namen van de variabelen en de strings hebben allen een waarde die kleiner is dan 128, dus heeft de interpreter alleen maar een codetabel nodig.

We maken nu een kleine test. Wis daarom eerst het eventuele programma uit met NEW en tik dan het volgende programma in (teken voor teken precies zo):

```
100 PRINT ''TEST''
```

We willen nu zien hoe dit 'programma' er in het geheugen uitziet. Voer daartoe in de direct-mode de volgende regel in:

```
FOR i = 368 TO 383:PRINT PEEK(i):NEXT
```

Bij byte 368 begint ons BASIC-geheugen; het bovenstaande geeft dus de eerste 16 bytes. Op het beeldscherm verschijnen dan de volgende waarden:

```
13 0 100 0 191 32 34 116 101 115 116 34 0 0 0 0
```

De eerste vier bytes stellen twee 16-bits-getallen voor. De eerste twee geven de lengte van de regel aan (13 bytes). Als de interpreter een speciale regel zoekt (bijv. bij GOTO), onderzoekt hij eerst of het eerste regelnummer het goede is. Als hij dit niet heeft gevonden, wordt eenvoudig de regellengte bij de bestaande opgeteld. En zo bereikt de interpreter de volgende regel en kan hij ook deze toetsen.

In de volgende twee bytes staat het regelnummer. Dan volgt het eerste TOKEN; de 191 staat voor PRINT. 32 en 34 zijn de ASCII-codes voor 'spatie' en 'aanhalingsteken'. Zoals niet moeilijk te raden is, vormen de volgende vier bytes de codes voor het woord 'TEST'. Met '34' voor het laatste aanhalingsteken eindigt onze regel. Omdat er verder geen programmaregels meer zijn opgeslagen, staan de laatste vier bytes op 0.

Met deze structuur kunnen we een beetje manipuleren. Vindt de interpreter bij het begin van het programmageheugen een regel met de waarde 0, dan wordt deze niet geLIST, ofschoon hij wel normaal wordt uitgevoerd. Een sprong naar deze regel functioneert niet, omdat BASIC een 0 niet als regelnummer accepteert. Als u dus de eerste regel voor LIST wilt beveiligen, hoeft u alleen maar de bytes 370 en 371 door middel van POKE op 0 te zetten. Probeer dat maar eens.

Bij regels die niet aan het begin van de BASIC-tekst staan, is dit niet mogelijk. Het regelnummer kan weliswaar op 0 worden gezet, maar deze regel wordt verderop normaal geLIST.

Misschien kent u van andere computers het commando RENEW, ook wel OLD genoemd. Het doel hiervan is om een programma dat met NEW is uitgewist, weer te reconstrueren. Dit functioneert omdat een aantal computers (zoals bijv. de Commodore) bij het commando NEW niet het geheugen uitwist, maar alleen de pointer, zodat voor de interpreter de indruk ontstaat dat er geen programma meer is. Onze CPC houdt zich echter niet aan deze spelregels. En daarom kan ook met een do-it-yourself methode een dergelijke opdracht niet ingebouwd worden.

Samenvatting: format van een BASIC-regel.

Alle opdrachten en commando's worden met TOKEN in het geheugen opgeslagen, teksten en variabelen in de ASCII-code. De eerste twee bytes bepalen de lengte van de regel en de volgende twee het regelnummer. Het regelnummer kan met POKE kunstmatig veranderd worden. Hiermee is een LIST-beveiliging mogelijk voor de eerste regel als die de waarde 0 krijgt.

9.2. GARBAGE COLLECTION

Hebt u wel eens van een verzameling 'afval' gehoord? (Dat is namelijk de vertaling van 'garbage collection'.) En dat in verband met computers? Want wat als familie van de stadsreiniging wordt aangekondigd, heeft een totaal andere betekenis. Om dat te begrijpen, hebt u enige voorkennis nodig.

Als de interpreter met stringvariabelen werkt, produceert hij veel afval. Elke keer als een string wordt veranderd (al is het maar één letter), krijgt hij een nieuwe plaats toegewezen, terwijl de 'oude' string onveranderd in het geheugen blijft staan en ook niet wordt getransporteerd. Deze staat dan ergens nutteloos in het geheugen. Op een zeker moment staat het hele geheugen vol, terwijl er slechts een klein gedeelte gebruikt gaat worden. De rest is afval. Om nu een nieuwe string eraan te kunnen toevoegen, moet er eerst ruimte worden gemaakt. Dit proces noemt men 'garbage collection'. Omdat dit veel tijd in beslag neemt, is het voor elke computerfan de grote schrik. Een klein voorbeeld zal dit duidelijk maken:

```
DIM a$(8000)
FOR i = 0 TO 8000: a$(i) = CHR$(1):NEXT i
```

Met deze twee opdrachten hebben we het geheugen aardig gevuld (zowat tot aan de bovenkant van de onderlip). Met PRINT FRE("") kunnen we nu deze garbage collection weer verwijderen (niet met FRE(0))!

Het probleem hierbij is echter dat deze operatie zo vreselijk lang duurt (een aantal minuten). De interpreter moet namelijk van de 8001 strings (die allemaal gelijk zijn) er 8000 uitwissen. Het is natuurlijk ook zinloos om 8000 gelijke strings op te slaan; men kan ze veel beter met een lus oproepen.

Om te voorkomen dat uw programma door een dergelijke garbage collection gedurende lange tijd zijn werk niet kan doen, is het verstandig om dit, vooral bij string-intensieve programma's, regelmatig er tussendoor te doen. Weliswaar duurt de hele bewerking dan niet korter, maar het is minder storend als er kleine stukken tegelijk worden gedaan.

9.3. OPGEPAST: FOUTEN!

Geheel volgens de oude regel: 'Waar gehakt wordt, vallen spaanders', heeft ook het BASIC ROM een fout, die men echter niet zonder meer kan zien. Bovendien komen deze fouten voor in REM-statements, waardoor men ze gemakkelijk over het hoofd ziet. (REM-statements hebben immers de naam dat ze geen wezenlijk deel uitmaken van het programma).

Als in een REM-regel een van de stuurtekens 'pijl rechts' (TAB-toets) of de 'loodrechte streep' (SHIFT + slinger a) staat, dan verslikt de interpreter zich; er ontstaan dan onberekenbare verschijnselen. Als men geluk heeft, beperkt dat zich tot een sprong naar regel 32511. Als deze regel niet bestaat, wordt het programma gewoon afgebroken. Daarbij kan het ook voorkomen dat er hele delen van het programma worden uitgewist of dat deze delen alleen maar onzichtbaar worden, dat wil zeggen dat men ze met GOTO, GOSUB, RUN of LIST niet meer kan bereiken, terwijl REM toch nog steeds naar deze 'oude' regels springt.

De uitwerking van deze REM-fout is heel verschillend en hangt waarschijnlijk af van de rest van het programma en van de plaats van het REM-commando. Zo kan er ook een verandering van de beeldschermmodus optreden. Misschien worden deze verschijnselen ook veroorzaakt door andere dan de hiervoor genoemde stuurtekens.

Het is mogelijk dat er na voldoende onderzoek van dit fenomeen nog een tweede LIST-beveiliging wordt gemaakt, of er zijn - zoals bij de HP-41 - synthetische commando's. Dat zijn commando's die eigenlijk niet door de fabrikant waren te voorzien. Voor opmerkingen over dit thema houd ik me overigens graag aanbevolen.

9.4. ONBEKENDE EIGENSCHAPPEN

Vergelijkbaar met de onbekende commando's uit hoofdstuk 2.5. wil ik hier een aantal eigenschappen van BASIC en het bedrijfssysteem noemen die niet in het handboek staan.

De eerste eigenschap gaat over de editor, die bijv. voor de cursorbesturing, de COPY-toets en het invoeren van regels kan worden gebruikt. Als er al een aantal tekens in een nieuwe regel staan of u een reeds bestaande met behulp van EDIT wilt bewerken en een van de tekens wilt corrigeren, dan moet u allereerst de cursor naar deze plaats brengen. Alle tekens die u nu gaat tikken, worden ertussen gevoegd, terwijl de oude tekens gewoon worden opgeschoven. Dit kan heel lastig zijn als men over de oude tekens heen wil schrijven. Dit invoegen gaat evenwel gemakkelijk als men tegelijk de CTRL- en de TAB-toets indrukt. Opnieuw indrukken schakelt weer in.

Voor de tweede eigenschap kan de BASIC-interpreter gebruikt worden. Als u een array (bijv. A(1)) voor het eerst oproept zonder dat deze eerst is gedimensioneerd, bepaalt de BASIC-interpreter zelf een zogenoemde voordimensionering met 11 elementen. Dit komt dan in de plaats te staan van DIM A(10).

Hieraan moet overigens worden toegevoegd dat het weglaten van de DIM-opdracht alleen dan de moeite waard is als het veld werkelijk 11 elementen moet bezitten. Voor elk element dat is gedimensioneerd, wordt geheugenruimte gereserveerd die door andere gegevens niet gebruikt kan worden. Het maakt een programma niet beter leesbaar als er plotseling een veld optreedt dat niet eerst nadrukkelijk werd gedimensioneerd.

9.5. VAN IEMAND DIE EROP UITGING OM BASIC BANG TE MAKEN

Zoals elk programma, heeft ook de BASIC-interpreter een aantal geheugenbytes nodig waarin hij zijn interne gegevens kan opslaan. Omdat deze bytes met behulp van POKE te beïnvloeden zijn, kan men BASIC danig in de war brengen.

Nemen we bijvoorbeeld de in BASIC ingebouwde 'programmabeveiliging'. Als we een beveiligd programma vanaf de cassette willen inlezen, wordt dit door de interpreter op een speciale plaats opgemerkt. Wordt daarna het einde van het programma bereikt, dan wordt deze plaats onderzocht en eventueel een NEW gegeven. Dit byte staat op de plaats &AE45. Als daar een andere waarde dan 0 staat, wordt het programma dat zich in het geheugen bevindt, beveiligd. Door middel van POKE &AE45,1 kan men deze beveiliging ook met de hand inschakelen. POKE &AE45,0 werkt dus precies de andere kant op.

Het byte met het adres &AC00 heeft een andere nuttige eigenschap. Afhankelijk van de waarde ervan worden bij het invoeren van programmaregels de overbodige spaties uitgewist of ze blijven (dat is de normale wijze) behouden. Met POKE &AC001, kan men deze zeer nuttige en geheugensparende functie inschakelen.

In hoofdstuk 3.1. hebben we de werking van HIMEM bestudeerd, waarbij het bovenste deel van de geheugenruimte wordt gereserveerd. Men kan echter net zo goed het begin van het BASIC-geheugen naar boven verleggen. Daarvoor bestaat er weliswaar geen afzonderlijke opdracht, maar het biedt misschien wel nieuwe mogelijkheden. De geheugencellen &AE81 en &AE82 bezitten de pointer voor het begin van het programma. Normaliter wordt hier naar het adres 367 verwezen, dus naar de byte die voor het programma staat. Veranderen we deze wijzer, dan wordt hierdoor niet het geheugen veranderd, maar de interpreter zoekt nu op een andere plaats naar het programma. Alle delen die voor dit nieuwe startpunt liggen, worden door BASIC niet meer herkend. Op deze manier kan men delen van het programma verbergen als men de adressen weet waar de afzonderlijke regels ophouden.

Het is overigens heel gemakkelijk om het hele programma te laten verdwijnen. De registers &AE83 en &AE84 bevatten de wijzer voor het einde van het programma. Met het commando:

POKE &AE81,PEEK(&AE83):POKE &AE81,PEEK(&AE84):NEW

wordt het begin voor de interpreter verschoven tot voorbij de tekst van het programma. Het commando NEW is nodig om oude variabelen uit te wissen die door BASIC als programmaregels kunnen worden gezien; ons programma wordt daardoor echter niet beïnvloedt omdat de wijzer al vóór die tijd was veranderd. Nu kan men zonder meer een nieuw programma in het geheugen laden en verwerken zonder het oude te beïnvloeden. Alleen de variabelen worden uitgewist.

POKE &AE81,111:POKE &AE82,1

brengt ons weer terug naar de oorspronkelijke staat.

Dergelijke verborgen programma's hebben overigens een interessante eigenschap. Als de commando's voor het veranderen van de wijzer tijdens het programmaverloop worden uitgevoerd, heeft dat geen invloed op de volgende regels. Alleen de sprong-opdrachten werken niet meer omdat de interpreter bij deze bewerkingen de wijzer als oriëntatiepunt voor het zoeken naar een regel nodig heeft.

Samenvatting: Over + LIST en van BASIC

&AE45 dient als merkbyte voor het beveiligen van programma's.
POKE &AC00,1 schakelt de comprimeermodus in, wat wil zeggen dat alle overbodige spaties worden uitgewist (uitschakelen met POKE AC00,0).

De bytes &AE81 en &AE82 wijzen naar de byte van het begin van een programma, &AE83 en &AE84 naar het einde van het programma.

9.6 NOG EEN PAAR TRUCS

In het algemeen worden de waarden van de commando's zoals SPEED xxxx of dergelijke snel vergeten als men er weinig mee experimenteert. Dan is goede raad meestal duur. Bij de SPEED INK-opdracht weet ik echter een goed hulpmiddel. De twee parameters van dit commando worden namelijk vanuit BASIC gemakkelijk in de twee bytes &BID7 en &BID8 opgeslagen, waar het bedrijfssysteem ze naar behoefte (bij elke kleurwisseling) vandaan haalt. En wat het bedrijfssysteem kan, kunnen wij met PEEK al veel langer...

U hebt vast en zeker ook wel eens van een zelf te definiëren teken gebruik gemaakt. Misschien ging het bij u net als bij mij - ik ergerde me er steeds aan dat ik alle 8 bytes moest berekenen ook als er maar een enkele punt moest worden veranderd. Wel, ook hierbij hebben we een aardig hulpmiddel ter beschikking. De zelf te definiëren tekens CHR$(240) tot CHR$(225) staan in het RAM van geheugenplaats &AB80 tot 8ABFF opgeslagen. Voor elk teken zijn 8 bytes gereserveerd, die we gemakkelijk met PEEK kunnen uitzoeken. Dan hoeft alleen het punt dat we willen veranderen opnieuw worden berekend.

Het adres van dat teken kunnen we met de volgende formule berekenen:

Adres = 43904 + (X-240) * 8

X is daarbij het nummer van het gewenste teken in het gebied van 240 tot 255.

Samenvatting: trucs voor het bedrijfssysteem

De parameters van het SPEED + INK-commando kan men in de bytes &BID7 en &BID8 opzoeken.
De tekenmatrices van de zelf te definiëren karakters liggen in het gebied &AB80 tot &ABFF.

10. HULP- OF RANDAPPARATEN EN HET FUNCTIONEREN ERVAN

Elke computer heeft een aantal hulpapparaten nodig om goed te kunnen functioneren. Bij sommige merken, zoals bijv. bij de IBM-PC, worden al deze apparaten afzonderlijk erbij geleverd, zoals het toetsenbord en de monitor. Gelukkig gaat dit bij de Schneider anders, want bij de CPC worden er zelfs nog een cassetterecorder en een monitor bijgeleverd. Maar ook bij de CPC kan men nog een aantal andere apparaten erbij nemen.

10.1. DE DISK-DRIVE

Behoort u ook tot de 'rustelozen' bij wie het laden van een programma van cassettes te lang duurt? Hebt u daarbij ook last van nerveuze spanningen als de luidspreker met een afschuwelijk geruis meedeelt dat het laden van het zesentwintigste blok is begonnen? In dat geval hebt u een disk-drive nodig. Ervaren computergebruikers weten allang wat een enorme arbeidsbesparing zelfs een langzame floppy-drive ten opzichte van een cassetterecorder betekent. Een klein onderzoekje bewijst dit al. Om een programma van 20K te laden, heeft een cassetterecorder met 'speedload' 2 minuten 27 seconden nodig, een floppy heeft aan een slordige 9 seconden al genoeg.

Maar dit is niet het enige voordeel van een disk-drive. Omdat op een diskette de verschillende magneetsporen (waarvan er 40 zijn) afzonderlijk kunnen worden gekozen, zoals dat ook bij een grammofoonplaat gebeurt, is het mogelijk om die nuttig toe te passen. Daarmee wordt bedoeld dat de computer niet eerst alle bestanden vanaf het begin hoeft door te zoeken naar het gewenste gegeven, maar kunt u als het ware zeggen: 'Haal de derde byte uit het adressenbestand'. De computer accepteert weliswaar alleen de vertaling in BASIC, maar die wordt dan ook bijzonder snel uitgevoerd. Op deze manier kan de diskette als een vergroot geheugen worden gebruikt.

De disk-drive heeft overigens nog meer voordelen (er is bijv. een inhoudsopgave met alle gegevens die er op de floppy staan), maar heeft ook nadelen. Ten eerste, is hij nogal duur en ten tweede hebt u er speciale besturingselektronica voor nodig, de 'controller'. Bij de eerste disk-drive die u koopt, wordt deze erbijgeleverd en daarom is hij ook duurder. De controller kan twee disk-drives tegelijk besturen, waardoor u de tweede disk-drive aan de eerste kunt koppelen. Zonder controller werkt er echter niets. Bovendien bevat de disk-drive nog een ROM, waarmee het aantal commando's voor het besturen van de disk-drive wordt uitgebreid.

10.2. DE PRINTER

Een van de voortreffelijke mogelijkheden van uw CPC is de ingebouwde printer-interface. In tegenstelling tot uw floppy heeft deze interface geen andere software of controller nodig, want die zijn al ingebouwd.

Het betreft hierbij in het bijzonder de Centronics-interface, die is aangepast aan de interface van de Centronic-printer. De meeste fabrikanten van printers gebruiken eveneens deze interface. Daarom kunt u de meeste printers die op de markt verschijnen gewoon op uw CPC aansluiten. Maar ook hierbij is er weer een kleine moeilijkheid te overwinnen. De CPC draagt slechts de eerste 7 bits van de 8-bits-ASCII-codes over aan de printer en dus 'pikt' hij de helft in van de bestaande tekens. Hier hebt u overigens minder last van dan u denkt, omdat de codes 0 tot 127 de belangrijkste tekens bevatten en de codes 128 tot 255 de grafische tekens.

Vaak komt het voor dat de ASCII-code van de printer niet in overeenstemming is met de ASCII-code van uw computer. In dat geval moet u een aanpassingstabel programmeren. Dat lijkt moeilijker dan het is. Verander gewoon de code van de printer die bij de overeenkomstige code van de CPC hoort, noem deze nieuwe code ARRAY(X), waarbij X loopt van 0 tot 127. Als er nu gegevens moeten worden afgedrukt, dan doet u dat niet met PRINT + 8,CHR$(X) maar print + 8,CHR$(ARRAY(X)). Dan wordt niet de code van de computer gebruikt, maar de code van de printer.

10.3. DE 'JOYSTICK'

Zoals zoveel andere gebruikers zult u ook wel eens geprobeerd hebben een programma te schrijven waarbij de joystick moet worden gebruikt. In de meeste gevallen hebt u daarbij de verschillende posities van de joystick met een FOR...NEXT-lus benut, en wel op een van de volgende manieren:

```
IF JOY(0) = 1 THEN 100
IF JOY(0) = 2 THEN 200
IF JOY(0) = 4 THEN 300
```

Deze methode is bijzonder langzaam en omslachtig. Het gaat veel beter met het volgende programma (de verklaring volgt later):

```
10 A = LOG(JOY(0))/LOG2
20 ON A GOTO 100,200,300...
```

Met regel 20 wordt er, afhankelijk van de waarde van A, naar verschillende programmaregels gesprongen. Zou u als variable gewoon de joy-waarde nemen, dan werkt het ON-commando niet naar wens omdat JOY geen op elkaar volgende waarden voor de verschillende richtingen oplevert (zoals 1, 2, 3, 4...), maar machten van 2 (1, 2, 4, 8, 16, 32). Daarom worden van deze machten de logaritmen van het grondtal 2 berekend, waardoor de gewenste waarden ontstaan.

De werkwijze van een joystick is heel eenvoudig. Hij bestaat simpel uit 5 of 6 meer of minder ingebouwde toetsen. Een, of soms twee ervan, worden als 'vuurknop' gebruikt en de andere zitten onder de stuurknuppel waarmee vier verschillende richtingen kunnen worden bereikt. Afhankelijk van de stand van deze knuppel wordt dan de bijbehorende toets ingedrukt. De CPC registreert dit en geeft dan de bijbehorende waarde af.

Er zijn natuurlijk een aantal verschillende joysticks in de handel. Eenvoudige en goedkope exemplaren werken met eenvoudig foliecontact en (oudbezitters van de ZX-81 hebben daaraan een niet zo beste herinnering). Andere exemplaren hebben daarentegen microschakelaars, wat men in vele gevallen kan merken aan een klik. Bij de aanschaf van een joystick moet men erop letten dat de stuurknuppel afgeronde zijkanten bezit, omdat er anders tijdens het spel snel een vermoeidheid kan optreden. Overigens passen de bekende Atari-joysticks ook op de Schneider-computer.

11. IETS OVER 'INTERFACES'

Aan de achterzijde van uw CPC bevindt zich een aantal aansluitingen en stekkers. Als u zich hebt afgevraagd wat dat allemaal betekent, moet u dit hoofdstuk maar eens lezen.

11.1. INVENTARISATIE VAN DE INTERFACES

Voor we ons met de aansluitingen bezighouden, zullen we eerst nog een keer een uiteenzetting geven over het begrip 'interface'.

Bij een interface gaat het niet altijd om een verbinding met een randapparaat. Deze aansluitingen geven ook toegang tot de computer. Het beste voorbeeld hiervoor is de 'expansion connector', waarbij op de behuizing van de computer simpel 'Floppy-disk' staat. Het gaat hierbij namelijk om een uitbreidingsmogelijkheid waarmee niet alleen de noodzakelijke verbinding met de floppy tot stand komt, maar waarmee ook de aansluiting met de ROM-module plaatsvindt. ROMs zijn geheugenbouwstenen en kunnen daarom direct door de processor worden gebruikt. Daarom bestaat de expansion-connector ook 'alleen maar' uit adres-, data- en besturingsingangen van de Z-80 plus nog een paar aanvullende stuursignalen.

Daarnaast vindt u een aansluiting voor de printer, die is aangepast aan de centronics-norm. Ook de printer-poort is verbonden met de processordata-ingang, echter niet met de adres- en besturingsingang. In de plaats daarvan is er een aantal stuurleidingen van de 8255.

Helaas hebben de ontwerpers van de CPC op deze plaats bezuinigd. In plaats dat de gebruikelijke 8-bits-opdracht wordt uitgevoerd, heeft men bij de CPC eenvoudig het achtste bit weggelaten. Maar waar ter wereld bestaat er eigenlijk wel een perfecte computer?

De aansluiting van de joystick is alleen een uitbreiding van het toetsenbord. Dit is met verschillende bouwstenen van de computer verbonden (daar komen we later nog op terug).

Ook de aansluitingen voor monitor en stereo-versterker zijn eigenlijk interfaces. Maar deze werken niet direct met de computer samen.

11.2. HOE WERKT EEN INTERFACE?

Het functioneren van een interface is bij alle computers in beginsel hetzelfde. De informatie die moet worden overgedragen, wordt door de processor aan de interface geleverd, die dan op zijn beurt het betreffende randapparaat aanzet en de gegevens overdraagt. Daarbij kan het nodig zijn dat deze apparaten moeten worden gesynchroniseerd. In dit geval worden via speciale stuurleidingen impulsen uitgewisseld om de overdracht mogelijk te maken.

Als er een data-overdracht moet plaatsvinden, moet de processor bovendien nog aangeven of deze gegevens moeten worden ontvangen, dan wel moeten worden verzonden. Afhankelijk van de gewenste modus wordt dan de poort op in- of op uitvoer geschakeld.

In sommige gevallen wordt er geen bijzondere interface toegepast, en dus ook niet bij de CPC. Dan neemt de processor zelf de noodzakelijke operaties over (wat dan natuurlijk iets langzamer gaat). De hardware beperkt zich hierbij tot het aanpassen van het spanningsniveau.

De voor de toepassing van een interface noodzakelijke programma's zijn alle al in ROM opgeslagen en kunnen met BASIC-opdrachten worden uitgevoerd, bovendien is een efficiënte interface-programmering (op een enkele uitzondering na) alleen maar in machinetaal mogelijk.

11.3. UW PERSOONLIJKE INTERFACE

Behoort u ook tot de onverbeterlijke hardware-specialisten die bij computers alleen maar aan draden en hete soldeerbouten denken? Bij de aanhangers van het type homo electronicus is vaak de wens te horen om gegevens van eigen instrumenten (bijv. thermometers) in te lezen. Voor dit doel is de Centronic-interface en de expansion-connector zeer geschikt. Beide zijn echter in de praktijk voor een andere toepassing bedoeld.

Dit geldt weliswaar ook voor de aansluiting van de joystick, maar die wordt meestal alleen voor spelletjes gebruikt. Ondanks dat kan hij toch ook voor serieuze toepassingen worden gebruikt.

De USER-PORT (zo staat het zwart op wit op de behuizing van de computer) kan vanuit BASIC heel goed met de functies 'joy' en 'inkey' worden toegepast. Daarmee zijn alle mogelijkheden voor contacten met de buitenwereld gegeven.

Aan de pennen 1 tot 7 (zie hiervoor het handboek: appendix 5) kunnen steeds twee schakelaars worden aangesloten. Van de eerste schakelaar worden de pennen met COMMON (pen 8) verbonden en van de tweede met COM 2 (pen 9). Met de laatste twee pennen onderscheidt de computer joystick 0 en 1. Bij het sluiten van een schakelaar wordt een ingang (pen 1 tot 7) verbonden met een COMMON-signaal. Voor de CPC is dat hetzelfde als het indrukken van een toets of het gebruiken van de joystick. U kunt dat door de genoemde commando's vaststellen.

Aan de ingangen van de joysticks kunt u willekeurige schakelaars aansluiten. Wat daar voor apparaten achter staan (relais, transistoren enz.) laten we aan uw eigen fantasie over.

11.4. ONDERZOEK VAN HET TOETSENBORD

Volledigheidshalve zal ik in dit gedeelte uiteenzetten hoe onderzoek van het toetsenbord werkt. Zoals u door na te tellen kunt vaststellen, heeft de CPC 73 toetsen (SHIFT wordt dan slechts één keer geteld), die allemaal op regelmatige tijdstippen (1/50ste seconde) moeten worden onderzocht. Voor dit doel is het toetsenbord elektrisch in 10 kolommen opgedeeld, die de Z-80 afzonderlijk kan inschakelen. Daarvoor moet alleen het nummer van de desbetreffende kolom naar 8255 worden gestuurd.

Is er een toets van de bedoelde kolom ingedrukt, dan wordt een bit op 0 gezet en anders houdt hij de waarde 1. Per kolom ontstaan er zo tot 8 afzonderlijke bits. Op deze manier kan men een hele byte samenstellen, die dan door de SOUND-chip (jazeker, de SOUND-chip) via de 8255 naar de Z-80 wordt teruggestuurd. De processor kan daardoor gemakkelijk, door het benutten van de uitgewiste bits, vaststellen welke toets er in de betreffende kolom is ingedrukt. Deze op het eerste gezicht ingewikkelde manier via de sound-chip wordt gebruikt omdat deze bouwsteen al van huis uit met de bijbehorende poort is uitgerust en de andere, vlugger bereikbare poorten niet met deze relatief langzame methode moeten worden belast.

Het principe van het onderzoeken van deze kolommen wordt overigens bij elke computer op meer of minder vergelijkbare manier uitgevoerd.

12. CASSETTERECORDER EN TOETSENBORD

De bediening van de cassetterecorder is in het CPC-handboek wat stiefmoederlijk behandeld. Dit tekort zal ik nu een beetje gladstrijken. En ook zijn er een aantal onbekende mogelijkheden bij de toepassing van het toetsenbord.

12.1. HOE BOUWT MEN GEGEVENS OP?

In het CPC-handboek hebt u zeker wel de omschrijving van het bestandstype 'ASCII' gelezen. Helaas werd daarbij de mogelijkheid verzwegen dat er niet alleen maar listings van een tekstverwerker kunnen worden gegeven (per SAVE "naam",A). De CPC is ook in staat om string- en rekenvariabelen op de band te zetten. Bovendien wil ik u laten zien hoe men ASCII-listings vanuit BASIC kan inlezen.

De naam 'ASCII-bestand' komt door het feit dat alle gegevens als een eenvoudige volgorde van ASCII-codes (of bytes) worden opgeslagen. Het enige verschil met een programmabestand is dat het niet eenvoudig is een deel van het geheugen op de band te kopiëren, maar dat de afzonderlijke invoeren met een CHR$(13) moeten worden afgesloten. Daarbij maakt het niets uit of deze gegevens uit variabelen of uit iets anders bestaan.

Het lezen en schrijven van deze bestanden lijkt sterk op het afdrukken op het beeldscherm; daarvoor hebben we de commando's PRINT + = S en INPUT + = S. Dat komt niet bij toeval tot stand, want ook bij een uitvoer naar het beeldscherm wordt CHR$(13) als scheiding van afzonderlijke informaties gebruikt. Het stuurteken heeft daarbij echter de opdracht om aan het bedrijfssysteem mee te delen dat de cursor naar de volgende regel moet worden verplaatst.

Laten we voor ons genoegen nu eens aannemen dat u in uw grenzeloze verzamelwoede, die een echte computerfan nu eenmaal heeft, twee hele variabelen met gegevens gevuld heeft die u ook morgen nog wilt bezitten. In dat geval kunnen de onderstaande programma's u misschien van dienst zijn:

```
10 REM GEGEVENS WEGSCHRIJVEN
20 OPENOUT "TEST"
30 PRINT + 9, a$
40 PRINT + 9,b
50 CLOSEOUT
```

```
10 REM GEGEVENS WEER TERUGLEZEN
20 OPENIN "TEST"
30 INPUT + 9, a$
40 INPUT + 9, b
50 CLOSE In
```

Het eerste programma schrijft deze twee variabelen a$ en b op de cassette. Met het tweede programma worden deze twee variabelen weer teruggehaald. Daarbij is de naam van de variabelen niet belangrijk; u kunt ze net zogoed x$ en y noemen. In tegenstelling hiermee moet het type van de variabelen wel kloppen, anders loopt u de kans om een TYPE-MISMATCH-ERROR te krijgen.

Bij het opslaan van ASCII-listings worden op een vergelijkbare manier strings gevormd die het BASIC ook als strings kan lezen. Als voorbeeld geven we ook hiervan een programma:

```
10 INPUT "HOEVEEL REGELS?";a:a = a-1
20 DIM a$(a)
30 OPENIN "filename"
40 FOR i = 0 TO a
50 INPUT + 9,a$(i)
60 NEXT
70 CLOSE IN
```

Na het aflopen van dit programma staan de regels in een string-array, waar u ze dan verder kunt bewerken. Helaas heeft dit systeem ook haken en ogen. Het BASIC gebruikt een komma als scheidingsteken voor strings. Daarom worden alle regels waarin een komma staat (en dat zijn er tamelijk veel) op de betreffende plaats gescheiden en wordt er een tweede string opgezet om de rest van de regel in op te slaan. Dit kan echter door de juiste programmering weer goedgemaakt worden. De geschikte programmering bestaat eenvoudig daaruit dat het commando INPUT moet worden vervangen door LINE INPUT. Met deze speciale vorm wordt een CHR$(13) als 'einde van een regel' markering gezien.

In het hierboven vermeldde voorbeeld moet u met de hand invoeren, hoeveel regels er in de listing voorkomen. Als u zich daarbij hebt vergist, kan er een 'EOF' optreden als u probeert meer regels in te lezen dan er zijn. Deze fout is te voorkomen. In BASIC bestaat er een functie die het einde van een bestand aangeeft. Deze functie is EOF (End Of File). Print EOF geeft de waarde 0 als er nog gegevens aanwezig zijn en aan het einde van het bestand geeft dit de waarde -1. Ons programma kan dan als volgt veranderd worden:

10, 20, 30 en 70 zoals hierboven
40 WHILE NOT(EOF)
50 LINE INPUT +9,a$(i):i + 1
60WEND

Door de WHILE-WEND-constructie worden er alleen gegevens van de band gelezen zolang ze er nog zijn. Helaas kan ook hierbij nog een foutmelding ontstaan en wel als de listing meer regels heeft dan er als strings werden gedimensioneerd.

Bij het ontwerpen van een eigen bestand kunt u dit voorkomen door bij het opbouwen hiervan eerst het aantal gegevens als normale variabele op de band te schrijven en dan pas de gegevens te laten volgen. Als deze gegevens later weer worden ingelezen, leest men eerst het aantal waarmee dan de velden worden gedimensioneerd, waarna pas de rest wordt ingelezen.

Op de geheugenplaatsen van &B807 tot &B816 en van &B84C tot &B85B is een zeer bijzondere toepassing mogelijk, waarin de namen van de INPUT- (&B807) en van de OUTPUT-bestanden (&B84C) door het bedrijfssysteem als ASCII-strings worden opgeslagen. Door de volgende regel kunt u de laatste OUTPUT-bestandsnaam teruglezen:

FOR i = &B84C TO &B85B:Print CHR$(PEEK(i));:NEXT

Hoe zou het zijn als u uw programma's met een testroutine zou kunnen uitrusten waarbij zou kunnen worden voorkomen dat de naam wordt veranderd? Daarvoor moet u aan het begin van het programma op de genoemde manier de naam van het bestand uitzoeken. Als deze naam niet in overeenstemming is met de 'echte' naam, dan wordt het programma gewoon met NEW uitgewist en wellicht wordt er dan ook nog enig commentaar aan toegevoegd.

En als u wilt weten met welke schrijfsnelheid het programma is opgenomen, hoeft u alleen maar PRINT PEEK(B8D1) in te tikken. Is het resultaat een 6, dan is er op de langzame manier geschreven, bij 12 is SPEED WRITE 1 gebruikt.

Samenvatting: Gegevens op de cassette

Met LINE INPUT kan men ook strings inlezen die een komma bevatten. De functie EOF wijst het einde van een bestand aan. Met deze twee commando's kan men ASCII-listings in een BASIC-arry aanbrengen die dan kunnen worden verwerkt. De naam van het laatste bestand kan men uit de geheugenplaatsen &B807-&B816 en &B84C-&B85B halen. Het byte &B8D1 geeft de schrijfsnelheid weer.

12.2. INKEY$ IN EEN ANDER LICHT

Als u uw eerste BASIC-spel schrijft, hebt u een manier nodig om te registreren of er verscheidene toetsen zijn ingedrukt. Gelukkig hebben we hiervoor een BASIC-functie. INKEY(X) (heel goed opgemerkt: zonder het '$'-teken) geeft aan of de toets X is ingedrukt. Omdat deze functie onafhankelijk van de buffer werkt, wordt niet het eerste teken in de buffer opgeslagen, maar wordt er slechts een elektrisch toetsencontact onderzocht. Als voorbeeld voor deze toepassing volgt de volgende listing:

```
10 CLS
20 IF INKEY(69) = 0 THEN LOCATE 1,5:PRINT "TOETS A INGEDRUKT"
30 IF INKEY(36) = 0 THEN LOCATE 1,10:PRINT "TOETS L INGEDRUKT"
40 LOCATE 1,5:PRINT SPACE$(20):REM REGEL 5 UITWISSEN
50 LOCATE 1,10:PRINT SPACE$(20):REM REGEL 10 UITWISSEN
60 GOTO 20
```

Als u dit programma start en dan de toetsen A en L tegelijkertijd indrukt, verschijnen beide mededelingen op het beeldscherm. Met de overeenkomstige programmering met INKEY$ zou dat niet mogelijk zijn, omdat daarmee slechts één teken kan worden gehaald.

Ik wil hier nog een kleine truc toepassen. In veel programma's worden er speciale lussen gebruikt om het programma op te houden tot er een toets is ingedrukt. Een dergelijke lus ziet er meestal zo uit:

```
WHILE INKEY$ = " ":WEND
```

Ditzelfde kan ook met een machineroutine in ROM worden gedaan door CALL &BB18 te gebruiken.

Samenvatting: onderzoek van het toetsenbord

Het commando INKEY kan worden gebruikt om te zien of er meer toetsen tegelijk worden ingedrukt.
CALL &BB18 wacht tot er een willekeurige toets is ingedrukt.

13. INVOEREN IN DE Z-80-MACHINETAAL

In diverse publikaties vindt u steeds weer programma's om in te tikken. Heel vaak zijn deze programma's in machinetaal geschreven, een taal die door de nieuweling als zeer vreemd wordt ervaren. Toegegeven, de machinetaal is niet zo gemakkelijk te leren als BASIC, maar daar staat tegenover dat zij veel sneller is en de ervaren programmeur veel meer mogelijkheden biedt. Daarom wil ik hier de beginselen van het machineachtige programmeren uiteenzetten. Na het bestuderen van dit hoofdstuk bent u dan in staat de grondbeginselen van machinetaalprogramma's te begrijpen en kunt u zelf uitmaken of u zich in deze taal wat verder wilt verdiepen. Als deze machinetaal u niet bevalt, is dat ook geen probleem. De opgedane kennis is ook bij het toepassen van andere taken zeer nuttig en ten slotte is PASCAL of LOGO ook geen slechte manier om uw computer iets bij te brengen.

13.1. WAT IS 'MACHINETAAL' EIGENLIJK?

Zoals u zeker al weet, is de machinetaal de enige mogelijkheid om rechtstreeks de computer te programmeren zonder daarbij de interpreter of de compiler in te schakelen. Daarom is het met deze taal ook mogelijk om immens hoge snelheden te bereiken.

De machinetaal omvat verschillende commando's waarmee men alle complexe bewerkingen van BASIC of van andere talen kan opbouwen. Men kan de machinecommando's grofweg in drie groepen indelen. Voor de BASIC-programmeur het gemakkelijkst te begrijpen zijn de sprongopdrachten, waarbij het programma vergelijkbaar met GOTO en GOSUB door het geheugen heen kan springen. Met andere commando's zijn de gegevens te manipuleren, zoals optellingen en verbindingen. De laatste groep bevat de bewerkingen waarmee gegevens van de ene plaats van het geheugen naar een andere plaats worden gebracht.

In beginsel geldt voor alle processors dat ze geen variabelen kennen. Hij kent alleen de normale geheugencellen en interne registers. Voor het onderscheid tussen de gegevens en de programmabytes moet de programmeur zelf zorgen. In het algemeen kunnen manipulaties met de gegevens slechts in het interne register worden uitgevoerd.

Een machinetaalcommando bestaat altijd uit een zogenoemde 'operatie-code' (of opcode) waarmee zogezegd het 'nummer' van het commando wordt aangegeven. Deze opcode kan tot drie bytes omvatten. Bovendien kunnen op deze code nog twee bytes met gegevens volgen. Zuiver theoretisch kunnen de Z-80-comando's tot 5 bytes lang zijn. In de praktijk zijn ze echter niet langer dan 4, omdat de 3-bytes-commando's alleen gebruikt worden bij een gegeven dat slechts 1 byte lang is.

13.2. DE FREQUENTIE

Alle onderdelen van een computer worden gestuurd door een kleine kwarts, dat een frequentie heeft van 4Mhz (= 4000 000 hertz; dat zijn 4 miljoen trillingen per seconde). Dit is nodig om de verschillende IC's te synchroniseren. Zou dat niet worden gedaan, dan kan het gebeuren dat bijv. het geheugen informatie naar de processor stuurt, terwijl deze nog niet klaarstaat om ze te ontvangen. Al is een microprocessor nòg zo snel, hij heeft toch enige tijd nodig om de gegevens te verwerken.

13.3. DE OPBOUW VAN DE Z-80

Elke microprocessor heeft interne registers waarmee de bewerkingen worden uitgevoerd. Het belangrijkste register is de zogenoemde 'accumulator', waarin de meeste rekenkundige en logische verbindingen tot stand komen. De accumulator (afgekort tot accu of tot A) is een 8-bits register en daarom kan hij slechts 1 byte opnemen en bewerken (eigenlijk bewerkt de accu zelf niets en worden de uitkomsten slechts in dit register opgeslagen). De meeste rekencommando's hebben twee operanden nodig (bijv. het optellen van twee getallen). De eerste operand staat voor het uitvoeren van het commando al in de accu, de tweede komt uit een ander register in de processor of uit het geheugen. Na de optelling wordt het resultaat weer in de accu opgeslagen.

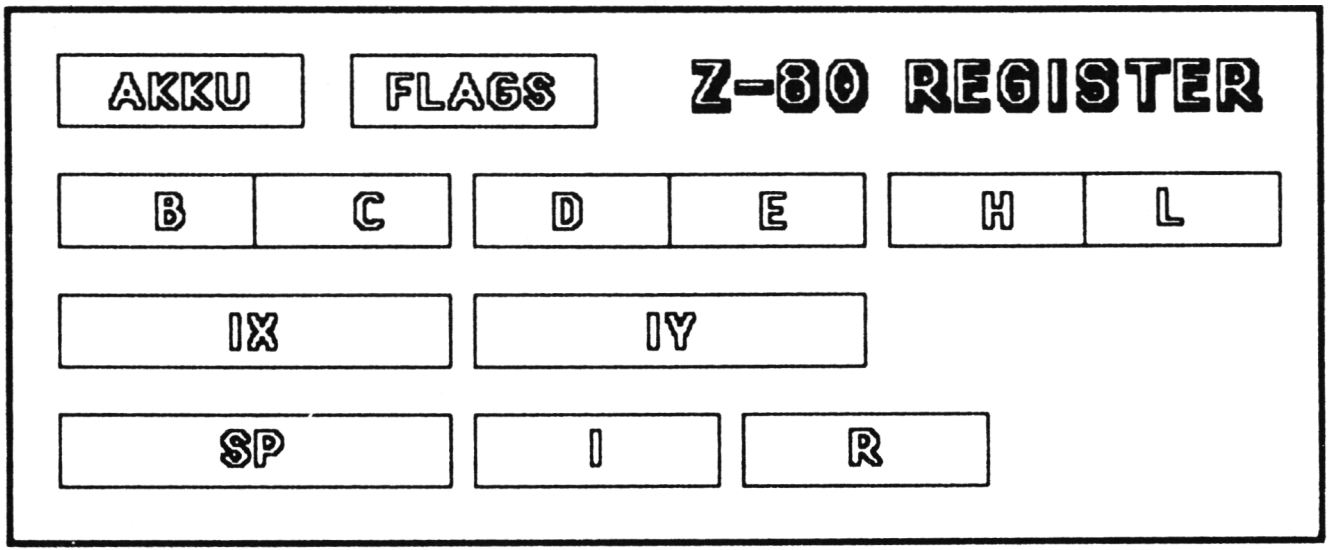
Een ander register (het Flag-register, kortweg F-register of F) bewaart de verschillende flags waarmee bepaalde toestanden van de processor kunnen worden weergegeven. Aan de hand van deze flags kan bijvoorbeeld worden vastgesteld of de inhoud van de accu 0 is.

Maar dat zijn nog niet alle registers. Er zijn nog 6 andere 8-bits-register met een bijzondere eigenschap. Steeds twee van deze geheugenplaatsen vormen samen een 16-bits-register. Het paar HL werkt daarbij als een 16-bits-accu, dat wil zeggen: hij doet hetzelfde als een normale accu, maar nu met 16 bis in plaats van met 8. Hiermee wordt het verwerken van grote getallen vereenvoudigd.

Andere registerparen zijn BC en DE. Dat is echter nog niet alles. IX en IY zijn twee indexregisters (elk 16-bits) die als wijzers naar bepaalde cellen in het geheugen wijzen. Met behulp van deze wijzers kan men op een gemakkelijke manier met hele groepen gegevens manipuleren. Hoe dat werkt, zal ik later nog wel uiteenzetten.

Het 16-bits-registers SP heeft een speciale opdracht. Hij wist altijd naar het bovenste element van de stack (SP betekent dan ook Stack Pointer). Iedere keer als er iets op de stack wordt geplaatst of eraf wordt genomen, aktualiseert de Z-80 deze wijzer zodat hij naar de nieuwe positie wijst.

Ten slotte zijn er dan nog de registers I en R, die bestemd zijn voor speciale taken (hardware-besturing).



afb. 9

Alsof dat nog niet genoeg is, bestaat er bij elk van de registers A tot L nog een tweede register, dat met het oorspronkelijke register kan worden verwisseld. Er kan echter slechts met een van deze registers tegelijk worden gewerkt. Derhalve worden deze tweede registers meestal als klein tussengeheugen gebruikt. In de commando's kan men deze reserveregisters herkennen aan een apostrof.

Nu kent u reeds de registers van de Z-80 die voor de machinetaal gebruikt kunnen worden. In het volgende zal ik het verloop van een enkele machine-opdracht in de processor uitleggen.

13.4. HET FUNCTIONEREN VAN DE Z-80

Laten we eens aannemen dat er in het geheugen van uw computer een machine-programma staat, dat er alleen maar op wacht om uitgevoerd te worden. Natuurlijk moet de microprocessor dan weten waar dat programma eigenlijk staat. Daarvoor is er een 16-bits-register die in de Z-80 de programma-teller wordt genoemd (Engels: Program Counter = PC). Daarin is het adres van het commando opgeslagen dat als eerste moet worden uitgevoerd. Als deze opdracht wordt gegeven, haalt de processor de byte uit het aangewezen geheugenadres. Dit byte wordt door de processor vast-gehouden en de programmateller wordt met de waarde 1 verhoogd, waarmee dan het volgende adres te gebruiken is. Tegelijkertijd wordt dan de opcode (dat is namelijk de betekenis van dit eerste byte) gedecodeerd, dat wil zeggen dat de Z-80 vaststelt welke van de vele commando's er in het geheugen staat. Sommige commando's hebben een opcode die meer dan een byte lang is (anders kunnen er slechts 256 commando's worden herkend). In dat geval worden de benodigde bytes eenvoudig achter elkaar uit het geheugen gehaald (de PC wijst immers altijd naar de actuele geheugencel omdat deze na elke bewerking met de waarde 1 wordt verhoogd).Ook kan het gebeuren dat er, na de een of twee bytes, gegevens volgen die dan ook moeten worden ingelezen. Die worden dan echter niet direct gedecodeerd, maar worden in een bepaald register gezet (waarmee het commando al een einde kan hebben gevonden), of ze worden voor het verwerken op een bepaalde plaats in het geheugen klaargezet.

Moeten de gegevens nog ergens veranderd worden (bijv. door op te tellen of iets dergelijks) dan wordt deze bewerking nu uitgevoerd en de uitkomst ervan wordt weer opgeslagen (bijv. in de accu). Daarmee is het commando uitgevoerd en kan men met het volgende beginnen.

Al deze gebeurtenissen hebben een zekere tijd nodig om uitgevoerd te worden, evenals bijv. elektrische stroom. In het algemeen heeft elke stap, zoals: 'haal de opcode', een bepaalde tijd nodig. Deze tijd noemt men een 'machinecyclus'. Complexere gebeurtenissen hebben meer dan een machinecyclus nodig.

Samenvatting: de PC wijst altijd naar de geheugencel waar de volgende te bewerken byte staat. Achter elkaar worden de opcodes en de gegevens ingelezen. De opcode wordt dan gedecodeerd en het commando vervolgens uitgevoerd.

13.5. HET HEXADECIMALE SYSTEEM

Als u zich met de machinetaal bezighoudt, zult u ook de schrijfwijze van hexadecimale getallen moeten kennen. Dit systeem bestaat, in tegenstelling tot onze decimale getallen, uit 16 cijfers (0-9 en A-F voor de waarden 10 tot 15). Het wordt zo vaak gebruikt omdat de omzetting naar binaire getallen heel gemakkelijk is. Een ander voordeel van het hexadecimale getallensysteem is, dat een hexcijfer precies een halve byte is (het grootste tweecijferige hexadecimale getal FF is hetzelfde als het binaire getal 1111 1111, de grootst mogelijke inhoud van een byte). Ment neemt dan soms een halve byte en vertaalt dit naar een hexgetal. De tabel laat de waarden zien van de decimale, hexadecimale en binaire getallen:

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

Bij het omzetten van hexadecimale in decimale getallen worden allereerst de cijfers afzonderlijk omgezet in het decimale equivalent. Het meest rechts staande cijfer wordt dan met 16 0 = 1, het tweede met 16 1, het derde met 16 3 = 256, enz., vermenigvuldigd. De uitkomsten worden dan alle opgeteld. Een voorbeeld:

ABCD16 (dus de waarde 10, 11, 12 en 13)
= 10*16 3 + 11*16 2 + 12*16 1 + 13*16 0
= 10*4096 + 11*256 + 12*16 + 13*1
= 43981

De byte 1010 1011 wordt dus het hexadecimale getal AB16 omdat 1010 = A en 1011 = B. Dit werkt natuurlijk ook de andere kant op.
Voor het omrekenen in de andere richting (decimaal naar hexadecimaal) kunt u het decimale getal door 16 delen en de rest van deze deling als hexadecimaal getal noteren. De uitkomst wordt weer door 16 gedeeld, enz., tot de waarde 0 wordt. Ook hiervan een voorbeeld:

53000/16 = 3312, rest 8 =» 8
3312/16 = 207, rest 0 =» 0
207/16 = 12, rest 15 =» F
12/16 = 0, rest 12 =» C

= 5300010 = CF0816

Intussen zijn er al zakrekenmachines die een aparte functies hebben voor deze omzetting. Een goede assembler of hexmonitor biedt deze mogelijkheid ook.

Ook uw CPC heeft een BASIC-functie die het mogelijk maakt om hexadecimale getallen te verwerken. Dit wordt simpel met een '&' aangegeven en het hexadecimale getal word dan direct omgezet in een decimaal getal.

13.6. BINAIR REKENEN

13.6.1. OPTELLEN

Om het maar direct bij het begin te zeggen: het binaire 'optellen' onderscheidt zich van het decimale alleen maar door het getalsysteem, maar verder gaat het precies eender.

De som van twee nullen of van een nul met een 1 (of andersom) behoeft geen toelichting, maar wordt heel gewoon opgeteld. Willen we echter twee getallen 1 (dat is dus 1 + 1) optellen, dan ontstaat er een probleem. Decimaal geeft als uitkomst 2, maar die bestaat niet in het binaire getallensysteem. Dus moet er (zoals bij het overschrijden van de waarde 9 in het decimale getallensysteem) een overdracht naar de volgende plaats worden gemaakt:

```
  0      0      1      1
+ 0    + 1    + 0    + 1

  0      1      1     10
```

Ook hele bytes zijn heel gemakkelijk met elkaar te verbinden. Hierbij wordt gewoon op elke plaats de optelling uitgevoerd (en een eventuele overdracht opgemerkt):

```
  01101101 =  109
+ 00001001 = +  9

    1  1            (overdracht)
-------------------
  01110110  = 118
```

Voor een beter overzicht zijn hier de overdrachten tevens uitgevoerd. Als het voorkomt dat er twee enen moeten worden opgeteld en er dan ook nog een overdracht bijkomt (1 + 1 + 1 = 3) is het duidelijk dat de uitkomst 1 1 is.

Probeert u zelf eens de volgende optelling uit te voeren:

```
  10010011
+ 11011111
1    11111    (overdrachten)

  101110010
```

Nu bestaat de uitkomst plotseling uit 9 bits! Het negende bit heet een overdrachts' of 'Carry'-bit. Hierdoor wordt aangegeven dat de optelling van twee 8-bits-getallen het toegestane gebied van een byte (0-255) heeft overschreden, waarmee we dan bij de 16-bit-optelling zijn aangekomen. Er is geen enkele computer die met een 8-bits-register de getallen kan weergeven, omdat de getallen meestal veel groter zijn. Het is wel een feit dat een 8-bits-microprocessor (zoals bijv. de 6510) slechts acht bits tegelijk kan verwerken. Bestaat een getal bijvoorbeeld uit twee bytes, dan moeten deze afzonderlijk worden opgeteld. Omdat, op de overdracht na, de beide delen van die twee getallen volledig onafhankelijk van elkaar worden opgeteld, heeft men alleen het carry-bit nodig om ook grotere getallen te verwerken. Het heeft tot taak om de overdracht van de laatste plaats van de eerste byte naar de eerste plaats van de tweede byte door te schuiven. Een voorbeeld:

```
   00110101  10010011
+  10011011  11011111

    1111111      11111 (overdrachten)
   ---------------------------
   11010001  01110010
```

Het rechterdeel van deze optelling kent u al uit het vorige voorbeeld.

13.6.2 AFTREKKEN

Als een computer een getal van een ander getal wil aftrekken, vormt hij eerst het negatieve equivalent van dat getal (vermenigvuldigd met -1) en telt het er dan bij. Dat wordt zo gedaan omdat een optelling en een negatie met de electronische bouwstenen (zoals AND, OR, XOR en NOT) kan worden samengesteld, maar een aftrekking niet.

Om een negatief getal te bereiken, wordt het getallenbereik van een byte van 0 tot 255 verschoven naar -127 tot + 127. Het bit met de hoogste waarde (bit 7) dient dan als voorteken. Is deze waarde 1, dan spreken we van een negatief getal, bij 0 wordt deze byte positief. Bij het negatief maken van een getal kunnen we echter niet alleen maar een voorteken plaatsen. Een voorbeeld zal onze problemen duidelijk maken:

```
   00000001
+  10000001
-------------------
   10000010
```

Naar het decimale getallensysteem overgezet, zou dit betekenen dat + 1 + (-1) = -2 is. Daarom wordt er een andere weg ingeslagen. Een byte kan door het vormen van een zogenoemd twee-complement heel gemakkelijk met -1 worden vermenigdvuldigd. Daarvoor worden alle bits geïnverteerd en met 1 vermeerderd.

```
Voorbeeld:        01011011
Inverteren:       10100100
             +           1
             ---------------
                  10100101
```

Als we volgens dit schema het verschillen tussen 1 en 1 (dus 1-1) berekenen, krijgen we het juiste antwoord:

```
   00000001
+  11111111

   11111111 (overdrachten)
   ----------
   100000000
```

Zoals u ziet, ontstaat er schijnbaar een overdracht. Maar ook hier gedraagt een aftrekking zich anders. We kunnen dat hier gewoon weglaten. Willen we 16-bit aftrekken, dan zal het nu overbodige carry-bit ervoor zorgen dat de plaatsen van het tweede byte ook op nul gezet worden. Dat is belangrijk, omdat bij negatieve 16-bits-getallen alle 16 plaatsen worden geïnverteerd. Het getal -1 ziet er als twee-byte-getal als volgt uit: 11111111 11111111. Zou het carry-bit nu ontbreken, dan zou de uitkomst zijn: 11111111 00000000. En dat is fout!

Gelukkig is het programmeren van een aftrekking niet zo gecompliceerd. Het aftrekkingscommando van de Z-80 omvat reeds het vormen van een twee-complement.

13.6.3. VERMENIGVULDIGEN

U wilt het wellicht niet geloven, maar de Z-80-machinetaal heeft slechts twee rekenbewerkingen, en wel die voor het optellen en voor het aftrekken. Alle andere rekenroutines worden samengesteld uit deze twee bewerkingen, en meestal als subroutine.

Omdat we niet alle bijzonderheden van de machinetaal zullen behandelen (daartoe zijn er betere en uitvoeriger boeken) behandel ik alleen het eenvoudigste algoritme van de vermenigvuldiging. Deze wordt door de vakmensen niet graag gebruikt omdat het niet efficiënt is. Maar nu ter zake. Om het produkt van x * n te berekenen, is het voldoende om de waarde x n-maal op te tellen. Dat werkt natuurlijk alleen maar bij gehele getallen. Bij decimale breuken is die handelwijze gecompliceerder, waarbij getallen bijv. plaats voor plaats en niet als heel getal met elkaar worden verbonden, wat echter in principe op dezelfde manier werkt. Voor een beter begrip nog een voorbeeld:

4 * 3 = 4 + 4 4= 12

13.6.4. DELEN

Ook voor het delen bestaat er een gemakkelijke methode. Ook x door n te delen, wordt gewoon voortdurend n van x afgetrokken. Het aantal mogelijke aftrekkingen, tot n groter is dan x, is de uitkomst van deze deling. Hier een voorbeeld:

```
10/3 = ?
10-3 = 7    telregister = 1
 7-3 = 4    telregister = 2
 4-3 = 1    telregister = 3

   = 10/3 = 3,rest 1
```

Deze methoden zijn mogelijk omdat de machinetaal zo onnoemelijk snel is. Overigens werkt een zakrekenmachine op dezelfde manier. Iedere keer als u een rekentoets indrukt, gaat er een klein machineprogramma lopen (uiteraard met het bijbehorende algoritme).

Met behulp van de vier rekenmethoden zijn er ook hogere functies samen te stellen (zoals machtsverheffen, sinus, e.a.). Op deze manier kan elke wiskundige bewerking met AND, OR, XOR en NOT worden verkregen (want ook optellen en aftrekken zijn zo te construeren).

13.7. HOE WERKT EEN VERGELIJKING?

In BASIC stellen vergelijkingen niet veel voor. Maar hoe kan men die in machinetaal verkrijgen? Bekijk daarvoor eerst eens het volgende voorbeeld:

A = B (=) A - B = 0

Zoals u ziet, kan men een vergelijking tussen twee getallen (hier zijn dat A en B) heel simpel veranderen. Voor de computer heeft deze vorm het voordeel dat er aan de rechterzijde van deze vergelijking een 0 staat. Het getal 0 is het enige getal waarvan de microprocessor kan vaststellen of dit al dan niet in het rekenregister (meestal de accu) staat. Daarvoor worden gewoon alle bits op de volgende manier OR met elkaar verbonden:

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

Als alle bits van de accu de waarde 0 hebben, is de uitkomst van deze verbindingsketen ook een 0, in alle andere gevallen (d.w.z. als er ten minste 1 bit de waarde 1 heeft) is de uitkomst 1. En zo kan de microprocessor vaststellen of het rekenregister (waar immers de uitkomst van de laatste bewerking staat) gelijk of ongelijk aan 0 is. Voilà, de eerste twee vergelijkingen zijn onderzocht. Voor de vergelijking A = B of A B hoeven we alleen maar de twee getallen van elkaar af te trekken en dan te kijken of de inhoud van de accu de waarde 0 heeft. Dit kan door middel van de Z-flag (Z staat voor Zero). Is deze waarde 1, dan is de uitkomst van de laatste b ewerking een 0. Heeft Z de waarde 0, dan is de uitkomst van de laatste bewerking een 0. Heeft Z de waarde 0, dan is de uitkomst ongelijk aan 0. De flag is dus niet alleen maar beperkt tot de accu, maar aan de andere kant worden ook niet alle flags door elk commando veranderd. Òf dat gebeurt, kunt u in de appendix nalezen. Maar nu terug naar de vergelijkingen. Bij 'groter dan' en 'kleiner dan' gaat het op ongeveer dezelfde manier als bij 'gelijk aan'. Na de aftrekking kijken we of deze waarde groter dan wel kleiner is dan 0. Dit is te zien aan de bitwaarde van het voorteken:

A > B (=) A - B groter dan 0 (waar, als bit 7 = 0).
A < B (=) A - B kleiner dan 0 (waar, als bit 7 = 1).

Het voortekenbit wordt door vele commando's naar de S-flag gestuurd (S betekent 'Sign' = voorteken). Daar kan men het door middel van een speciale opdracht onderzoeken.

Een andere flag heet P/V (omdat we dat in dit boek nauwelijks gebruiken, noem ik het simpel en ook korter P). Dit heeft twee functies. Met de ene kan het de pariteit (even of oneven) aangeven en met de andere, of het voortekenbit is veranderd (wat dit allemaal betekent, hoeven we hier niet te weten, we willen alleen maar kennismaken met de machinetaal).

13.8. HET EERSTE PROGRAMMA

Nadat u de diverse begrippen van de machinetaalprogrammering hebt leren kennen, zullen we nu met het eenvoudigste programma beginnen. We ontwerpen een optel-programma voor twee 8-bits-getallen.

Allereerst moeten we het programma ergens meedelen welke twee getallen moeten worden opgeteld. Een soort INPUT-commando bestaat er niet in de machinetaal en daarom moeten we ons ermee behelpen deze getallen ergens in het geheugen op te slaan. Daar kan het programma dan zelf de waarden vandaan halen. Dat is dan al de eerste opdracht die ons programma moet uitvoeren. Het commando 'LD A,(nnnn)' werkt ongeveer als PEEK dat de byte van het adres nnnn naar de accumulator haalt.

We weten ook waar de tweede byte staat, maar we kunnen dat echter niet met 'LD,B,(nnnn)' in de processor laden omdat dit programma in de machinetaal niet bestaat. Het is echter mogelijk om het registerpaar HL als wijzer voor onze byte te gebruiken. Daartoe brengen we het adres door middel van LD HL,nnnn naar het gewenste register. Let erop dat de uitdrukking 'nnnn' nu niet meer tussen haakjes staat, hetgeen ons erop wijst dat we deze uitdrukking direct in HL moeten laden en niet dat het als adres voor de eigenlijke waarde staat.

Met het volgende commando moeten de twee bytes eindelijk worden opgeteld. Dit luidt 'ADD A,(HL)'. Hiermee wordt bereikt dat de waarde uit de accumulator en het byte, dat in adres HL staat, worden opgeteld. De uitkomst hiervan komt dan weer in de accu te staan. Als de som van de twee getallen groter is dan 255, wijst de Z-80 dit aan door de overdrachtsbit op 1 te zetten.

Aan een uitkomst in de accumulator hebben we natuurlijk weinig. Daarom hebben we nog een ander commando nodig, om dit resultaat in een geheugencel op te slaan, waaruit het dan door middel van PEEK kan worden opgezocht. Dit wordt gedaan met 'LD (nnnn),A'. Dit commando werkt op dezelfde manier als 'LD A,(nnnn)' maar dan in de omgekeerde richting.

Ten slotte wordt met RET dit onderprogramma beëindigd (zoals ook RETURN in BASIC). Daartoe moet u weten dat de interpreter door CALL een machineroutine op dezelfde manier behandelt als een onderprogramma; het laatste commando in een machinetaalroutine moet dan ook altijd RET zijn, omdat de computer zich anders ophangt.

We hebben er tot nu toe vrolijk op los geprogrammeerd, zonder ons af te vragen in welke geheugencellen dat allemaal plaatsvindt. In ons programma kunt u die adressen bijna willekeurig kiezen, alleen moet u zich ervan overtuigen dat deze plaatsen niet al door de computer zelf worden gebruikt (u kunt dat met MEMORY verhinderen). Mijn voorstel is dat u het BASIC-geheugen beperkt tot &AAFF. U kunt dan alle bytes van &AB00 tot &AB7F gebruiken. Ons programma zetten we vanaf &AB00 in het geheugen; de waarden die we willen optellen en de uitkomst zetten we het beste aan het einde van het (vorige) BASIC-gebied. Dan ziet ons programma er als volgt uit:

Adres	Commando	Commentaar
AB00	LD A, (AB7F)	Eerste getal uit AB7F laden.
AB03	LD HL,AB7E	Tweede getal staat in AB7E.
AB06	ADD A,(HL)	Accu en tweede getal optellen
AB07	LD (AB7D),A	Uitkomst opslaan
AB0A	RET	Einde programma.

Aan de adressen ziet u dat de commando's een verschillend aantal bytes nodig hebben. Commando's waarin een adres wordt aangewezen, hebben minstens drie bytes nodig, terwijl er andere zijn, zoals bijv. 'ADD A,(HL)', die aan één byte genoeg hebben.

Voordat u de bytes in het geheugen kunt POKEn, moet u weten hoe ze eruitzien. In de tabel met de Z-80-commando's kunt u voor elk commando opzoeken welke opcode erbij hoort. Voor 'LD A,(nnnn)' is dat 3A plus nog twee adresbytes. Daarbij mag u NOOIT vergeten dat de Lowbyte steeds voor de Highbyte van het adres staat. Het complete commando ziet er in getallen uitgedrukt zo uit:

3A 7F AB

En hier zijn dan de codes voor de rest van het programma:

21 7E AB	(LD HL,AB7E)
86	(ADD A,(HL))
32 7D AB	(LD (AB7D),A)
C9	RET

Deze waarden moeten in de bijbehorende geheugencel gePOKEd worden. Dat gebeurt met het volgende BASIC-programma:

```
10 MEMORY &AAFF'
20 FOR i = &AB00 TO &AB0A:READ a:POKE i,a:NEXT
30 DATA &3A,&7F,&AB,&7E,&AB,&86,&32,&70,&AB,&C9
```

Deze machineroutine kan met CALL &AB00 gestart worden. Voordien moet u echter eerst de twee getallen die moeten worden opgeteld door middel van POKE &AB7F,z1 en POKE &AB7E,z2 in het geheugen inlezen. Na het CALL geeft PRINT PEEK(&AB7D) de uitkomst. Opdat u op een eenvoudige wijze een aantal verschillende getallen kunt proberen, kunt u aan het vorige programma de volgende regels hangen:

```
40 INPUT''Getal 1, getal 2'';z1,z2
50 POKE &AB7F,z1:POKE &AB7E,z2
60 CALL &AB00:PRINT PEEK (&AB7D):GOTO 40
```

De aftrekking wordt op dezelfde manier geprogrammeerd, waarbij u vanzelfsprekend het commando ADD moet vervangen door SUB (HL). In de DATA-regel wordt daardoor alleen de zevende waarde &86 veranderd in &96.

13.9. HOE WORDT EEN 'LUS' GEPROGRAMMEERD?

Als u in BASIC een bepaalde gebeurtenis enige malen wilt herhalen, dan zijn er twee mogelijkheden om dit te programmeren: met FOR...NEXT en met WHILE...WEND. Een derde mogelijkheid is niet zo gemakkelijk en wordt daarom ook minder toegepast. Het is zonder meer mogelijk om aan het einde van een programmadeel een teller met de waarde 1 te verhogen en door middel van een IF te onderzoeken of er nog een keer naar het begin van dit programma-onderdeel moet worden teruggesprongen. Hoe ongemakkelijk deze laatste manier ook lijkt, het is de enige mogelijkheid die de machinetaal ons voor dit doel te bieden heeft. In plaats van een variabele gebruiken we hier een processorregister en de lengte van de lus wordt van boven naar beneden geteld. Het onderstaande programma doet niets anders dan 255 keer op een doorloop wachten (omdat dan de lus leeg is). Ik heb er geen BASIC-lader bij vermeld omdat er toch niets te zien is. De machinetaal werkt zó snel dat het programma in breukdelen van een seconde is afgelopen. Hier is dan het programma:

```
AB00    LD B,FF      Laden van de luslengte.
AB02    DEC B        B met de waarde 1 verminderen
AB03    JP NZ,AB02   Spring, als ongelijk 0.
AB06    RET          Anders einde programma.
```

Het eerste commando zet de luslengte in registrer B. Dit getal staat vast in het programma en staat dus direct achter de opcode. U zult zich misschien de vraag stellen waarom uitgerekend register B voor de teller wordt gebruikt. Wel, de accu is voor de rekenbewerkingen voorbestemd en op vergelijkbare wijze bestaan er speciale telopdrachten voor het B-register.

Het DEC-commando (decrement = met 1 verminderen) houdt in dat steeds de waarde 1 van B wordt afgetrokken. Als het resultaat hiervan 0 is, wordt de Zero-flag op 1 gezet en anders blijft de waarde 0. Deze uitkomst hebben we voor de volgende stap nodig. JP NZ springt alleen dan naar het aangegeven adres als de Z-flag op 0 staat, dus het terugspringen gebeurt alleen dan als het einde van de lus nog niet bereikt is. Staat Z op 1, dan gaat het programma verder met de volgende opdracht. Dat wil in dit geval zeggen dat het einde van het programma bereikt is.

13.10. NOG MEER REKENROUTINES

13.10.1. 16-BIT-OPTELLING

Met de 8-bit-getallen kunnen - zoals al is gezegd - slechts getallen tot 255 worden verwerkt. Bij 16-bits zijn dat al getallen tot 32767. De Z-80 behoort tot de weinige 8-bits-processors die ook commando's verstrekken voor het verwerken van 16-bits-berekeningen. Ook heeft hij de daarvoor noodzakelijke 16- bits-registers.

Voor een 16-bits-optelling moet het eerste getal in het registerpaar DE geladen worden. Met 'LD DE,(nnnn)' wordt byte nnnn in het register D gezet, E krijgt dan de waarde van de geheugencel met het adres nnnn + 1. Ook hier wordt dus het 'pointerformat' LOW-HIGH toegepast. Voor HL hebben we hetzelfde commando. Het commando voor het optellen van deze 16-bits luidt 'ADD HL,DE'. Zoals u al vermoedde, worden daarmee de twee getallen uit DE en HL opgeteld en de uitkomst wordt in HL opgeslagen. Daarvandaan wordt het door 'LD (nnnn), HL' in het geheugen gezet. Het hele programma ziet er dan als volgt uit:

```
AB00    LD DE,(AB7E)
AB04    LD HL,(AB7C)
AB07    ADD HL,DE
AB08    LD (AB7A),HL
AB0B    RET
```

Hierbij ziet u dat elke LD-opdracht twee bytes tegelijk bewerkt. Het laadprogramma luidt:

```
10 DATA &ED,&5B,&7E,&AB
11 DATA &2A,&7C,&AB
12 DATA &19
13 DATA &22,&7A,&AB
14 DATA &C9
20 FOR i = &AB00 to &AB0B: READ a: POKE i,a: NEXT
30 INPUT "Zahl 1, Zahl 2"; z1,z2
40 POKE &AB7E, z1 AND 255: POKE &AB7F, INT(z1/256)
50 POKE &AB7C, z2 AND 255: POKE &AB7D, INT(z2/256)
60 CALL &AB00: PRINT PEEK(&AB7A) + 256 * PEEK(&AB7B)
70 GOTO 30
```

Ook hierbij kunt u weer met verschillende waarden experimenteren.

13.10.2. VERMENIGVULDIGEN

Om twee getallen met elkaar te vermenigvuldigen, moet we meermaals optellen (dat hebben we al in hoofdstuk 13.6.3 vastgesteld). In BASIC wordt dit algoritme als volgt omgezet:

```
10 INPUT''getal 1, getal 2'';z1,z2:e = 0
20 FOR i = 1 to z1
30 e = e + z2:NEXT
40 PRINT''UITKOMST'';e
```

Zoals u ziet, heeft dit programma in feite slechts een lus en een optelling. Beide hebben we al in de machinetaal geprogrammeerd.

De vermenigvuldiging van twee 8-bits-getallen heeft als uitkomst een 16-bits-getal. Daarom is het aan te bevelen dat we hierbij een 16-bits-optelroutine gebruiken. In de lus moet alleen de optelroutine staan. Alle andere commando's zijn slechts nodig voor de voorinstelling van de registers en voor het opslaan van de uitkomst. Met de eerste opdracht (zie de listing) wordt het getal 1 in de accu geladen. Dit moet tenslotte in register B terechtkomen, maar omdat B niet direct uit het geheugen kan worden gelezen, halen we dat getal eerst in de accu om het dan met een tweede opdracht naar B te brengen. Hiermee is dan de lengte van de lus bepaald. Het andere 8-bits-getal wordt steeds weer met het register HL opgeteld. Daarom komt ook dit (ook via een omweg) in het register E terecht. Omdat ADD HL,DE ook altijd het D-register mede verhoogd, moet dit steeds weer op nul worden gezet. Ook HL moet bij het begin van de lus de waarde nul hebben. Dat wordt met de volgende twee commando's gedaan.

Daarna begint de lus met het ADD-commando. Omdat het registerpaar HL alleen door de optelling wordt veranderd, staat hierin steeds de laatste tussenuitkomst. De rest van het machineprogramma kunt u waarschijnlijk zelf verklaren. DEC B en JP NZ,ABOD geven het einde van de lus aan, LD (AB7C),HL zet de uitkomst in het geheugen en RET beëindigt het programma. De listing luidt:

```
AB00 LD A, (AB7F)      getal 1
AB03 LD B,A            naar B
AB04 LD , (AB7E)       getal 2
AB07 LD E,A            naar E
AB08 LD D,00           D wissen
AB0A LD HL,0000        HL wissen
ABOD ADD HL,DE         optellen
ABOE DEC B             B verminderen
ABOF JP NZ, ABOD       herhaal, als  0
AB12 LD (AB7C), HL     resultaat opslaan
AB15 REt               terug naar BASIC
```

Natuurlijk is er ook hiervoor een BASIC laadprogramma

```
10 DATA &3A,&7F,&AB
11 DATA &47
12 DATA &3a,&7E,&AB
13 DATA &5F
14 DATA &16,&00
15 DATA &21,&00,&00
16 DATA &19,&05
17 DATA &C2,&0D,&AB
18 DATA &22,&7C,&AB
19 DATA &C9
20 FOR i= &AB00 TO &AB15: READ a: POKE i,a: NEXT
30 INPUT "Zahl 1, Zahl 2"; z1,z2
40 POKE &AB7F, z1: POKE &AB7E, z2
50 CALL &AB00: PRINT PEEK(&AB7C)+256*PEEK(&AB7D)
60 GOTO 30
```

13.11 NUTTIGE MACHINEROUTINES.

In dit gedeelte wil ik het nuttige met het aangename verenigen en u een aantal machineroutines voorstellen die niet alleen slechts nuttig zijn bij het leren van de machinetaal, maar ook nog een praktisch nut hebben.

Beginnen we met een stukje destructie, het uitwissen van het hele geheugenbereik. In BASIC kan dat met een FOR...NEXT-lus en een POKE-opdracht worden bereikt, maar dat duurt heel lang. Het kan natuurlijk ook met de veel snellere machinetaal worden toegepast. Hier geldt evenwel een beperking, want er kunnen slechts 256 types per keer worden uitgewist.

Vergelijkbaar met de SAVE-opdracht voor bepaalde geheugendelen moeten in ons programma het startadres en het aantal uit te wissen bytes staan. HL dient als wijzer voor de byte, die uitgewist moet worden en wordt na iedere lusdoorgang verhoogd met de waarde 1. In het register B staat weer op de bekende manier de lengte van de lus (en eveneens het aantal bytes dat uitgewist moet worden). In de lus wordt allereerst de byte, die door HL wordt aangewezen met de waarde nul geladen. Dan worden de beide registers actueel gemaakt en, als het nodig is, springt JP NZ,(nnnn) weer terug naar het begin van de lus.

Hiermee is ons programma eigenlijk al klaar, maar we voegen er nog enige luxe aan toe. Aan het einde van de lus staat in HL het adres dat als volgende uitgewist moet worden. Dit adres wordt opnieuw in het geheugen gezet en wel op de plaats, waar het eerste commando het beginadres afhaalt. Als we nu verder willen uitwissen, hoeven we niet moeizaam een nieuw startadres in te POKEn, maar is een CALL-commando al voldoende.

Iets dergelijks kunt u ook voor de luslengte toepassen. Omdat de geheugencel door dit programma niet wordt veranderd en derhalve de waarde die erin staat evenmin, is ook hier geen POKE-commando meer nodig. Wilt u dus 100 bytes uitwissen in stappen van 10, dan kunt u het volgende programma gebruiken:

```
POKE &AB7F,10:POKE &AB7D,LOWBYTE:POKE &AB7E,HIGHBYTE
FOR i = 1 TO 10:CALL &AB00:NEXT
```

Dit is weliswaar niet erg zinvol, want dat is met een enkele aanroep en een luslengte van 100 beter op te lossen. Maar als u nog meer commando's aan deze FOR...NEXT-lus wilt toevoegen (bijv. wachten op het indrukken van een toets.) is het toch mogelijk om onder andere in stappen het beeldschermgeheugen uit te wissen. Bovendien kan men op deze manier lange geheugenblokken uitwissen.

Over de lengte van de lus is nog iets op te merken. Als een routine met een luslengte van 0 wordt opgeroepen, dan worden er 256 bytes uitgewist omdat voor de eerste JP (dus voor het eerste mogelijke einde) B met de waarde 1 wordt verminderd. 0 min 1 is voor de Z-80 (en voor alle andere microprocessors) 255 (bekijk hiertoe nog eens de binaire getallen en tel dan 255 en 1 op). Daarom is deze lus nog niet ten einde en worden er nog eens 255 uitgewist. Het programma ziet er dan als volgt uit:

```
AB00 LD HL,(AB7D)
AB03 LD A,(AB7F)
AB06 LD B,A
AB07 LD (HL),0
AB09 INC HL
AB0A DEC B
AB0B JPNZ,AB07
AB0E LD (AB7D),HL
AB11 RET
```

Het bijbehorende BASIC-laadprogramma luidt:

```
10 DATA &2A,&7D,&AB
11 DATA &3A,&7F,&AB
12 DATA &47
13 DATA &36,&00
14 DATA &23
15 DATA &05
16 DATA &C2,&07,&AB
17 DATA &22,&7D,&AB
18 DATA &C9
20 FOR i = &AB00 TO &AB11: READ a: POKE i,a: NEXT
30 INPUT "Startadresse";a
40 POKE &AB7D, a-INT(a/256): POKE &AB7E, INT(a/256)
50 INPUT "Länge";b
60 POKE &AB7F, b
70 CALL &AB00: GOTO 30
```

Een tweede routine kan gebruikt worden om geheugenblokken te kopiëren. Omdat we hierbij een speciaal Z-80-commando kunnen gebruiken, is dit programma korter dan het vorige en de beperking tot 256 bytes vervalt.

Dit speciale programma heet LDIR. Het kopiëert zonder dat wij hoeven in te grijpen in hele geheugenblokken van willekeurige lengte. Deze lengte moeten we eerst in het registerpaar BC opslaan. Bovendien moet de Z-80 nog het startadres weten van het deel dat we willen kopiëren alsmede het doeladres. Het startadres komt in HL en het doeladres in DE te staan. Dan kan het commando LDIR beginnen. Het kopiëert de byte die in adres HL staat naar het adres dat zich in DE bevindt. Daarna wordt zowel HL als DE met de waarde 1 vermeerderd en wijzen ze dus de volgende byte aan. Bovendien wordt het register BC met 1 verminderd. Zolang BC nog ongelijk is aan 0, wordt LDIR automatisch herhaald. Er is dus slechts één commando nodig voor deze hele lus!

De rest hiervan heeft geen toelichting nodig. Dit programma werd overigens voor een ander deel van het geheugen geschreven. Daarom wordt het niet met de eigen uitwis-routine behandeld en kan het gelijktijdig hiermee worden gebruikt.

```
AB 20 LD HL,(AB6E)
AB23 LD DE,(AB6C)
AB27 LD BC,(AB6A)
AB2B LDIR
AB2D RET
```

```
10 DATA &2A,&6E,&AB
11 DATA &ED,&5B,&6C,&AB
12 DATA &ED,&4B,&6A,&AB
13 DATA &ED,&B0
14 DATA &C9
20 FOR i = &AB20 TO &AB2D: READ a: POKE i,a: NEXT
30 INPUT "Quellbereich";a
40 POKE &AB6E, a-INT(a/256): POKE &AB6F, INT(a/256)
50 INPUT "Zielbereich";b
60 POKE &AB6C, b-INT(b/256): POKE &AB6D, INT(b/256)
70 INPUT "Länge";c
80 POKE &AB6A, c-INT(c/256): POKE &AB6B, INT(c/256)
90 CALL &AB20: GOTO 30
```

13.12. DE ADRESSERINGSMOGELIJKHEDEN

Als u de lijst met commando's eenmaal hebt bekeken, zal het u zeker opgevallen zijn dat de operand niet alleen bestaat uit registers zoals A,B,C enz., maar dat er ook haakjes en dergelijke worden gebruikt. Bij een opdracht zoals ADD A,B is het duidelijk wat er gebeurt; hierbij worden de registers A(ccu) en B opgeteld. Maar wat betekent ADD A,(HL)?

In het algemeen geldt dat een uitdrukking die tussen haakjes staat het adres van een byte in het geheugen voorstelt en niet de operand zelf; 'nn' betekent dus 'inhoud van de geheugenplaats nnnn' (in deze verkorte schrijfwijze stelt een letter een byte voor; nn staat dus voor 2 bytes = 16 bits = 4 hexgetallen). Deze methode noemt men wel de 'absolute adressering'. Als de haakjes ontbreken, wordt deze waarde direct als operand overgenomen. ADD A,n telt dus de byte nn (die direct na de opcode in het geheugen staat, als een soort constante) bij de inhoud van de accu op.
Deze adresseringmethode heet 'direct'.

Weer een andere manier is de 'indirecte adressering'. Met indirect wordt hier bedoeld dat de operand niet het adres aanwijst van het gegeven dat moet worden verwerkt, maar de plaats waar dit staat. Het commando ADD A,(HL) werkt dus op zo'n manier dat de processor eerst 'nakijkt' welke waarde er in het register HL staat opgeslagen. Deze waarde dient als wijzer voor de geheugencel waar de byte staat opgeslagen, die dan tenslotte bij de accu wordt opgeteld. 'Dat is ingewikkeld,' zult u zeggen. 'Dat is slim,' zeggen de ontwerpers van de Z-80. Door middel van deze indirecte adressering kan men namelijk hele blokken bytes als een array opzoeken.

Deze methode heeft nog een variant. Dat is de 'indirecte-geïndexeerde' adressering. De operand '(IX + d)' betekent in dit geval dat bij de waarde van het register IX nog de constante dd moet worden opgeteld en dat pas de uitkomst hiervan het uiteindelijke adres is.

De laatste in deze lijst is de 'relatieve adressering', die men alleen bij bepaalde typen sprongen kan vinden. Vergelijkbaar met de geïndexeerde adressering volgt op de opcode nog een constante. Deze wordt bij de programmateller opgeteld. Als nu bit 7 van de bytes de waarde 1 heeft, wordt hij als een negatief getal behandeld (wat overigens ook voor de indirect-geïndexeerde adressering geldt). Het resultaat is dan een sprong terug. Omdat de programmateller na het inlezen van de constante al naar het volgende commando wijst, kan men maximaal 129 stappen naar voren en maximaal 127 stappen naar achteren springen.

13.13. DE COMMANDO'S VAN DE Z-80

Als u de lijst met de Z-80-opcodes bekijkt (waarop alle commando's met hun codes en de bijbehorende flags staan) dan zult u zien dat er verschillende groepen commando's zijn waarvan het verschil alleen in de wijze van adresseren zit. Deze groepen worden hierna beschreven. U moet niet alle omschrijvingen uit het hoofd leren. Het is alleen maar de bedoeling om een overzicht te krijgen van de vele mogelijkheden van de machinetaal. Als u later in de machinetaal gaat programmeren, kunt u de opcodetabel als naslagwerk gebruiken.

Omdat de commando's voor de diverse operanden op dezelfde manier werken, worden de laatste in de omschrijving door letters (bijv. x, n, enz.) vervangen.

ADC A,X
De operand x wordt bij de accu opgeteld. Hierbij wordt ook acht geslagen op een eventuele overdracht naar de carry-bit. De uitkomst hiervan wordt weer in de accu opgeslagen. Na de uitvoering worden de flags overeenkomstig de inhoud van de accu gezet.

ADC HL,X
Dit commando werkt als het vorige, alleen is het een 16-bits-optelling. Daartoe worden de operand (dat is nu een 16-bits-register) en het carry-bit bij het registerpaar HL opgeteld. De uitkomst komt in HL te staan en de flags worden overeenkomstig opgeslagen.

ADD A,X
De operand wordt toegevoegd aan de inhoud van de accu en de uitkomst wordt daar weer opgeslagen. Er wordt geen rekening gehouden met de carry-bit, maar na de optelling zijn de flags overeenkomstig geplaatst.

ADD HL,X
Zoals ADC HL,X, de carry-bit wordt bij de optelling evenwel niet meegenomen.

ADD IX,X
De operand wordt opgeteld bij het indexregister, de uitkomst wordt daar opgeslagen en het carry-bit wordt geactualiseerd. De andere flags worden echter niet veranderd.

ADD IY,X
Als ADD IX,X, maar dan voor het indexregister IY.

AND X
De operand en de accu worden AND-verbonden, de uitkomst wordt in de accu opgeslagen. Afhankelijk van de uitkomsten worden de flags geactualiseerd. De carry-bit wordt 0 (AND levert geen overdracht).

BIT N,X

De bit n van de operand wordt getest, dat wil zeggen, hij wordt gecomplementeerd en overgedragen aan de Z-flag. Had de gekozen bit de waarde 1, dan wordt Z = 0, stond de bit op 0, dan wordt Z = 1. De flags S en P bezitten toevallige waarden.

CALL NN

Het onderprogramma vanaf adres nnnn wordt aangeroepen (vergelijkbaar met GOSUB).

CALL-VOORWAARDE, NN

Het onderprogramma wordt slechts dan aangesprongen als aan de voorwaarde is voldaan. Als voorwaarde kan elke van de vier flags S, Z, P en C(arry) getest worden. Afhankelijk van de toestand wordt er dan gesprongen of met de volgende opdracht in het programma verdergegaan.

CCF

Complementeert het carry-bit (1 = »0 0 = »1).

CP X

De operand x wordt vergeleken met de accu; dat wil zeggen, ze worden van elkaar afgetrokken, maar de uitkomst wordt niet opgeslagen, zodat de inhoud van de accu onveranderd blijft. Als de operand kleiner is dan de accu, wordt S = 1, is hij groter of gelijk, dan wordt S = 0. Als beide waarden gelijk zijn, wordt Z = 1 en bij een ongelijkheid wordt Z = 0.

CPD

HL dient als wijzer op een geheugenplaats, die met de accu wordt vergeleken. Als beide waarden gelijk zijn, dan is Z = 1, als de accu gelijk of groter is, dan is S = 0; bij S = 1 was de accu kleiner dan de byte uit het geheugen. Dan worden HL en BC met 1 verlaagd. Is de inhoud van de BC = 0, dan wordt P-flag op 0 gezet, anders op 1. Daarom kan BC als byte-teller worden gebruikt.

CPDR

Zoals CPD; de uitvoering wordt echter automatisch zo lang herhaald tot er een gelijkheid wordt vastgesteld (Z = 1) of BC = 0 (P-flag op 0).

CPI

Zoals CPD; HL wordt dan echter niet verlaagd maar verhoogd (met 1).

CPIR

Zoals CPDR; HL wordt daarbij echter verhoogd.

CPL

De bits van de accu worden geïnverteerd (0 wordt 1 en omgekeerd).

DAA

Na een rekenkundige bewerking (ADC, ADD, DEC, INC, NEG, SBC, SUB) worden de eventueel aanwezige verkeerde BCD-cijfers in de accu en de flags gecorrigeerd.

DEC X

De genoemde operanden worden met de waarde 1 verminderd. Behalve bij de registers BC, DE, HL, IX, IY en SP worden dan ook nog de flags van deze uitkomsten gezet.

DI

Na het uitvoeren van deze opdracht laat de Z-80 alle volgende interrupts achterwege.

DJNZ E

B wordt met 1 verminderd. Als de inhoud van B ongelijk is aan 0, wordt de relatieve sprongafstand E bij het adres van de volgende opdracht opgeteld, waarna het programma dan bij het zo ontstane nieuwe adres wordt voortgezet.

EI

Na dit commando worden de interrupts weer vrijgegeven, dat wil zeggen, dat na het aanroepen van een interrupt de processor naar een speciale routine wordt gestuurd.

EX X, Y

De twee aangegeven operanden worden met elkaar verwisseld. X kan ook voor (SP) staan, dan wordt de tweede operand met het bovenste stapelelement verwisseld.

EXX

De registers BC, DE en HL worden verwisseld met de twee-registers.

HALT

De Z-80 stopt zolang met de uitvoering van het programma tot een interrupt is uitgevoerd.

IM N

Interruptmodus uitkiezen.
Modus 0: De bouwsteen die de interrupt nodig heeft, moet op de data-ingang een commando voor de uitvoering gereed houden.
Modus 1: De Z-80 springt naar het adres 0038 (hexadecimaal) en voert daar een interruptroutine uit.
Uw CPC werkt ook in deze modus. U mag daarom de interruptmodus ook nooit veranderen, omdat uw computer dan niet meer goed functioneert.
Modus 2: De bouwsteen levert de onderste helft van een adres, terwijl de bovenste helft in register I is opgeslagen. Onder het aangegeven adres staat de wijzer voor de start van de interruptroutine.

IN X,(C)

Register C geeft de poort aan waarvandaan een byte naar X moet worden verwezen.

IN A,(N)

De accu wordt met een byte van poort N geladen.

INC X

Zoals DEC, maar hierbij wordt de operand met 1 verhoogd.

IND

Het register HL dient als wijzer naar een geheugenplaats waar een byte van de poort wordt gelezen. Register C geeft het nummer van de poort aan. Bovendien worden HL en B met 1 verlaagd. Als B = 0, wordt Z-flag op 1 gezet. De overige flags (behalve carry) hebben willekeurige waarden.

INDR

Zoals IND, deze opdracht wordt echter zolang herhaald tot 6-0.

INI

Zoals IND, maar HL wordt dan verhoogd.

INIR

Zoals INDR, maar HL wordt dan verhoogd.

JP NN

Het programma springt naar adres nnnn (vergelijkbaar met GOTO).

JP voorwaarde, NN

Springt als aan de voorwaarde is voldaan (voor voorwaarde zie CALL).

JP X

Springt naar het adres dat in de operand X (HL, IX, IY) is opgeslagen.

JR N en JR voorwaarde, N

Zoals de overeenkomstige JP-commando's. Er wordt echter relatief gesprongen, dat wil zeggen, dat de byte die achter de opcode staat, bij de PC wordt opgeteld.

LD X,Y

In deze groep bevinden zich de meeste commando's. Ze zijn principieel echter hetzelfde. De inhoud van de tweede operand komt in de eerste operand te staan. Dat kan er bijv. zo uitzien dat de inhoud van de accu in de geheugencel nnnn wordt geladen, maar het kan ook andersom. Zo zijn er ook laad-commando's voor de 16-bits-registers. Als een 16-bits-register in het geheugen moet komen te staan, komt LOW-byte in nnnn en HIGH-byte in nnnn + 1 te staan. Deze vorm wordt in de computertechniek gebruikt (opmerking: eerst LOW-byte, dan HIGH-byte).

LDD

HL en DE worden als wijzers in het geheugen gebruikt. De inhoud van de cel die door HL wordt aangewezen, wordt naar het adres DE overgedragen. Daarna worden HL, DE en BC met 1 verlaagd. Als BC = 0, dan wordt de P-flag op 0 gezet. Op deze manier kan BC als teller worden gebruikt.

LDDR

Als LDD, maar de opdracht wordt automatisch zo lang herhaald tot BC = 0.

LDI

Als LDD; de registers HL en DE worden echter verhoogd.

LDIR

Als LDDR; de registers HL en DE worden echter verhoogd.

NEG

Het byte uit de accu wordt negatief gemaakt (dus met -1 vermenigvuldigd). De uitkomst komt in de accu te staan en de flags worden geactualiseerd. P = 1 als de inhoud van de accu voordien 80 (hexdecimaal) was, C = 1 als de accu de inhoud 0 had.

NOP

Met dit commando gebeurt er niets (vergelijkbaar met REM in BASIC). Is alleen nuttig voor kleine wachttijden.

OR X

De accu en de operand worden OR-verbonden. De uitkomst wordt weer in de accu gezet en de flags worden geactualiseerd. Daarbij wordt de carry-bit op 0 gezet, omdat bij een OR-verbinding geen overdracht kan optreden.

OTDR

HL dient als wijzer naar een geheugencel, waarvan de inhoud naar een poort (C) wordt gestuurd. Bovendien worden HL en B verlaagd. Dit wordt zo lang herhaald tot B = 0. Behalve de carry-bit en Z (die op 1 worden gezet) hebben alle andere flags toevallige waarden.

OTIR

Zoals OTDR, maar HL wordt echter verhoogd.

OUT (C), X

De operand X wordt naar een poort gestuurd waarbij het register C het nummer aanwijst.

OUT (N),A

De accu wordt naar poort N gestuurd.

OUTD

HL dient als wijzer naar een geheugencel, waarvan de inhoud naar poort (C) wordt gezonden. Bovendien worden HL en B verlaagd. Als B = 0 krijgt de Z-flag de waarde 1. De andere flags (behalve carry) hebben toevallige waarden.

OUTI

Zoals OUTD, HL wordt echter verhoogd.

POP X

Het registerpaar van de operand wordt van de stack gehaald en de stack-pointer wordt geactualiseerd. AF is de afkorting van Accu & Flags.

PUSH X

Het registerpaar van de operand wordt naar de stack gebracht en de stack-pointer wordt geactualiseerd. AF is de afkorting van Accu & Flags.

RES N,X

Het bit N van de operand X wordt uitgewist.

RET en RET-VOORWAARDE

De commando's veroorzaken een terugsprong uit een subroutine (vergelijkbaar met RETURN en BASIC). Hier kan ook nog een voorwaarde aan vastzitten, dat wil zeggen dat de terugsprong slechts dan wordt uitgevoerd als een bepaalde flag een zekere waarde heeft.

RETI

Veroorzaakt een terugsprong uit een interrupt (zoals RET).

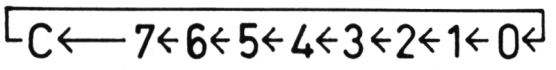
RETN

Veroorzaakt een terugsprong vanuit een NMI-routine.

RL X en RLA

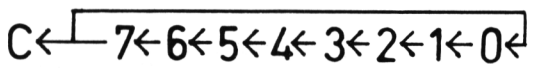
De operand wordt naar links gedraaid, waarbij het carry-bit in bit 0 en bit 7 in het carry-bit wordt geschoven (zie afbeelding). De RL-commando's veranderen alle flags.

Het RLA-commando (niet RL A) vormt hierop een kleine uitzondering. Het werkt als RL A, maar beïnvloedt alleen de carry-flag.



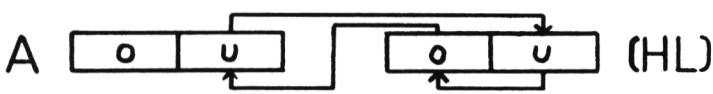
RLC X en RLCA

De operand wordt naar links gedraaid, waarbij bit 7 in de carry komt te staan (zie afbeelding).
Behalve RLCA veranderen de RLC-commando's alle flags.



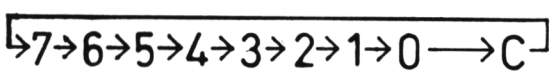
RLD

Met dit commando wordt een BDC-rotatie uitgevoerd, dat wil zeggen, de linkerhelft van de geheugencel waarvan het adres in HL staat, wordt naar de rechterhelft van de accu verplaatst, de rechterhelft van deze cel gaat naar links en de oorspronkelijke rechterhelft van de accu komt op de rechterhelft van de geheugencel te staan (zie de afbeelding). De flags S, Z en P worden beïnvloed.



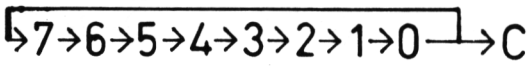
RR X en RRA

Deze commando's werken als RL X en RLA, alleen wordt de operand rechtsom gedraaid.



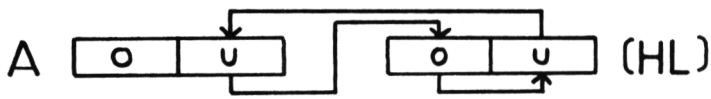
RRC X en RRCA

Zoals RLC X en RLCA, maar wordt naar rechts gedraaid.



RRD

Als RLD, echter een rotatie naar rechts. De onderste accu-nibble (1 nibble is een halve byte) wordt geschoven naar de bovenste nibble van de geheugencel, die door HL wordt aangewezen. De nibble van de bovenste geheugencel wordt naar het onderste geschoven en het onderste komt op de onderste helft van de accu te staan.



RST N

Bij een (vast) adres n wordt een subroutine gestart.

SBC A,X

De operand X wordt van de accu afgetrokken, waarbij ook op een eventueel transport uit het carry-bit wordt gelet. De uitkomst hiervan komt weer in de accu te staan, een nieuwe overdracht wordt in het carry-bit opgeslagen. Na de uitvoering ervan worden de flags overeenkomstig de accu-inhoud gezet.

SBC HL,X

Dit commando werkt als het vorige, alleen veroorzaakt het nu een 16-bits-optelling. Dan worden de operand (nu een 16-bits-register) en het carry-bit van het registerpaar HL afgetrokken. De uitkomsten komen weer in HL te staan en de flags worden overeenkomstig geplaatst.

SCF

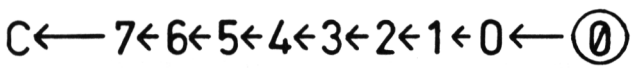
Het carry-bit krijgt de waarde 1.

SET N,X

Bit n van de operand X krijgt de waarde 1.

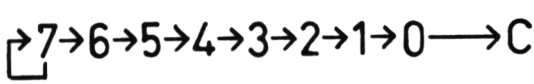
SLA X

De operand wordt naar links geschoven. Bit 0 wordt opgevuld met een 0, bit 7 komt in het carry-bit (zie afbeelding). De flags worden overeenkomstig geplaatst.



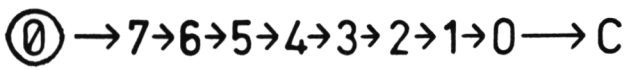
SRA X

De operand wordt naar rechts geschoven; bit 7 (het voorteken) blijft bestaan. Bit 0 komt in het carry-bit (zie afbeelding). Alle flags worden beïnvloed.



SRL X

De operand wordt naar rechts geschoven. Bit 7 wordt met een 0 opgevuld, bit 0 komt in het carry-bit (zie afbeelding). Alle flags worden beïnvloed.



SUB X

De operand wordt van de accu afgetrokken, de flags worden in overeenstemming met de uitkomst geplaatst en dit laatste wordt in de accu opgeslagen.

XOR X

De operand wordt met de accu 'exclusief-of' verbonden. De uitkomst komt in de accu te staan. De flags worden geactualiseerd, waarbij het carry-bit uitgewist wordt (XOR roept geen overdracht op).

13.14. Z-80-OPCODES

In de volgende tabel zijn alle beschikbare commando's van de Z-80 met opcodes vermeld. Bovendien vermeldt de kolom 'flags' alle flags die door een commando veranderd worden. Bij een aantal commando's kan het echter gebeuren dat na het uitvoeren verschillende flags toevallig geplaatst zijn. U wordt verzocht om deze gevallen uit de uitvoerige omschrijvingen te halen.

Mnemonic	Opcode	Flags	Omschrijving
ADC A,A	8F	S Z P C	Telt carry en A op tot A
ADC A,B	88	S Z P C	. . . tot B
ADC A,C	89	S Z P C	. . . tot C
ADC A,D	8A	S Z P C	. . . tot D
ADC A,E	8B	S Z P C	. . . tot E
ADC A,H	8C	S Z P C	. . . tot H
ADC A,L	8D	S Z P C	. . . tot L
ADC A,n	CEnn	S Z P C	. . . tot volgende byte
ADC A,(HL)	8E	S Z P C	. . . tot adres in HL
ADC A,(IX + d)	DD8Edd	S Z P C	. . . tot adres in IX + d
ADC A,(IY + d)	FD8Edd	S Z P C	. . . tot adres in IY + d
ADC HL,BC	ED4A	S Z P C	Telt carry en HL op tot BC
ADC HL,DE	ED5A	S Z P C	. . . tot DE
ADC HL,HL	ED6A	S Z P C	. . . tot HL
ADC HL,SP	ED7A	S Z P C	. . . tot SP
ADD A,A	87	S Z P C	Telt accu op tot accu
ADD A,B	80	S Z P C	. . . tot B
ADD A,C	81	S Z P C	. . . tot C
ADD A,D	82	S Z P C	. . . tot D
ADD A,E	83	S Z P C	. . . tot E
ADD A,H	84	S Z P C	. . . tot H
ADD A,L	85	S Z P C	. . . tot L
ADD A,n	C6nn	S Z P C	. . . tot volgende byte
ADD A,(HL)	86	S Z P C	. . . tot adres in HL
ADD A,(IX + d)	DD86dd	S Z P C	. . . tot tot adres in IX + d
ADD A,(IY + d)	FD86dd	S Z P C	. . . tot adres in IY + d
ADD HL,BC	09	C	Telt HL op bij BC
ADD HL,DE	19	C	. . . op bij DE
ADD HL,HL	29	C	. . . op bij HL
ADD HL,SP	39	C	. . . op bij SP

ADD IX,BC	DD09	C	Telt IX op bij BC
ADD IX,DE	DD19	C	. . . op bij DE
ADD IX,HL	DD29	C	. . . op bij HL
ADD IX,SP	DD39	C	. . . op bij SP
ADD IY,BC	FD09	C	Telt IY op bij BC
ADD IY,DE	FD19	C	. . . op bij DE
ADD IY,HL	FD29	C	. . . op bij HL
ADD IY,SP	FD39	C	. . . op bij SP
AND A	A7	S Z P C = 0	Verbindt accu AND met accu
AND B	A0	S Z P C = 0	. . . met B
AND C	A1	S Z P C = 0	. . . met C
AND D	A2	S Z P C = 0	. . . met D
AND E	A3	S Z P C = 0	. . . met E
AND H	A4	S Z P C = 0	. . . met H
AND L	A5	S Z P C = 0	. . . met L
AND n	E6nn	S Z P C = 0	. . . met volgende byte
AND (HL)	96	S Z P C = 0	. . . met adres in HL
AND (IX + d)	DD96dd	S Z P C = 0	. . . met adres in IX + d
AND (IY + d)	FD96dd	S Z P C = 0	. . . met adres in IY + d
BIT 0,A	CB47	Z	Test bit 0 van de accu
BIT 0,B	CB40	Z	. . . van B
BIT 0,C	CB41	Z	. . . van C
BIT 0,D	CB42	Z	. . . van D
BIT 0,E	CB43	Z	. . . van E
BIT 0,H	CB44	Z	. . . van H
BIT 0,L	CB45	Z	. . . van L
BIT O,(HL)	CB46	Z	. . . van adres in HL
BIT 0,(IX + d)	DDCBdd46	Z	. . . van adres in IX + d
BIT 0,(IY + d)	FDCBdd46	Z	. . . van adres in IY + d
BIT 1,A	CB4F	Z	Test bit 1 van de accu
BIT 1,B	CB48	Z	. . . van B
BIT 1,C	CB49	Z	. . . van C
BIT 1,D	CB4A	Z	. . . van D
BIT 1,E	CB4B	Z	. . . van E
BIT 1,H	CB4C	Z	. . . van H
BIT 1,L	CB4D	Z	. . . van L
BIT 1,(HL)	CB4E	Z	. . . van adres in HL
BIT 1,(IX + d)	DDCBdd4E	Z	. . . van adres in IX + d
BIT 1,(IY + d)	FDCBdd4E	Z	. . . van adres in IY + d
BIT 2,A	CB57	Z	Test bit 2 van de accu
BIT 2,B	CB50	Z	. . . van B
BIT 2,C	CB51	Z	. . . van C
BIT 2,D	CB52	Z	. . . van D

BIT 2,E	CB53	Z	. . . van E
BIT 2,H	CB54	Z	. . . van H
BIT 2,L	CB55	Z	. . . van L
BIT 2,(HL)	CB56	Z	. . . van (HL)
BIT 2,(IX + d)	DDCBdd56	Z	. . . van adres in IX + d
BIT 2,(IY + d)	FDCBdd56	Z	. . . van adres in IY + d
BIT 3,A	CB5F	Z	Test bit 3 van de accu
BIT 3,B	CB58	Z	. . . van B
BIT 3,C	CB59	Z	. . . van C
BIT 3,D	CB5A	Z	. . . van D
BIT 3,E	CB5B	Z	. . . van E
BIT 3,H	CB5C	Z	. . . van H
BIT 3,L	CB5D	Z	. . . van L
BIT 3,(HL)	CB5E	Z	. . . van adres in HL
BIT 3,(IX + d)	DDCBdd5E	Z	. . . van adres in IX + d
BIT 3,(IY + d)	FDCBdd5E	Z	. . . van adres in IY + d
BIT 4,A	CB67	Z	Test bit 4 van de accu
BIT 4,B	CB60	Z	. . . van B
BIT 4,C	CB61	Z	. . . van C
BIT 4,D	CB62	Z	. . . van D
BIT 4,E	CB63	Z	. . . van E
BIT 4,H	CB64	Z	. . . van H
BIT 4,L	CB65	Z	. . . van L
BIT 4,(HL)	CB66	Z	. . . van adres in HL
BIT 4,(IX + d)	DDCBdd66	Z	. . . van adres in IX + d
BIT 4,(IY + d)	FDCBdd66	Z	. . . van adres in IY + d
BIT 5,A	CB6F	Z	Test bit 5 van de accu
BIT 5,B	CB68	Z	. . . van B
BIT 5,C	CB69	Z	. . . van C
BIT 5,D	CB6A	Z	. . . van D
BIT 5,E	CB6B	Z	. . . van E
BIT 5,H	CB6C	Z	. . . van H
BIT 5,L	CB6D	Z	. . . van L
BIT 5,(HL)	CB6E	Z	. . . van adres in HL
BIT 5,(IX + d)	DDCBdd6E	Z	. . . van adres in IX + d
BIT 5,(IY + d)	FDCBdd6E	Z	. . . van adres in IY + d
BIT 6,A	CB77	Z	Test bit 6 van de accu
BIT 6,B	CB70	Z	. . . van B
BIT 6,C	CB71	Z	. . . van C
BIT 6,D	CB72	Z	. . . van D
BIT 6,E	CB73	Z	. . . van E
BIT 6,H	CB74	Z	. . . van H
BIT 6,L	CB75	Z	. . . van L

BIT 6,(HL)	CB76	Z	. . . van adres in HL
BIT 6,(IX + d)	DDCBdd76	Z	. . . van adres in IX + d
BIT 6,(IY + d)	FDCBdd76	Z	. . . van adres in IY + d
BIT 7,A	CB7F	Z	Test bit 7 van de accu
BIT 7,B	CB78	Z	. . . van B
BIT 7,C	CB79	Z	. . . van C
BIT 7,D	CB7A	Z	. . . van D
BIT 7,E	CB7B	Z	. . . van E
BIT 7,H	CB7C	Z	. . . van H
BIT 7,L	CB7D	Z	. . . van L
BIT 7,(HL)	CB7E	Z	. . . van adres in HL
BIT 7,(IX + d)	DDCBdd7E	Z	. . . van adres in IX + d
BIT 7,(IY + d)	FDCBdd7E	Z	. . . van adres in IY + d
CALL nn	CDnnnn		Roept subroutine nnnn op
CALL C,nn	DCnnn		. . . als carry = 1
CALL M,nn	FCnnnn		. . . als S = 1 (minus)
CALL NC,nn	D4nnnn		. . . als carry = 0
CALL NZ,nn	C4nnnn		. . . als Z = 0
CALL P,nn	F4nnnn		. . . als S = 0 (plus)
CALL PE,nn	ECnnnn		. . . als P = 1
CALL PO,nn	E4nnnn		. . . als P = 0
CALL Z,nn	CCnnnn		. . . als Z = 1
CCF	3F	C	Complementeert de carry-flag
CP A	BF	S Z P C	Vergelijkt A met A
CP B	B8	S Z P C	. . . met B
CP C	B9	S Z P C	. . . met C
CP D	BA	S Z P C	. . . met D
CP E	BB	S Z P C	. . . met E
CP H	BC	S Z P C	. . . met H
CP L	BD	S Z P C	. . . met L
CP n	FEnn	S Z P C	. . . met volgende byte
CP (HL)	BE	S Z P C	. . . met adres in HL
CP (IX + d)	DDBEdd	S Z P C	. . . met IX + d
CP (IY + d)	FDBEdd	S Z P C	. . . met IY + d
CPD	EDA9	S Z P	Vergelijkt en verlaagt
CPDR	EDB9	S Z P	Vergelijkt een blok en verlaagt
CPI	EDA1	S Z P	Vergelijkt en verhoogt
CPIR	EDB1	S Z P	Vergelijkt een blok en verhoogt
CPL	2F		Complementeert accu
DAA	27	S Z P C	BCD-aanpassing van accu

DEC A	3D	S Z P	Verlaagt A
DEC B	05	S Z P	. . . B
DEC BC	0B		. . . BC
DEC C	0D	S Z P	. . . C
DEC D	15	S Z P	. . . D
DEC DE	1B		. . . DE
DEC E	1D	S Z P	. . . E
DEC H	25	S Z P	. . . H
DEC HL	2B		. . . HL
DEC IX	DD2B		. . . IX
DEC IY	FD2B		. . . IY
DEC L	2D	S Z P	. . . L
DEC SP	3B		. . . SP
DEC (HL)	35	S Z P	. . . adres in HL
DEC (IX + d)	DD35dd	S Z P	. . . adres in IX + d
DEC (IY + d)	FD35dd	S Z P	. . . adres in IY + d
DI	F3		Sluit interrupt
DJNZ e	10ee		Verlaging en sprong bij ()
EI	FB		Openen interrupt
EX AF,AF'	08		Verwisselt A/flags met tweede register
EX DE,HL	EB		Verwisselt DE met HL
EX (SP),HL	E3		Verwisselt HL met stack-byte
EX (SP),IX	DDE3		. . . met IX
EX (SP),IY	FDE3		. . . met IY
EXX	D9		Verwisselt BC,DE,HL met tweede register
HALT	76		Stopt Z-80 tot interrupt
IM 0	ED46		Interrupt modus 0
IM 1	ED56		Interrupt modus 1
IM 2	ED5E		Interrupt modus 2
IN A,(C)	ED78	S Z P	Leest byte van poort C in A
IN A,(n)	DBnn		. . . van poort n
IN B,(C)	ED40	S Z P	. . . in B
IN C,(C)	ED48	S Z P	. . . in C
IN D,(C)	ED50	S Z P	. . . in D
IN E,(C)	ED58	S Z P	. . . in E
IN H,(C)	ED60	S Z P	. . . in H
IN L,(C)	ED68	S Z P	. . . in L

INC A	3C	S Z P	Verlaagt A
INC B	04	S Z P	. . . B
INC BC	03		. . . BC
INC C	0C	S Z P	. . . C
INC D	14	S Z P	. . . D
INC DE	13		. . . DE
INC E	1C	S Z P	. . . E
INC H	24	S Z P	. . . H
INC HL	23		. . . HL
INC IX	DD23		. . . IX
INC IY	FD23		. . . IY
INC L	2C	S Z P	. . . L
INC SP	33		. . . SP
INC (HL)	34	S Z P	. . . adres in HL
INC (IX + d)	DD34dd	S Z P	. . . adres IX + d
INC (IY + d)	FD34dd	S Z P	. . . adres IY + d
IND	EDAA	Z	Inlezen met verlagen
INDR	EDBA	Z = 1	Inlezen van blok met verlagen
INI	EDA2	Z	Inlezen met verhogen
INIR	EDB2	Z = 1	Inlezen van blok met verhogen
JP C,nn	DAnnnn		Springt naar nn als C = 1
JP M,nn	FAnnnn		. . . naar nn als S = 1
JP NC,nn	D2nnnn		. . . naar nn als C = 0
JP nn	C3nnnn		. . . naar nn
JP NZ,nn	C2nnnn		. . . naar nn als Z = 0
JP P,nn	F2nnnn		. . . naar nn als S = 0
JP PE,nn	EAnnnn		. . . naar nn als P = 1
JP PO,nn	E2nnnn		. . . naar nn als P = 0
JP Z,nn	CAnnnn		. . . naar nn als Z = 1
JP (HL)	E9		. . . naar adres in HL
JP (IX)	DDE9		. . . naar adres in IX
JP (IY)	FDE9		. . . naar adres in IY
JR C,e	38ee		Springt relatief als C = 1
JR e	18ee		. . . naar PC + e
JR NC,e	30ee		. . . als C = 0
JR NZ,e	20ee		. . . als Z = 0
JR Z,e	28ee		. . . als Z = 1
LD A,A	7F		Laadt accu met accu
LD A,B	78		. . . met B
LD A,C	79		. . . met C
LD A,D	7A		. . . met D
LD A,E	7B		. . . met E
LD A,H	7C		. . . met H

LD A,I	ED57	S Z P	. . . met I
LD A,L	7D		. . . met L
LD A,n	3Enn		. . . met volgende byte
LD A,R	ED5F	S Z P	. . . met R
LD A,(BC)	0A		. . . uit adres in BC
LD A,(DE)	1A		. . . uit adres in DE
LD A,(HL)	7E		. . . uit adres in HL
LD A,(IX + d)	DD7Edd		. . . uit adres in IX + d
LD A,(IY + d)	FD7Edd		. . . uit adres in IY + d
LD A,(nn)	3Annnn		. . . uit adres in nn
LD B,A	47		Laadt B met accu
LD B,B	40		. . . met B
LD B,C	41		. . . met C
LD B,D	42		. . . met D
LD B,E	43		. . . met E
LD B,H	44		. . . met H
LD B,L	45		. . . met L
LD B,n	06nn		. . . met volgende byte
LD B,(HL)	46		. . . uit adres in HL
LD B,(IX + d)	DD46dd		. . . uit adres in IX + d
LD B,(IY + d)	FD46dd		. . . uit adres in IY + d
LD BC,nn	01nnnn		Laadt BC met de volgende byte
LD BC,(nn)	ED4Bnnnn		. . . met nnnn en nnnn + 1
LD C,A	4F		Laadt C met accu
LD C,B	48		. . . met B
LD C,C	49		. . . met C
LD C,D	4A		. . . met D
LD C,E	4B		. . . met E
LD C,H	4C		. . . met H
LD C,L	4D		. . . met L
LD C,n	0Enn		. . . met volgende byte
LD C,(HL)	4E		. . . uit adres in HL
LD C,(IX + d)	DD4Edd		. . . uit adres in IX + d
LD C,(IY + d)	FD4Edd		. . . uit adres in IY + d
LD D,A	57		Laadt D met accu
LD D,B	50		. . . met B
LD D,C	51		. . . met C
LD D,D	52		. . . met D
LD D,E	53		. . . met E
LD D,H	54		. . . met H
LD D,L	55		. . . met L
LD D,n	16nn		. . . met volgende byte

LD D,(HL)	56	. . . uit adres in HL
LD D,(IX + d)	DD56dd	. . . uit adres in IX + d
LD D,(IY + d)	FD56dd	. . . uit adres in IY + d
LD DE,nn	11nnnn	Laadt DE met volgende byte
LD DE,(nn)	ED5Bnnnn	. . . met nnnn en nnnn + 1
LD E,A	5F	Laadt E met accu
LD E,B	58	. . . met B
LD E,C	59	. . . met C
LD E,D	5A	. . . met D
LD E,E	5B	. . . met E
LD E,H	5C	. . . met H
LD E,L	5D	. . . met L
LD E,n	1Enn	. . . met volgende byte
LD E,(HL)	5E	. . . uit adres in HL
LD E,(IX + d)	DD5Edd	. . . uit adres in IX + d
LD E,(IY + d)	FD5Edd	. . . uit adres in IY + d
LD H,A	67	Laadt H met accu
LD H,B	60	. . . met B
LD H,C	61	. . . met C
LD H,D	62	. . . met D
LD H,E	63	. . . met E
LD H,H	64	. . . met H
LD H,L	65	. . . met L
LD H,n	26nn	. . . met volgende byte
LD H,(HL)	66	. . . uit adres in HL
LD H,(IX + d)	DD66dd	. . . uit adres in IX + d
LD H,(IY + d)	FD66dd	. . . uit adres in IY + d
LD H, nn	21nnnn	Laadt HL met de volgende byte
LD H,(nn)	2Annnn	uit nnnn en nnnn + 1
LD I,A	ED47	Laadt I met accu
LD IX,nn	DD21nn	Laadt IX met de volgende byte
LD IX,(nn)	DD2Annnn	. . . uit nnnn en nnnn + 1
LD IY,nn	FD21nnnn	Laadt IY met de volgende byte
LD IY,(nn)	FD2Annnn	. . . uit nnnn en nnnn + 1
LD L,A	6F	Laadt L met accu
LD L,B	68	. . . met B
LD L,C	69	. . . met C
LD L,D	6A	. . . met D
LD L,E	6B	. . . met E
LD L,H	6C	. . . met H
LD L,L	6D	. . . met L
LD L,n	2Enn	. . . met volgende byte
LD L,(HL)	6E	. . . uit adres in HL

LD L,(IX + d)	DD6Edd		. . . uit adres in IX + d
LD L,(IY + d)	FD6Edd		. . . uit adres in IY + d
LD R,A	ED4F		Laadt R met accu
LD SP,HL	F9		Laadt SP met HL
LD SP,IX	DDF9		. . . met IX
LD SP,IY	FDF9		. . . met IY
LD SP,nn	31nnnn		. . . met volgende byte
LD SP,(nn)	ED7Bnnnn		. . . uit nnnn en nnnn + 1
LD (BC),A	02		Laadt adres uit BC met accu
LD (DE),A	12		Laadt adres uit DE met accu
LD (HL),A	77		Laadt adres uit HL met accu
LD (HL),B	70		. . . met B
LD (HL),C	71		. . . met C
LD (HL),D	72		. . . met D
LD (HL),E	73		. . . met E
LD (HL),H	74		. . . met H
LD (HL),L	75		. . . met L
LD (HL),n	36nn		. . . met volgende byte
LD (IX + d),A	DD77dd		Laadt adres IX + d met accu
LD (IX + d),B	DD70dd		. . . met B
LD (IX + d),C	DD71dd		. . . met C
LD (IX + d),D	DD72dd		. . . met D
LD (IX + d),E	DD73dd		. . . met E
LD (IX + d),H	DD74dd		. . . met H
LD (IX + d),L	DD75dd		. . . met L
LD (IX + d),n	DD36ddnn		. . . met volgende byte
LD (IY + d),A	FD77dd		Laadt adres IY + d met accu
LD (IY + d),B	FD70dd		. . . met B
LD (IY + d),C	FD71dd		. . . met C
LD (IY + d),D	FD72dd		. . . met D
LD (IY + d),E	FD73dd		. . . met E
LD (IY + d),H	FD74dd		. . . met H
LD (IY + d),L	FD75dd		. . . met L
LD (IY + d),n	FD36ddnn		. . . met volgende byte
LD (nn),A	32nnnn		Laadt cel nnnn met accu
LD (nn),BC	ED43nnnn		. . . met C en nnnn + 1 met B
LD (nn),DE	ED53nnnn		. . . met E en nnnn + 1 met D
LD (nn),HL	22nnnn		. . . met L en nnnn + 1 met H
LD (nn),IX	DD22nnnn		. . . en nnnn + 1 met IX
LD (nn),IY	FD22nnnn		. . . en nnnn + 1 met IY
LD (nn),SP	ED73nnn		. . . en nnnn + 1 met SP
LDD	EDA8	P	Laden en verlagen
LDDR	EDB8	P = 0	Laden van blok en verlagen

LDI	EDA0	P	Laden en verhogen
LDIR	EDB0	P = 0	Laden van blok en verhogen
NEG	ED44	S Z P C	Maakt accu negatief
NOP	00		Wacht (nul-bewerking)
OR A	B7	S Z P C	OR verbonden accu met accu
OR B	B0	S Z P C	. . . met B
OR C	B1	S Z P C	. . . met C
OR D	B2	S Z P C	. . . met D
OR E	B3	S Z P C	. . . met E
OR H	B4	S Z P C	. . . met H
OR L	B5	S Z P C	. . . met L
OR n	F6nn	S Z P C	. . . met volgende byte
OR (HL)	B6	S Z P C	. . . met adres in HL
OR (IX + d)	DDB6dd	S Z P C	. . . met adres in IX + d
OR (IY + d)	FDB6dd	S Z P C	. . . met adres in IY + d
OTOR	EDBB	S Z P	Uitvoeren van blokken met verlagen
OTIR	EDB3	S Z P	Uitvoeren van blokken met verhogen
OUT (C),A	ED79		Voert via poort C de accu uit
OUT (C),B	ED41		. . . B uit
OUT (C),C	ED49		. . . C uit
OUT (C),D	ED51		. . . D uit
OUT (C),E	ED59		. . . E uit
OUT (C),H	ED61		. . . H uit
OUT (C),L	ED69		. . . L uit
OUT (n),A	D3nn		Geeft accu aan poort n uit
OUTD	EDAB	S Z P	Uitvoeren en verlagen
OUTI	EDA3	S Z P	Uitvoeren en verhogen
POP AF	F1		Haalt accu en flags van stack
POP BC	C1		Haalt BC van stack
POP DE	D1		Haalt DE van stack
POP HL	E1		Haalt HL van stack
POP IX	DDE1		Haalt IX van stack
POP IY	FDE1		Haalt IY van stack
PUSH AF	F5		Brengt accu en flags naar stack
PUSH BC	C5		Brengt BC naar stack
PUSH DE	D5		Brengt DE naar stack
PUSH HL	E5		Brengt HL naar stack
PUSH IX	DDE5		Brengt IX naar stack
PUSH IY	FDE5		Brengt IY naar stack

RES 0,A	CB87	Wist bit 0 van accu uit
RES 0,B	CB80	. . . van B uit
RES 0,C	CB81	. . . van C uit
RES 0,D	CB82	. . . van D uit
RES 0,E	CB83	. . . van E uit
RES 0,H	CB84	. . . van H uit
RES 0,L	CB85	. . . van L uit
RES 0,(HL)	CB86	. . . van adres in HL uit
RES 0,(IX + d)	DDCBdd86	. . . van adres IX + d uit
RES 0,(IY + d)	FDCBdd86	. . . van adres IY + d uit
RES 1,A	CB8F	Wist bit 1 van accu uit
RES 1,B	CB88	. . . van B uit
RES 1,C	CB89	. . . van C uit
RES 1,D	CB8A	. . . van D uit
RES 1,E	CB8B	. . . van E uit
RES 1,H	CB8C	. . . van H uit
RES 1,L	CB8D	. . . van L uit
RES 1,(HL)	CB8E	. . . van adres in HL uit
RES 1,(IX + d)	DDCBdd8E	. . . van adres IX + d
RES 1,(IY + d)	FDCBdd8E	. . . van adres IY + d
RES 2,A	CB97	Wist bit 2 van accu uit
RES 2,B	CB90	. . . van B uit
RES 2,C	CB91	. . . van C uit
RES 2,D	CB92	. . . van D uit
RES 2,E	CB93	. . . van E uit
RES 2,H	CB94	. . . van H uit
RES 2,L	CB95	. . . van L uit
RES 2,(HL)	CB96	. . . van adres in HL uit
RES 2,(IX + d)	DDCBdd96	. . . van adres IX + d
RES 2,(IY + d)	FDCBdd96	. . . van adres IY + d
RES 3,A	CB9F	Wist bit 3 van accu uit
RES 3,B	CB98	. . . van B uit
RES 3,C	CB99	. . . van C uit
RES 3,D	CB9A	. . . van D uit
RES 3,E	CB9B	. . . van E uit
RES 3,H	CB9C	. . . van H uit
RES 3,L	CB9D	. . . van L uit
RES 3,(HL)	CB9E	. . . van adres in HL uit
RES 3,(IX + d)	DDCBdd9E	. . . van adres IX + d
RES 3,(IY + d)	FDCBdd9E	. . . van adres IY + d
RES 4,A	CBA7	Wist bit 4 van accu uit
RES 4,B	CBA0	. . . van B uit
RES 4,C	CBA1	. . . van C uit

RES 4,D	CBA2	. . . van D uit
RES 4,E	CBA3	. . . van E uit
RES 4,H	CBA4	. . . van H uit
RES 4,L	CBA5	. . . van L uit
RES 4,(HL)	CBA6	. . . van adres in HL uit
RES 4,(IX + d)	DDCBddA6	. . . van adres IX + d
RES 4,(IY + d)	FDCBddA6	. . . van adres IY + d
RES 5,A	CBAF	Wist bit 5 van accu uit
RES 5,B	CBA8	. . . van B uit
RES 5,C	CBA9	. . . van C uit
RES 5,D	CBAA	. . . van D uit
RES 5,E	CBAB	. . . van E uit
RES 5,H	CBAC	. . . van H uit
RES 5,L	CBAD	. . . van L uit
RES 5,(HL)	CBAE	. . . van adres in HL uit
RES 5,(IX + d)	DDCBddAE	. . . van adres IX + d
RES 5,(IY + d)	FDCBddAE	. . . van adres IY + d
RES 6,A	CBB7	Wist bit 6 van accu uit
RES 6,B	CBB0	. . . van B uit
RES 6,C	CBB1	. . . van C uit
RES 6,D	CBB2	. . . van D uit
RES 6,E	CBB3	. . . van E uit
RES 6,H	CBB4	. . . van H uit
RES 6,L	CBB5	. . . van L uit
RES 6,(HL)	CBB6	. . . van adres in HL uit
RES 6,(IX + d)	DDCBddB6	. . . van adres IX + d
RES 6,(IY + d)	FDCBddB6	. . . van adres IY + d
RES 7,A	CBBF	Wist bit 7 van accu uit
RES 7,B	CBB8	. . . van B uit
RES 7,C	CBB9	. . . van C uit
RES 7,D	CBBA	. . . van D uit
RES 7,E	CBBB	. . . van E uit
RES 7,H	CBBC	. . . van H uit
RES 7,L	CBBD	. . . van L uit
RES 7,(HL)	CBBE	. . . van adres in HL uit
RES 7,(IX + d)	DDCBddBE	. . . van adres IX + d
RES 7,(IY + d)	FDCBddBE	. . . van adres IY + d
RET	C9	Terug uit onderprogramma
RET C	D8	. . . als C = 1
RET M	F8	. . . als S = 1
RET NC	D0	. . . als C = 0
RET P	F0	. . . als S = 0
RET PE	E8	. . . als P = 1

RET PO	E0		. . . als P = 0
RET Z	C8		. . . als Z = 1
RETI	ED4D		Terug uit interrupt
RETN	ED45		. . . uit NMI-routine
RL A	CB17	S Z P C	Links-rotatie van carry en A
RL B	CB10	S Z P C	. . . en B
RL C	CB11	S Z P C	. . . en C
RL D	CB12	S Z P C	. . . en D
RL E	CB13	S Z P C	. . . en E
RL H	CB14	S Z P C	. . . en H
RL L	CB15	S Z P C	. . . en L
RL (HL)	CB16	S Z P C	. . . en adres in HL
RL (IX + d)	DDCBdd16	S Z P C	. . . en adres IX + d
RL (IY + d)	FDCBdd16	S Z P C	. . . en adres IY + d
RLA	17	C	. . . en accu
RLC A	CB07	S Z P C	Links-rotatie van de accu
RLC B	CB00	S Z P C	. . . van B
RLC C	CB01	S Z P C	. . . van C
RLC D	CB02	S Z P C	. . . van D
RLC E	CB03	S Z P C	. . . van E
RLC H	CB04	S Z P C	. . . van B
RLC L	CB05	S Z P C	. . . van B
RLC (HL)	CB06	S Z P C	. . . van adres in HL
RLC (IX + d)	DDCBdd06	S Z P C	. . . van adres IX + d
RLC (IY + d)	FDCBdd06	S Z P C	. . . van adres IY + d
RLCA	07	C	. . . van de accu
RLD	ED6F	S Z P	Decimaalrotatie links
RR A	CB17	S Z P C	Rechts-rotatie van carry en A
RR B	CB18	S Z P C	. . . en B
RR C	CB19	S Z P C	. . . en C
RR D	CB1A	S Z P C	. . . en D
RR E	CB1B	S Z P C	. . . en E
RR H	CB1C	S Z P C	. . . en B
RR L	CB1D	S Z P C	. . . en B
RR (HL)	CB1E	S Z P C	. . . en adres in HL
RR (IX + d)	DDCBdd1E	S Z P C	. . . en adres IX + d
RR (IY + d)	FDCBdd1E	S Z P C	. . . en adres IY + d
RRA	1F	C	. . . en de accu
RRC A	CB0F	S Z P C	Rechts-rotatie van de accu
RRC B	CB08	S Z P C	. . . van B
RRC C	CB09	S Z P C	. . . van C
RRC D	CB0A	S Z P C	. . . van D
RRC E	CB0B	S Z P C	. . . van E

RRC H	CB0C	S Z P C	. . . van H
RRC L	CB0D	S Z P C	. . . van L
RRC (HL)	CB0E	S Z P C	. . . van adres in HL
RRC (IX + d)	DDCBdd0E	S Z P C	. . . van adres IX + d
RRC (IY + d)	FDCBdd0E	S Z P C	. . . van adres IY + d
RRCA	0F	C	. . . van de accu
RRD	ED67	S Z P	Decimaalrotatie rechts
RST 00	C7		Oproep onderprogramma bij 00
RST 08	CF		. . . bij 08
RST 10	D7		. . . bij 10
RST 18	DF		. . . bij 18
RST 20	E7		. . . bij 20
RST 28	EF		. . . bij 28
RST 30	F7		. . . bij 30
RST 38	FF		. . . bij 38
SBC A,A	9F	S Z P C	Vermindert carry en A van A
SBC A,B	98	S Z P C	. . . B van accu
SBC A,C	99	S Z P C	. . . C van accu
SBC A,D	9A	S Z P C	. . . D van accu
SBC A,E	9B	S Z P C	. . . E van accu
SBC A,H	9C	S Z P C	. . . H van accu
SBC A,L	9D	S Z P C	. . . L van accu
SBC A,n	DEnn	S Z P C	. . . volgende byte van A
SBC A,(HL)	9E	S Z P C	. . . adres in HL van A
SBC A,(IX + d)	DD9Edd	S Z P C	. . . adres in IX + d van accu
SBC A,(IY + d)	FD9Edd	S Z P C	. . . adres in IY + d van accu
SBC HL,BC	ED42	S Z P C	Vermindert C en BC van HL
SBC HL,DE	ED52	S Z P C	. . . DE van HL
SBC HL,HL	ED62	S Z P C	. . . HL van HL
SBC HL,SP	ED72	S Z P C	. . . SP van HL
SCF	37	C = 1	Zet carry bit op 1
SET 0,A	CBC7		Zet bit 0 van de accu
SET 0,B	CBC0		. . . van B
SET 0,C	CBC1		. . . van C
SET 0,D	CBC2		. . . van D
SET 0,E	CBC3		. . . van E
SET 0,H	CBC4		. . . van H
SET 0,L	CBC5		. . . van L
SET 0,(HL)	CBC6		. . . van adres in HL
SET 0,(IX + d)	DDCBddC6		. . . van adres IX + d
SET 0,(IY + d)	FDCBddC6		. . . van adres IY + d
SET 1,A	CBCF		Zet bit 1 van de accu
SET 1,B	CBC8		. . . van B

SET 1,C	CBC9	. . . van C
SET 1,D	CBCA	. . . van D
SET 1,E	CBCB	. . . van E
SET 1,H	CBCC	. . . van H
SET 1,L	CBCD	. . . van L
SET 1,(HL)	CBCE	. . . van adres in HL
SET 1,(IX + d)	DDCBddCE	. . . van adres IX + d
SET 1,(IY + d)	FDCBddCE	. . . van adres IY + d
SET 2,A	CBD7	Zet bit 2 van de accu
SET 2,B	CBD0	. . . van B
SET 2,C	CBD1	. . . van C
SET 2,D	CBD2	. . . van D
SET 2,E	CBD3	. . . van E
SET 2,H	CBD4	. . . van H
SET 2,L	CBD5	. . . van L
SET 2,(HL)	CBD6	. . . van adres in HL
SET 2,(IX + d)	DDCBddD6	. . . van adres IX + d
SET 2,(IY + d)	FDCBddD6	. . . van adres IY + d
SET 3,A	CBDF	Zet bit 3 van de accu
SET 3,B	CBD8	. . . van B
SET 3,C	CBD9	. . . van C
SET 3,D	CBDA	. . . van D
SET 3,E	CBDB	. . . van E
SET 3,H	CBDC	. . . van H
SET 3,L	CBDD	. . . van L
SET 3,(HL)	CBDE	. . . van adres in HL
SET 3,(IX + d)	DDCBddDE	. . . van adres IX + d
SET 3,(IY + d)	FDCBddDE	. . . van adres IY + d
SET 4,A	CBE7	Zet bit 4 van de accu
SET 4,B	CBE0	. . . van B
SET 4,C	CBE1	. . . van C
SET 4,D	CBE2	. . . van D
SET 4,E	CBE3	. . . van E
SET 4,H	CBE4	. . . van H
SET 4,L	CBE5	. . . van L
SET 4,(HL)	CBE6	. . . van adres in HL
SET 4,(IX + d)	DDCBddE6	. . . van adres IX + d
SET 4,(IY + d)	FDCBddE6	. . . van adres IY + d
SET 5,A	CBEF	Zet bit 5 van de accu
SET 5,B	CBE8	. . . van B
SET 5,C	CBE9	. . . van C
SET 5,D	CBEA	. . . van D
SET 5,E	CBEB	. . . van E

SET 5,H	CBEC		. . . van H
SET 5,L	CBED		. . . van L
SET 5,(HL)	CBEE		. . . van adres in HL
SET 5,(IX + d)	DDCBddEE		. . . van adres IX + d
SET 5,(IY + d)	FDCBddEE		. . . van adres IY + d
SET 6,A	CBF7		Zet bit 6 van de accu
SET 6,B	CBF0		. . . van B
SET 6,C	CBF1		. . . van C
SET 6,D	CBF2		. . . van D
SET 6,E	CBF3		. . . van E
SET 6,H	CBF4		. . . van H
SET 6,L	CBF5		. . . van L
SET 6,(HL)	CBF6		. . . van adres in HL
SET 6,(IX + d)	DDCBddF6		. . . van adres IX + d
SET 6,(IY + d)	FDCBddF6		. . . van adres IY + d
SET 7,A	CBFF		Zet bit 7 van de accu
SET 7,B	CBF8		. . . van B
SET 7,C	CBF9		. . . van C
SET 7,D	CBFA		. . . van D
SET 7,E	CBFB		. . . van E
SET 7,H	CBFC		. . . van H
SET 7,L	CBFD		. . . van L
SET 7,(HL)	CBFE		. . . van adres in HL
SET 7,(IX + d)	DDCBddFE		. . . van adres IX + d
SET 7,(IY + d)	FDCBddFE		. . . van adres IY + d
SLA A	CB27	S Z P C	Schuift carry en accu links
SLA B	CB20	S Z P C	. . . en B links
SLA C	CB21	S Z P C	. . . en C links
SLA D	CB22	S Z P C	. . . en D links
SLA E	CB23	S Z P C	. . . en E links
SLA H	CB24	S Z P C	. . . en H links
SLA L	CB25	S Z P C	. . . en L links
SLA (HL)	CB26	S Z P C	. . . en adres in HL links
SLA (IX + d)	DDCBdd26	S Z P C	. . . en adres IX + d links
SLA (IY + d)	FDCBdd26	S Z P C	. . . en adres IY + d links
SRA A	CB2F	S Z P C	. . . en accu rechts
SRA B	CB28	S Z P C	. . . en B rechts
SRA C	CB29	S Z P C	. . . en C rechts
SRA D	CB2A	S Z P C	. . . en D rechts
SRA E	CB2B	S Z P C	. . . en E rechts
SRA H	CB2C	S Z P C	. . . en H rechts
SRA L	CB2D	S Z P C	. . . en L rechts
SRA (HL)	CB2E	S Z P C	. . . en adres in HL rechts

SRA (IX + d)	DDCBdd2E	S Z P C	. . . en adres IX + d rechts
SRA (IY + d)	FDCBdd2E	S Z P C	. . . en adres IY + d rechts
SRL A	CB3F	S Z P C	. . . en accu rechts
SRL B	CB38	S Z P C	. . . en B rechts
SRL C	CB39	S Z P C	. . . en C rechts
SRL D	CB3A	S Z P C	. . . en D rechts
SRL E	CB3B	S Z P C	. . . en E rechts
SRL H	CB3C	S Z P C	. . . en H rechts
SRL L	CB3D	S Z P C	. . . en L rechts
SRL (HL)	CB3E	S Z P C	. . . en adres in HL rechts
SRL (IX + d)	DDCBdd3E	S Z P C	. . . en adres IX + d rechts
SRL (IY + d)	FDCBdd3E	S Z P C	. . . en adres IY + d rechts
SUB A	97	S Z P C	Vermindert accu met accu
SUB B	90	S Z P C	. . . B met accu
SUB C	91	S Z P C	. . . C met accu
SUB D	92	S Z P C	. . . D met accu
SUB E	93	S Z P C	. . . E met accu
SUB H	94	S Z P C	. . . H met accu
SUB L	95	S Z P C	. . . L met accu
SUB n	D6nn	S Z P C	. . . volgende byte
SUB (HL)	96	S Z P C	. . . adres in HL
SUB (IX + d)	DD96dd	S Z P C	. . . adres IX + d van de accu
SUB (IY + d)	FD96dd	S Z P C	. . . adres IY + d van de accu
XOR A	AF	S Z P C = 0	Exclusief-of verbindt A en A
XOR B	A8	S Z P C = 0	. . . en B
XOR C	A9	S Z P C = 0	. . . en C
XOR D	AA	S Z P C = 0	. . . en D
XOR E	AB	S Z P C = 0	. . . en E
XOR H	AC	S Z P C = 0	. . . en H
XOR L	AD	S Z P C = 0	. . . en L
XOR n	EEnn	S Z P C = 0	. . . met volgende byte
XOR (HL)	AE	S Z P C = 0	. . . en adres in HL
XOR (IX + d)	DDAEdd	S Z P C = 0	. . . en adres IX + d
XOR (IY + d)	FDAEdd	S Z P C = 0	. . . en adres IY + d

14. TRUCS EN FORMULES IN BASIC

Met behulp van de volgende trucs en formules kunt u veel lastige opgaven bij het programmeren gemakkelijker oplossen, lastige fouten kunnen worden voorkomen en u spaart veel tijd uit. Laten we beginnen met wiskundeformules.

Misschien hebt u opgemerkt dat het BASIC van uw CPC (en ook van andere computers) alleen maar commando's heeft voor het natuurlijke logaritme en voor het logaritme met grondtal 10. Gelukkig bestaat er een eenvoudige formule waarmee de logaritmen van een willekeurig grondtal berekend kan worden. Deze luidt:

LOG-n (x) = LOG (X)/LOG (n)

Ook voor de omkeringen van goniometrische functies is de BASIC-interpreter niet zo best voorzien. Arcussinus en arcuscosinus ontbreken eraan! Beide zijn echter te berekenen met arcustangens (ATN):
ARCSIN(x) = ATN(x/SQR(1-x*x))
ARCCOS(x) = -ATN(x/SQR(1-x*x)) + PI/2

Om breuken te vereenvoudigen, moet u de grootste gemene deler (GGD) van twee getallen berekenen. Heel vaak wordt daarvoor het 'algoritme van Euclides' gebruikt, wat het voordeel heeft dat het zeer snel werkt.
De variabelen P en Q zijn deze twee getallen waarvan de GGD berekend moet worden. De rest van de deling P/Q noemen we de variabele R. Na de deling krijgt P de oude waarde van Q; Q krijgt de restwaarde R en de deling wordt opnieuw uitgevoerd. Dat gaat zo door tot R de waarde 0 heeft en dan staat de GGD in P. Deze droge verklaring willen we met een voorbeeld toelichten:

P	Q	R	Beginwaarden	
56	21	14	1.	doorgang
21	14	7	2.	doorgang
14	7	0	P is de GGD (P = 7)	
7	0			

De listing hiervan is verheugend kort:

```
10 P = getal 1: q = getal 2: r = 1
20 WHILE   O:r = p MOD q:p  =  Q:q = r:W EN D:ND
```

Het r = 1 is heel belangrijk, omdat anders direct aan deze voorwaarde wordt voldaan voordat de lus zelfs maar een enkele keer werd doorlopen.

Als u van een getal wilt weten hoeveel plaatsen er voor de komma staan, kunt u eenvoudig de volgende formule gebruiken:

```
S = INT(LOG10(ABS(x)) + 1)
```

De ABS-functie staat u toe om ook negatieve getallen te nemen, omdat LOG10 alleen voor positieve getallen is gedefiniëerd.

De volgende formule geeft een toevalsgetal uit het bereik van a tot b en niet (zoals gebruikelijk) van 0 tot b:

```
x = INT (RND(1)*(b-a + 1) + a)
```

Ook de laatste tip heeft betrekking op de geheimzinnige wereld van de wiskunde, als is deze nog zo ver verwijderd.

De basisomzetting van hexadecimale en binaire getallen door middel van & en &x heeft helaas een klein schoonheidsfoutje. Als de getallen groter worden dan &7FFF, staat het hoogste bit (nummer 15) op 1 en krijgt mu een negatieve waarde. Als dat zou gebeuren, kunt u er 65536 bij optellen, waardoor u de 'echte' waarde krijgt.

Op veel tekstverwerkers is het mogelijk om op verschillende manieren een tekst te doen verschijnen. Tot een van de mogelijkheden behoort de 'gecentreerde' uitvoer, die ook met een eenvoudige BASIC-formule is te verkrijgen:

```
PRINT TAB ((regellengte-LEN$(TEKST$))/2;TEKST$
```

De regellengte is bij verschillende modi anders, TEKST$ is de regel die moet worden afgedrukt.

De laatste truc heeft met strings te maken. Daarbij gaat het om een onaangename eigenschap van de CPC, die hij met andere BASIC-interpreters gemeen heeft. Als men probeert de ASCII-waarde van een lege string (dus ASC('')) te krijgen, reageert de computer met een 'IMPROPER ARGUMENT'. Men kan dat voorkomen met:

```
PRINT ASC(a$ + CHR$(0))
```

En nu is alles weer in orde.

APPENDIX

1. GEHEUGENOPBOUW

Op de volgende bladzijden vindt u een listing van de functies van alle geheugenbytes (voor zover bekend). Het is natuurlijk mogelijk dat er fouten in zijn geslopen. Daarom moeten experimenten door middel van PEEK en POKE met enige voorzichtigheid worden uitgevoerd. U moet zich hierdoor echter niet laten afschrikken, maar wel voordien uw programma SAVEn!

0000 -3FFF	Bedrijfssysteem-Rom
0000 -003F	Kopie van ROM voor bankswitching
0040 -013F	Invoerbuffer en gebied van de werkzaamheden
0170 -AB7F	BASIC-geheugen
3800 -3FFF	Karaktergenerator in ROM
AB80 -ABFF	Tekengenerator voor zelf te definiëren karakters
AC00	Flag voor spatie uitwissen
AC01 -AC03	Uitbreidingssprong voor READY
AC04 -AC06	. . . foutbehandeling
AC07 -AC09	. . . commando-uitvoering
AC0A -AC0C	. . . functieberekening
AC0D -AC0F	. . . constante halen
AC10 -AC12	. . . invoer van BASIC-regels
AC13 -AC15	. . . LIST
AC16 -AC18	. . . veranderen van cijfers
AC19 -AC1B	. . . operatoren
AC1C	Flag voor AUTO
AC1D -AC1E	AUTO-regelnummer
AC1F -AC20	stapgrootte voor AUTO
AC21	Laatste stream-nummer
AC22	Invoerkanaal
AC23	Laatste positie van de printer
AC24	WIDTH
AC25	Laatste positie op cassette
AC26	Flag voor FOR...NEXT
AC27 -AC2B	Tussengeheugen voor FOR
AC2C -AC2D	Adres voor NEXT
AC2E -AC2F	. . . WEND
AC34 -AC35	. . . ON-BREAK
AC38 -AC43	Toon 0
AC44 -AC4F	Toon 1
AC50 -AC5B	Toon 2
AC5C -AC6D	Bereik voor interrupt 0
AC6E -AC7F	. . . 1

AC80 -AC91	. . . 2
AC92 -ACA3	. . . 3
ACA4 -ADA3	Invoerbuffer
ADA6 -ADA7	Adres voor ERRORs
ADA8 -ADA9	BASIC-programmawijzer na ERROR
ADAA	Nummer van Error
ADAB-ADAC	BASIC-programmawijzer na interrupt
ADAD-ADAE	Adres van onderbroken regel
ADAF -ADB0	Adres voor ON-ERROR
ADB1	Flag: ON-ERROR actief
ADB2	Kanaalstatus
ADB3	ENT
ADB4	ENV
ADB5 -ADB6	Periode
ADB7	Ruisperiode
ADB8	Geluidssterkte
ADB9 -ADBA	Lengte
ADBB-ADBC	ENV en ENT
ADCB-ADCG	Tussengeheugen voor 'floating point'-berekeningen
ADD0 -AE03	Tabel voor schaalvariabelen
AE04 -AE05	FN-tabel
AE06 -AE0B	Array-tabel
AE0C -AE25	Voorgedefiniëerde variabelentypen A tot en met Z
AE2D	Scheidingsteken voor INPUT
AE2E -AE2F	Adres voor READ
AE30 -AE31	Adres voor DATA
AE32 -AE33	Geheugen voor pointer in BASIC-stack
AE34 -AE35	Actueel commando-adres
AE36 -AE37	Actueel programmaregeladres
AE38	Flag voor TRACE
AE3F -AE40	Startadres voor LOAD-commando
AE41	Flag voor CHAIN-MERGE
AE42	Bestandtype
AE43 -AE44	Lengte van bestand
AE45	Flag voor programmabeveiliging
AE46 -AE78	Werkgebied voor omzetten in ASC11
AE72 -AE73	CALL-adres
AE74	Bankswitchingsmodus voor CALL
AE75 -AE76	HL-register voor CALL
AE77 -AE78	SP-register voor CALL
AE79	Tabulatorbreedte
AE7B -AE7C	HIMEM-wijzer
AE7D -AE7E	Wijzer naar eind van vrije RAM-bereik

AE7F -AE80	. . . begin van vrije RAM-bereik
AE81 -AE82	. . . BASIC-programma start
AE83 -AE84	. . . einde programma
AE85 -AE86	. . . variabele start
AE87 -AE88	. . . array-start
AE89 -AE8A	. . . array-einde
AE8B -B08A	Stack voor BASIC (FOR, GOSUB enz.)
B08B -B08C	BASIC-stackpointer
B08D -B08E	Wijzer naar begin string
B08F -B090	Wijzer naar einde string
B09A -B098	Stringstack
B0BA -B0BC	Stringdestricptor
B0C1	Variabelentype
B0C2 -B0C3	Diverse adressen
B100 -B1AB	Werkomgeving voor besturing van bedrijfssysteem
B1C8	Bestaande beeldschermmodus
B1CA	Beeldschermoffset
B1CB	Beeldschermadres
B1CC -B1D6	Diverse doelen
B1D7 -BID8	Knippertijden
B1D9 -B20B	Verschillende registers voor knipperende kleuren
B20C	Actuele WINDOW
B20D -B276	Parameter voor WINDOWs
B285 -B286	Cursorpositie (regel, kolom)
B287 -B28B	Verschillende registers voor WINDOWs
B28C -B327	Verschillende registers voor beeldscherm
B328 -B329	ORIGIN-X
B32A -B32B	ORIGIN-Y
B32C -B4DD	Verschillende registers voor grafieken
B4DE -B550	. . . toetsenbord
B551 -B7FF	. . . SOUND
B800 -B8DC	. . . cassette
B807 -B816	Naam INPUT-bestand
B84C -B85B	Naam OUTPUT-bestand
B8D1	Schrijfsnelheid
B8DD	Insert-flag
B8E4 -B8E7	Toevalsgetal
B8E8 -B8D6	Tussengeheugen voor 'floating point'-berekeningen
B8F7	Flag voor DEG respectievelijk RAD
BF00 -BFFF	Processor-stack
C000 -FFFF	Video-RAM
C000 -FFFF	Interpreter-ROM
C000 -FFFF	Uitbreidings-ROM

Een boek voor iedereen die het gebruik van de PEEK en POKE-opdrachten wil kennen.
De opgenomen informatie bestrijkt het adresseerbereik van de microprocessor, het besturingssysteem en zelfs een inleiding in de machinetaal van de (veelgebruikte) Z80 a microprocessor.
Tenslotte vindt u de listings van complete utility's, handige routines en informatie over grafische toepassingen.

De listings zijn geschikt voor de CPC 464, 664 en 6128

ISBN 90 229 3363 6

CPC Bibliotheek 10

DATA BECKER
NEDERLANDS*

Peeks en pokes voor de CPC-computers

CPCB10

DATA BECKER NEDERLANDS