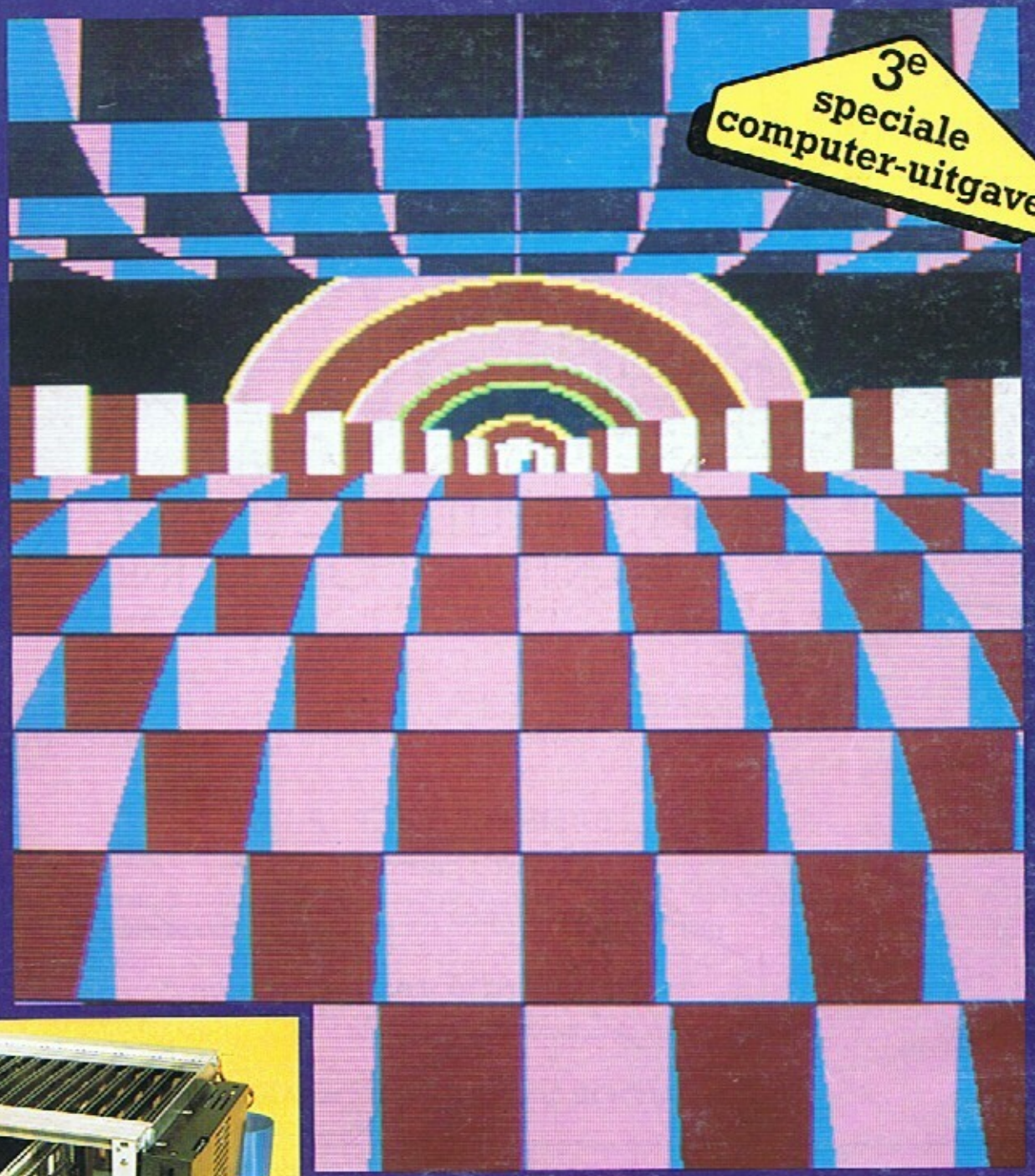
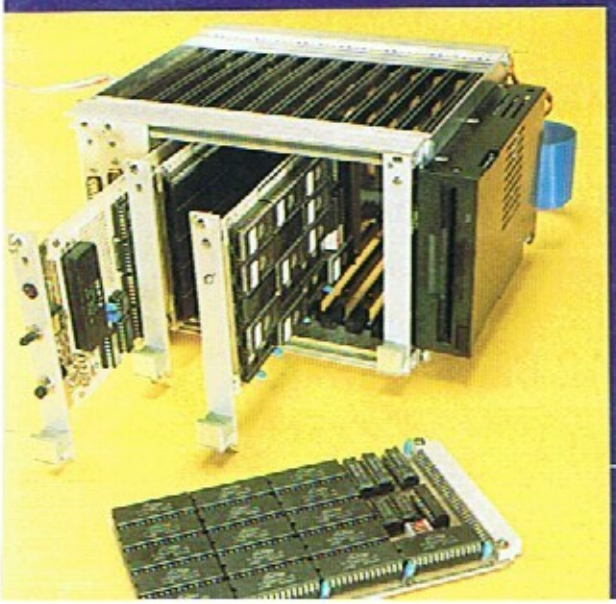
f 18,75 / Bfrs. 370

elektuur extra

COMPUTING



3e speciale computer-uitgave

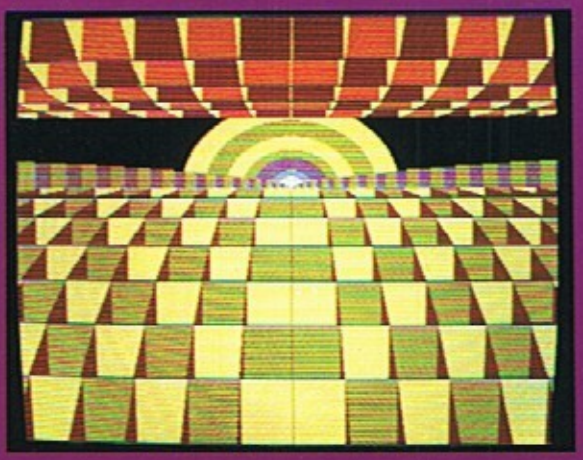


de super-computer: EC-68K

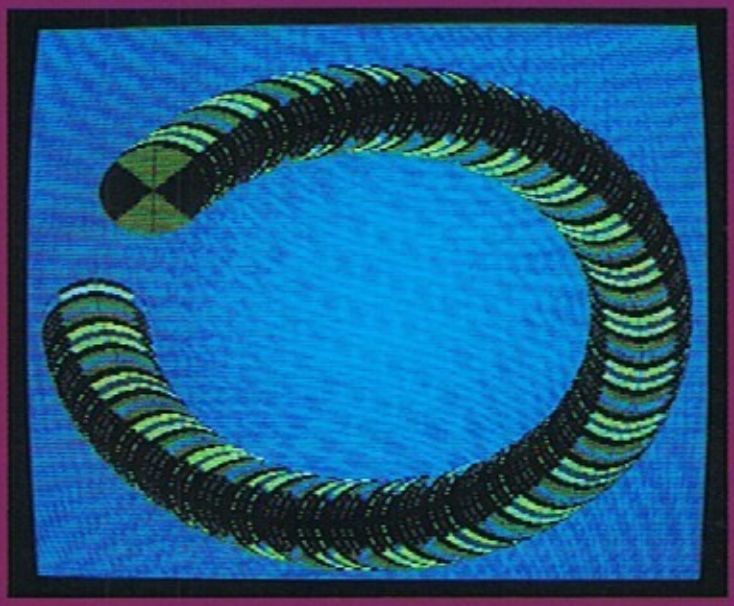
BASIC intern • Flex-computer

harddisks • IBM-kleurenkaart

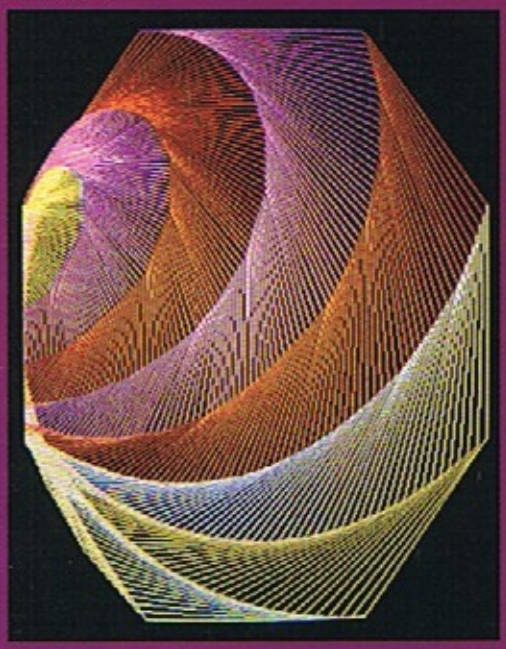
8 VASA



10. OCTANT



6. OKTA

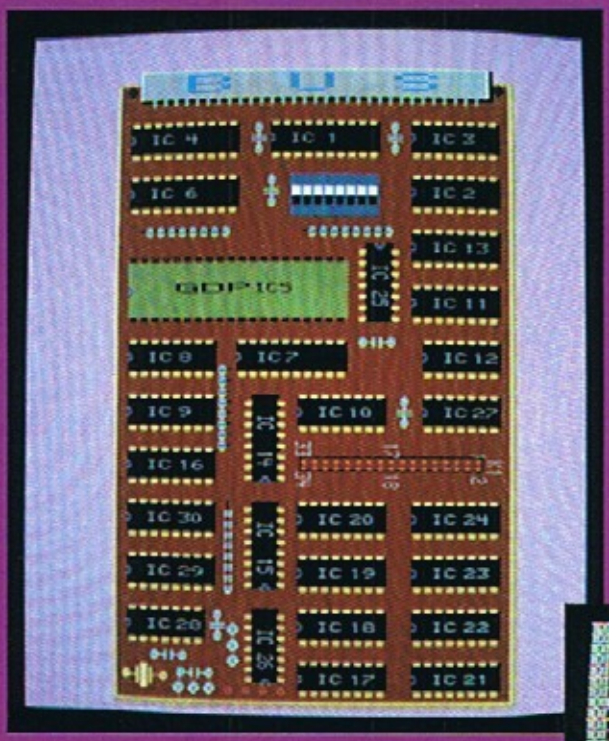


3. COLLAG

5. INUIT

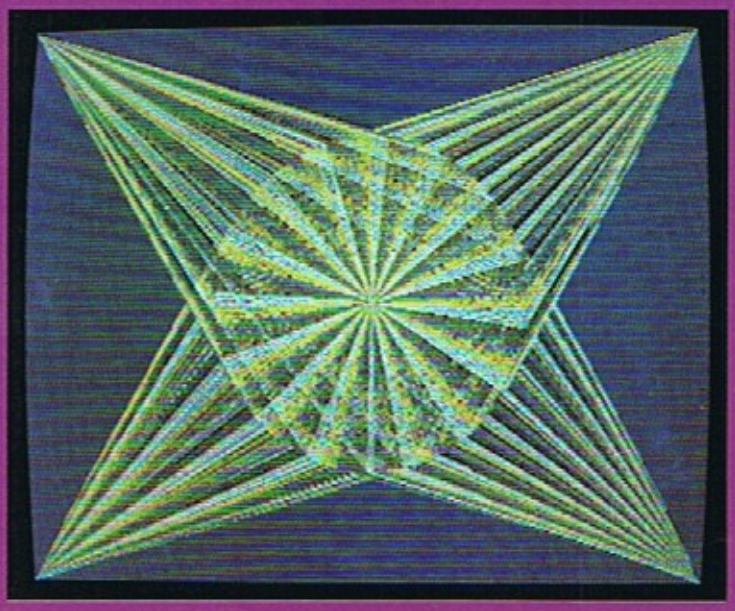
De bijbehorende programma's vindt u achter in deze uitgave

9. COVER



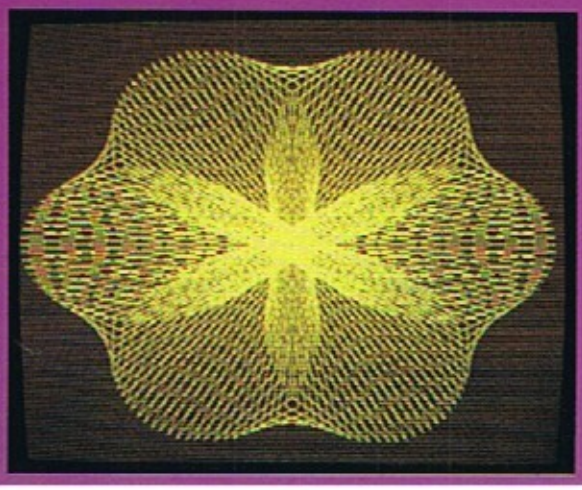
4. TESTXX

7. CON



2. DODEKA

1. SYMMET



De bijbehorende programma's vindt u achter in deze uitgave

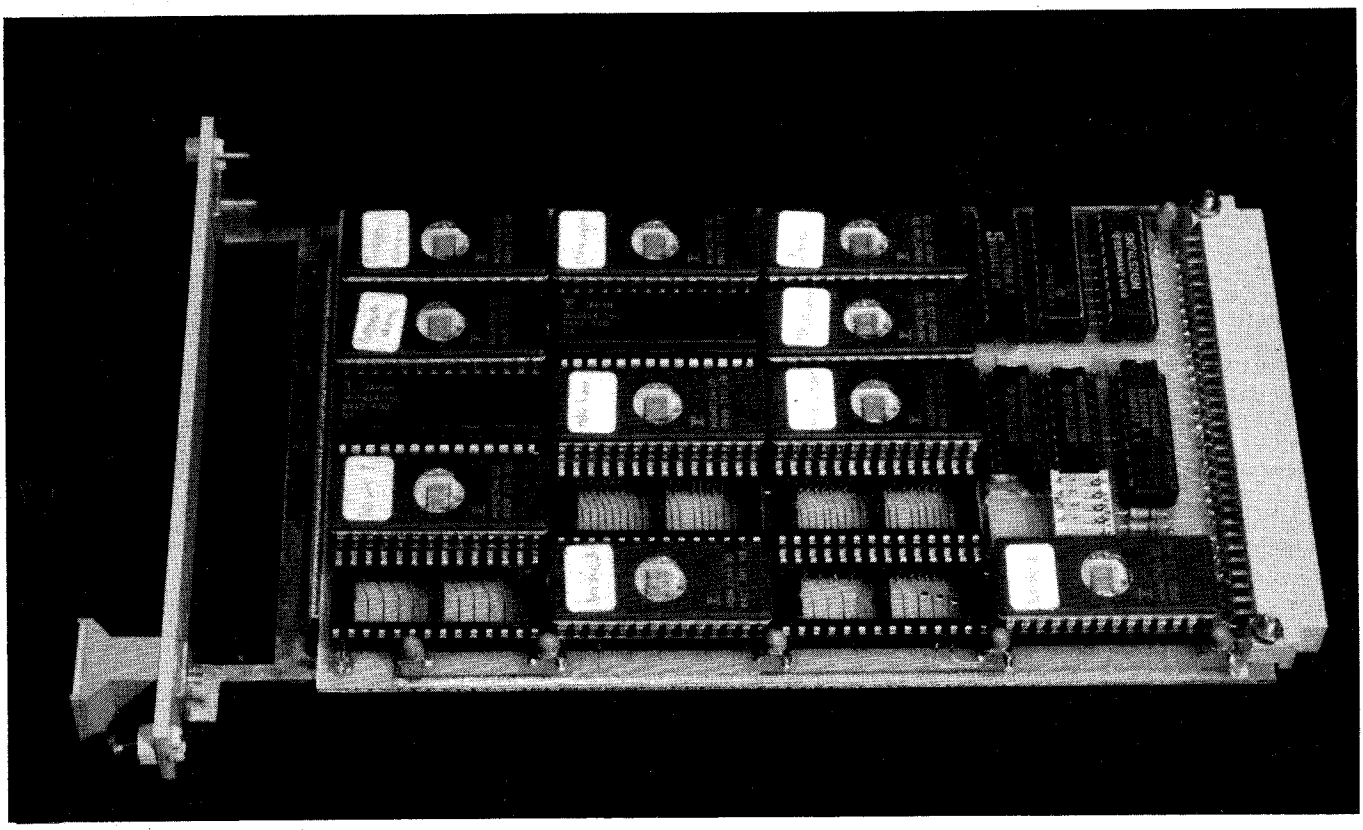
Redaktioneel

keus genoeg

Dit is nu alweer de derde uitgave van Elektuur Computing. In deze eerste drie nummers van het computer-blad voor zelfbouw hebben we een aantal zelfbouw-computers voorgesteld die zeker gezien mogen worden. Vijf in totaal zijn dat er. En er volgen nog meer. Maar we sommen die vijf nog even hier op:

* De EC-68K. Een 16-bitter met twee CPU's en veel software.
* De Octopus 65. Een ijzersterk 6502-ontwerp met een zeer snel DOS en veel grafische mogelijkheden.
* De EC-68. Een goedkope Flex-computer voor elke beurs.
* De Big-Board-II (BB II). Een CP/M-kameleon dat meer dan 40 verschillende floppy-formaten kan lezen.
* De IBM-PC-compatible. Een van de best verkochte zelfbouw-PC's.

Keus genoeg, dachten we zo. En er volgen binnenkort nog meer computers. We denken hierbij aan de EC-65K (verwarrend hè, al die EC-aanduidingen, maar dat went snel!), een opvolger voor de Octopus 65, upwards compatible met zijn voorganger. Deze computer werkt met de nieuwe, op 4 MHz draaiende 65SC816-microprocessor, die binnenkort leverbaar zal zijn op de Nederlandse markt.
Nog een nieuwtje voor de volgende nummers: we gaan ons intensiever bezig houden met toepassingen voor en het (nuttige) gebruik van de computer. Daarbij wordt speciaal aandacht besteed aan de periferie die hierbij nodig is. Op deze wijze blijft Elektuur Computing ook interessant voor degene die niet zelf een computer wil bouwen, maar zijn kant-en-klaar gekochte computer wel zelf wil uitbreiden met allerlei nuttige hard- en software.
Ook zullen we in de toekomst aandacht blijven besteden aan theoretische zaken, zoals in dit nummer "BASIC intern" en "de 68000 kort en bondig".
Maar nu weer terug van de toekomst naar het heden. U zult zien dat we ook in dit nummer weer een veelheid aan zelfbouwcomputers, software en beschrijvende artikelen hebben gestopt, waar ook voor u beslist iets interessants bij zal zitten. Lezen maar!



INHOUD

Elektor Computing

Deze Elektuur Computing is een uitgave van:
Uitgeversmij. Elektuur B.V.,
Peter Treckpoelstraat 2-4, Beek (L)
Telefoon: 04402-89444, Telex 56617
Korrespondentie-adres: Postbus 75,
6190 AB Beek (L)
Kantoortijden: 8.30-12.00 en 12.-16.00 uur
Direkteur: J.W. Ridder,
Bourgognestraat 13a, Beek (L)

Deze uitgave is samengesteld door de redaktie van het elektronicamaandblad Elektuur.
Medewerkers: A. Nachtmann, P. v.d. Linden, F. Aubert, W. Bascik, G. de Cuyper, B. Fasel, P. Lavigne, D. Meyer, N. de Vries.

Auteursrecht
Niets uit deze uitgave mag verveelvoudigd en/of openbaar gemaakt worden door middel van druk, fotokopie, mikrofilm of op welke wijze dan ook, zonder voorafgaande schriftelijke toestemming van de uitgeefster.
De auteursrechtelijke bescherming van deze Elektuur Special strekt zich mede uit tot de illustraties met inbegrip van de printed circuits, evenals tot de ontwerpen daarvoor.
In verband met artikel 30 Rijksoktrooiwet mogen de in deze uitgave opgenomen schakelingen slechts voor partikuliere of wetenschappelijke doeleinden vervaardigd worden en niet in of voor een bedrijf.
Het toepassen van schakelingen geschiedt buiten de verantwoordelijkheid van de uitgeefster.

© Uitgeversmaatschappij Elektuur B.V. - 1986
Printed in the Netherlands. ISSN 0013-5895

De hieronder vermelde printen kunnen worden besteld via de handel en rechtstreeks bij Elektuur B.V., Beek (L).
(E)PROM's kunt u door Elektuur B.V. laten programmeren.
Uitsluitsel over de bestelwijze en verkrijgbaarheid van onderstaande produkten geeft het overzicht in de laatste uitgave van het elektronicamaandblad Elektuur (pag. 6).

PROGRAMMEER SERVICE

bestelnr.	guldens	Bfrs.	programma
540	28,—	552	Flex-boot-ROM in 1 × 2716
541	28,—	552	Assist-09 in 1 × 2716
542	44,—	867	EC-68-karakter-generator in 1 × 2732

PRINT SERVICE

bestelnr.	guldens	Bfrs.	omschrijving
85210	47,35	933	EC-68-CPU/DRAM-kaart
85211	47,35	933	EC-68-CRT/FDC-kaart
83014			statische RAM-kaart

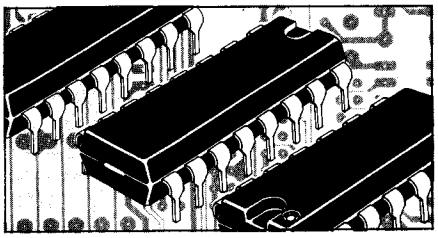
HARDWARE

SOFTWARE

KNOW-HOW

PROGRAMMA'S

HARDWARE



harddisks: snelle massageheugens

In de grote computercentra zijn ze al zo'n 20 jaar het belangrijkste massageheugen : De schijvengeheugens ofwel harddisks. Er worden zowel vaste als verwisselbare schijven toegepast. Bij deze laatste is de schijf of het schijvenpakket uitneembaar en net als bij de floppy disk te vervangen door één of meerdere schijven. Wat een schijvenpakket precies is, wordt in het artikel "harddisk-technologie" eveneens in deze uitgave verklaard. De schijfsystemen, zoals die in een computercentrum zijn te vinden, hebben nogal forse afmetingen. Schijfdiameters tussen 35 en 120 (!) centimeter zijn gangbaar, hoewel de grootste diameters ook hier ondertussen enigszins uit de mode beginnen te raken. Want ook voor schijfgeheugens geldt: steeds kleiner, steeds meer opslagkapaciteit per vierkante centimeter en natuurlijk ook steeds goedkoper.

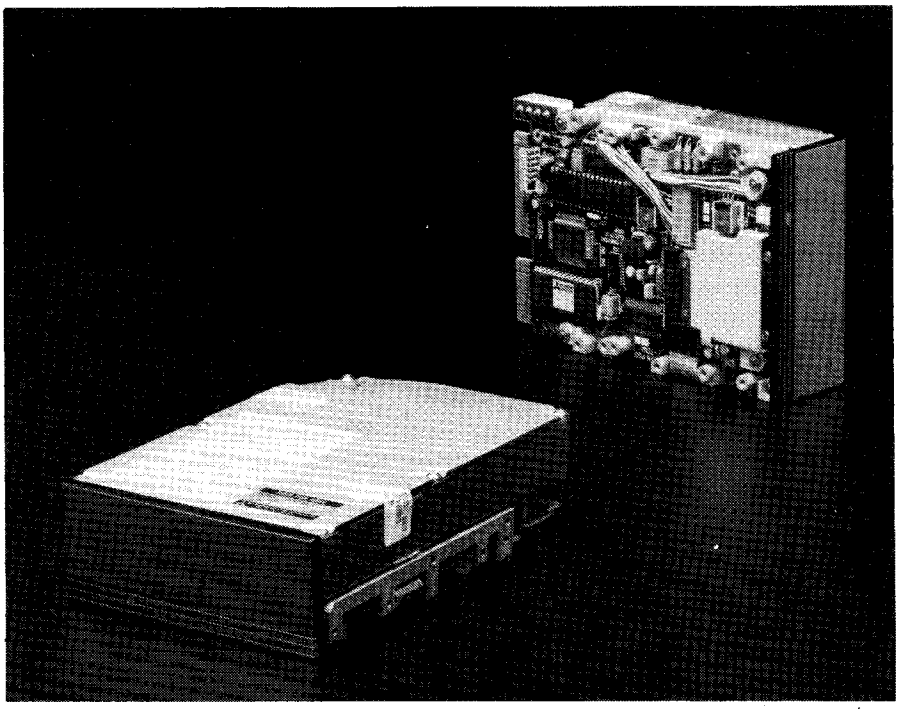
Dit type schijf begint dus interessant te worden. Waren de al bijna hanteerbare 8"-schijfsystemen nog onbetaalbaar voor de serieuze hobbyist, dit is al niet meer van toepassing op de sinds enige jaren verkrijgbare 5 1/4"- harddisks. Het afgelopen jaar heeft de 10-Mbyte-harddisk de 3000-gulden-grens (dalend) gepasseerd en het ziet er naar uit dat de 20-MByte-drive dit jaar hetzelfde gaat doen. De noodzakelijke controller is zelfs bij deze prijzen inbegrepen. Bovendien gaan de ontwikkelingen verder, inmiddels is de 3,5"-harddisk op de markt verschenen. Deze mikroschijven hebben inmiddels ook al een opslagkapaciteit tussen de 5 en 30 MbByte. De prijzen liggen in dezelfde orde van grootte als bij de 5 1/4"-types met gelijke kapaciteit. De tendens is nog steeds dalend, dus wellicht wordt de 1500-gulden-barrière dit jaar nog doorbroken.

Tot nu toe worden harddisks alleen in mainframes en PC's uit de hogere prijsklassen toegepast. Dat zou wel eens snel kunnen veranderen. Je hoeft geen profeet te zijn om te kunnen voorspellen dat in de niet al te verre toekomst een harddisk tot de standaard-uitrusting van een normale PC zal behoren, net zoals dat nu geldt voor een floppy-disk-drive. En niet alleen dat: ook de home-computer zal spoedig de harddisk als standaard-uitbreidingsmogelijkheid krijgen. De floppy-disk-drive behoort al tot de standaardmogelijkheden, dus wat volgt is wel duidelijk. Verderop in dit artikel komen we daar nog op terug.

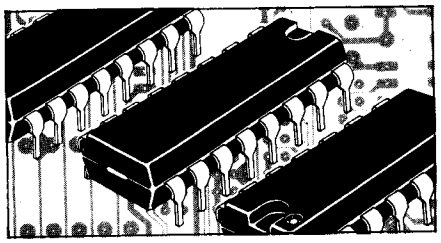
Tot nu toe hebben we het hoofdzakelijk over prijzen gehad, dan volgt nu een stukje techniek. In mechanisch opzicht is een harddisk het beste te vergelijken met een floppy-disk-drive. Het wezenlijke verschil is echter dat bij een "kleine" harddisk het geheugenmedium — de schijf — niet kan worden verwisseld. Dit ogenschijnlijke nadeel maakt de grote voordelen van een harddisk nu juist mogelijk: een grote opslagkapaciteit en korte toegangstijden. De belangrijkste redenen hiervoor zijn de hogere draaisnelheid van de harde schijf tegenover de floppy en het grotere aantal sporen per schijf. Het feit dat vaak meerdere dubbelzijdige schijven op een as worden gemonteerd, doet de opslagkapaciteit natuurlijk enorm toenemen. Op deze en andere technische details wordt nader ingegaan in het aansluitende artikel. De technologie van de harddisk heeft zich inmiddels zover ontwikkeld, dat de drives vrijwel "onkwetsbaar" zijn geworden. Althans voor bekende storingsoorzaken, zoals het uitvallen van de voedingsspanning of een foutieve besturing. Natuurlijk kunnen door dergelijke fouten gegevens op de schijf beschadigd worden, maar de drive zelf zal, in tegenstelling tot wat vroeger het geval was, geen schade oplopen. Een prettige wetenschap, indien men van plan is een harddisk toe te passen.

We hebben het al eerder opgemerkt: alleen een harddisk-drive is niet voldoende. Er is ook nog een speciale "controller" nodig met de bijbehorende software. Ook hier zijn de prijzen dalend en moder-

Foto: Een nieuwe tak van de grote harddisk-familie. De 3,5"-micro-harddisk-drives van Mitsubishi. "Micro" zijn hier alleen de afmetingen : even klein als een 3,5"-floppy-disk-drive. De kapaciteit laat weinig te wensen over. De MR 321 heeft met slechts één plaat een opslagkapaciteit van 12,75 Mbyte. Broertje MR 322 heeft twee plaatjes en kan maar liefst 25,5 Mbyte herbergen.



ne besturingssystemen, zoals bijvoorbeeld MS-DOS, hebben de gewenste besturingsroutines al "ingebouwd". De IC's die speciaal voor harddisk-controllers ontwikkeld worden, maken eveneens een stormachtige ontwikkeling door. Met slechts enkele VLSI-chips is het nu al mogelijk een universele controller op eurokaartformaat te maken, die floppy- en harddisk-controller op één kaart verenigt. Voor de zelfbouwende lezer wellicht nog interessanter: Door de inmiddels ontwikkelde DPLL-IC's (DPLL =Digital Phase Locked Loop) vervalt de zeer gekompliceerde afregeling van een harddisk-controller, die tot nu toe de zelfbouw haast onmogelijk maakte. De DPLL wordt gekombineerd met een aantal vertragingslijnen (delay-lines), waardoor afregeling overbodig wordt. Om toch weer even tot de werkelijk van vandaag terug te keren: de IC's zijn er, maar ze zijn nog niet in grote aantallen op de markt. En het ontwikkelen en testen van een harddisk-controller voor zelfbouw zal zeker nog enige tijd op zich laten wachten. Maar de controller voor zelfbouw wordt binnen afzienbare tijd een haalbare kaart.
Er is alweer een nog verdergaande ontwikkeling: harddisk-drives met ingebouwde controller. In dit geval is de controller-schakeling samen met de besturingselektronica een deel van de drive. De eerste "all-in"-drives schijnen binnenkort op de markt te komen : Geformatteerde kapaciteit 10 Mbyte, formaat 5 1/4" en een prijskaartje van rond de 2000 gulden. Het wordt dus al minder interessant om de soldeerbout op te warmen! Wat is er dan nog wel nodig ? Een snelle parallelle poort met wat handshake-mogelijkheden en wederom de onontbeerlijke software zijn voldoende. In de controller is echter al zoveel intelligentie aanwezig, dat de besturingssoftware naar verhouding eenvoudig is. Een C64 met harddisk behoort eind 1986 misschien al tot de realiteit.
Het enige nadeel van deze "all-in"-drives is, dat men aan één bepaald type vastzit. Weliswaar zijn 10 Mbyte niet weinig, maar ook die kunnen opgebruikt worden. Bijvoorbeeld wanneer er grote hoeveelheden grafieken punt voor punt op de schijf moeten worden gezet, om deze informatie weer snel op te kunnen roepen. Voor deze en soortgelijke gevallen is het handig, wanneer een aparte controller-kaart beschikbaar is, waarop eventueel harddisks van verschillende kapaciteit kunnen worden aangesloten. Een aansluitmogelijkheid voor een tweede drive is ook wenselijk.
Verder zullen we hier niet over dit onderwerp uitwijden. Het is wel duidelijk, dat wie zich met moderne computertechnologie bezighoudt, niet om het hoofdstuk "harddisk" heen kan. Deze uitgave is daarom voor een groot deel gewijd aan dit opslagmedium. Direkt aansluitend op deze inleiding worden een aantal basisbegrippen behandeld in het artikel "harddisk-technologie" en wordt u tevens een kijkje in het inwendige van de harddisk-drive gegund. Een volgend artikel beschrijft hoe een harddisk-drive op de "Big-Board-II (de CP/M-zelfbouwcomputer uit het tweede nummer van Elektuur Computing) kan worden aangesloten. Om het geheugen nog even op te frissen: Big-Board is met een zogenaamde SASI-bus uitgerust, waarop direkt een Xebec-controller kan worden aangesloten. Op deze controller wordt dan weer de drive aangesloten en klaar is Kees! In ieder geval wat de hardware betreft. De software wordt in het genoemde artikel eveneens behandeld. Wie nu nog niet weet wat de kreten SASI en Xebec-Controller zoal inhouden, is na het lezen van de beide genoemde artikelen voldoende, zoniet helemaal op de hoogte. "Onze" andere computers worden natuurlijk ook niet vergeten. De IBM-PC-Compatible is waarschijnlijk als volgende aan de beurt, namelijk al in de volgende EC-uitgave. Bij de Octopus 65 moet nog worden bekeken welke van de beide beschreven mogelijkheden (controller voor zelfbouw of een harddisk-drive met ingebouwde controller) de voorkeur verdient. In ieder geval is hiervoor nog enige ontwikkelingstijd nodig. Dat geldt in principe ook voor de in deze uitgave beschreven "EC-68" en "EC-68K".
Eén ding is duidelijk. Het is de bedoeling dat iedere tot nu toe (in EC) gepubliceerde computer voor zelfbouw de mogelijkheid krijgt met een harddisk-drive te worden uitgebreid. En indien U er in slaagt een C64, Acorn, enz., uit te breiden met een "all-in"-drive, neem dan eens kontakt op met de redaktie. Een publikatie in EC of Elektuur kan voor C64-, etc.-kollega's interessant zijn.



harddisk-technologie

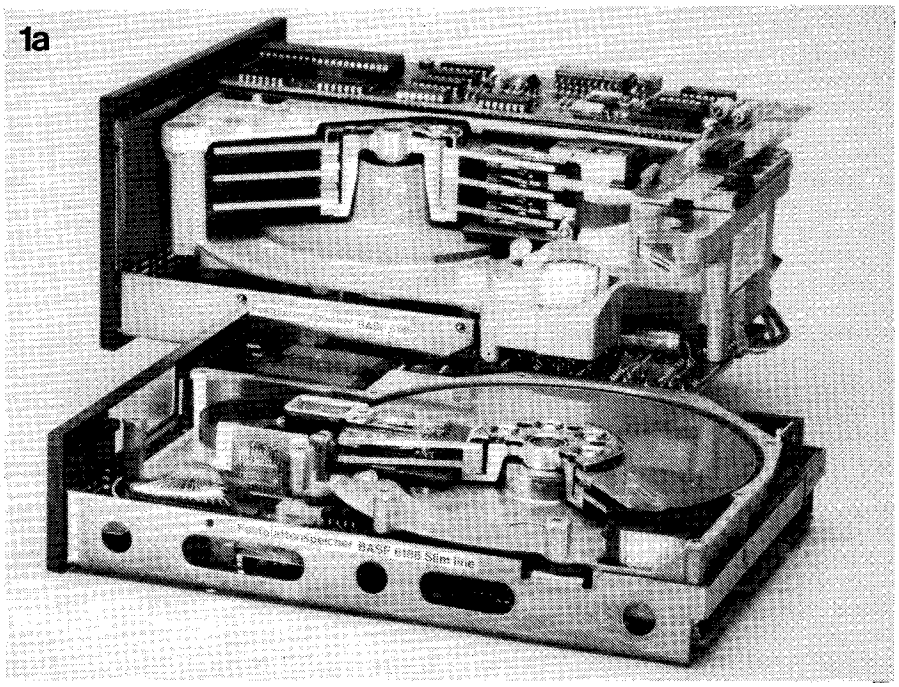
Harddisks waren tot nu toe wegens hun hoge aanschaffingskosten vrijwel uitsluitend te vinden in grote computersystemen (mainframes) en minicomputers. De steeds hogere eisen die aan microcomputers worden gesteld en de dalende prijzen hebben er echter toe geleid, dat steeds meer PC's met een harddisk worden uitgerust. In vergelijking met de gangbare opslagsystemen vertonen harddisks een aantal nieuwe technologische details, waarop in dit artikel nader wordt ingegaan.

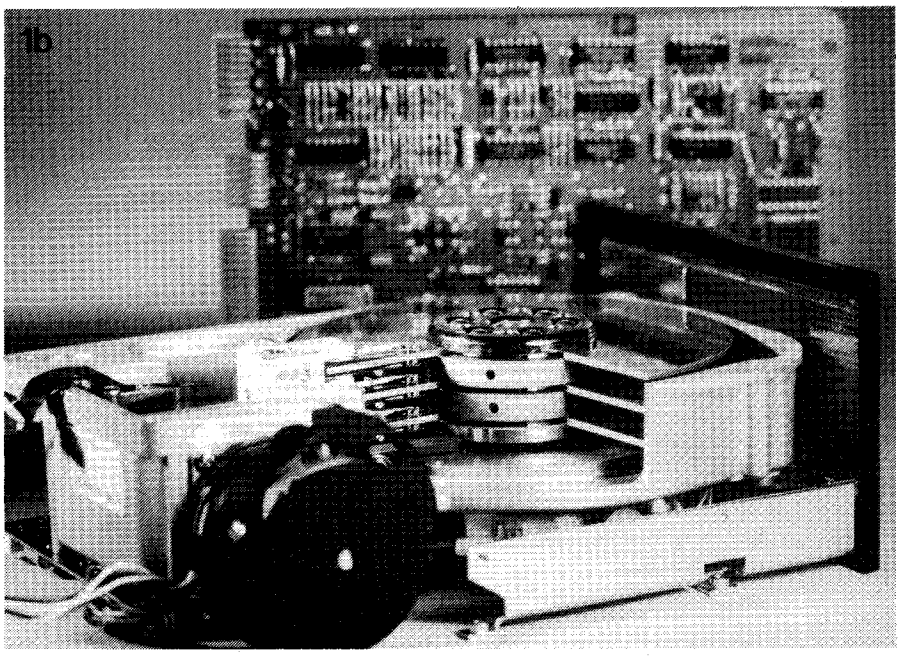
Het opslaan van gegevens in een microcomputersysteem stelt de gebruiker voor een zeer gevarieerde keuze. Aan opslagmedia staat momenteel nogal wat ter beschikking: ROM's, diverse typen RAM's, bandgeheugens, floppy disks, bubble-memories, en natuurlijk de harddisk. De meest universele eigenschappen hebben duidelijk de RAM's: Ze zijn zeer snel programmeerbaar en leesbaar. Als nadeel staat daar tegenover dat de informatie niet transportabel is, omdat een RAM na het uitschakelen van de voedingsspanning zijn geheugeninhoud verliest. Dit probleem is eventueel met batterijen op te lossen. Wanneer men echter de kosten per opslageenheid (bijv. kB) gaat berekenen, wordt meteen duidelijk dat deze manier van gegevensopslag een dure aangelegenheid is.
Op grond hiervan worden RAM's, een paar uitzonderingen daargelaten, alleen als tijdelijk geheugen toegepast. Wat betreft de kosten en de kapaciteit per chip steekt de EPROM aanzienlijk gunstiger af. Sinds kort zijn er EPROM's van het type 27512 in de handel, die een opslagkapaciteit hebben van 512 Kbit = 64 Kbyte. Dat is dus het komplete adresseerbereik van de doorsnee 8-bit-processor, zoals bijvoorbeeld een Z-80 of 6502. Voor permanente opslag zijn EPROM's gezien de lage prijs en de snelle toegankelijkheid zeker de beste keus. Maar dan ook alleen voor deze specifieke toepassing. Wanneer regelmatig programma's of gegevensbestanden moeten worden gewijzigd, valt de EPROM op grond van de tijdsintensieve en omstandige programmeerprocedure direkt af. Hetzelfde geldt op dit moment voor de optische geheugenmedia, die — wat het opslaan van de gegevens betreft — met de compact-disc vergelijkbaar zijn. De werkelijk fabuleuze opslagmogelijkheden van de optische schijf, tot in het gigabyte-gebied, doen ons echter hopen dat dit in de toekomst nog verbetert.
Terug nu naar de realiteit. Menigeen die zich als hobby met de computer bezighoudt, zal zijn eerste programma's op een cassette-bandje hebben opgeslagen. Deze meestal nogal schamper bekeken manier van gegevensopslag kan uitstekend worden toegepast wanneer grote hoeveelheden gegevens op goedkope wijze moeten worden bewaard. Dat gaat dan wel ten koste van een snelle toegan-

kelijkheid. Het zoeken naar gegevens op een magneetband vergt nogal wat tijd, maar er is een belangrijke toepassing voor deze manier van opslag, waarbij deze beperking op de koop toe genomen wordt: de zogenaamde "backup". In geval van nood kan zo'n veiligheidskopie de redding betekenen van maandenlange programmeerarbeid. Beslist goedkoper dan een verzekering, als die al bestaat voor dit soort kalamiteiten.
Een verder uitontwikkelde versie van de gewone cassette-recorder is momenteel in de handel verkrijgbaar onder de naam "tape streamer". De cassettebandgeheugens hebben meestal een opslagkapaciteit tussen de 20 en 40 Mbyte. De specifieke toepassing van deze bandgeheugens ligt in het maken van veiligheidskopieën van een komplete harddisk. In een meer professionele uitvoering hebben deze tape streamers meestal een opslagkapaciteit die nog vele malen hoger ligt dan hierboven werd genoemd.
De meeste personal computers maken gebruik van de floppy-disk als opslagmedium. De redenen hiervoor zijn de betrouwbaarheid, de bij niet al te grote bestanden redelijk korte toegangstijd en het gemakkelijk te transporteren opslagmedium, namelijk de diskette. Hierbij wordt dan niet alleen gedacht aan de diskette zelf, maar ook en vooral aan het "transport" van programma's en bestanden. Dit laatste punt gaat echter niet helemaal op. In de praktijk ontstaat vaak de indruk dat diskettes alleen uiterlijk overeenkomsten vertonen. De fabrikanten van software proberen hun produkt een "origineel tintje" mee te geven. Dit uit zich dan in de vorm van afwijkende data-formattering en verschillende opslagdichtheden, soms zelf binnen een en hetzelfde produkt. Een probleemloze uitwisseling van software per diskette is daardoor vaak niet mogelijk.

De harddisk: fijnmechanica...

In tegenstelling tot de floppy-disk is bij de harddisk-drive de diskette (een van een magnetisch gevoelige laag voorziene schijf) niet verwisselbaar. Dit lijkt op het eerste gezicht een nadeel, maar er vloeien een aantal geweldige voordelen uit voort. De vaste schijf is bijvoorbeeld veel beter af te stellen dan de floppy-disk. Deze laatste wordt niet voor niets ook wel eens flodderschijf genoemd. Met een vaste schijf zijn centrering en schijfhoogte zeer exakt in te stellen. Daardoor is het mogelijk de data veel dichter bij elkaar op het magnetische materiaal aan te brengen. Op één 5 1/4"-schijf kan dan meer dan 10 Mbyte worden ondergebracht. Door meerdere (x) schijven mechanisch gekoppeld op dezelfde as aan te brengen, wordt de opslagkapaciteit nog eens met een faktor "x" vermenigvuldigd. Een dergelijke konfiguratie wordt schijvenpakket genoemd (figuur 1). Vooral in de wat grotere computersystemen, zoals mini-computers, worden vrijwel uitsluitend schijvenpakketten toegepast. Figuur 1a toont het normaliter ontoegankelijke innerlijk van twee moderne harddisk-drives. De BASF 6185 heeft een pakket van drie schijven (boven) en een (ongeformatteerde) opslagkapaciteit van 27 MByte.

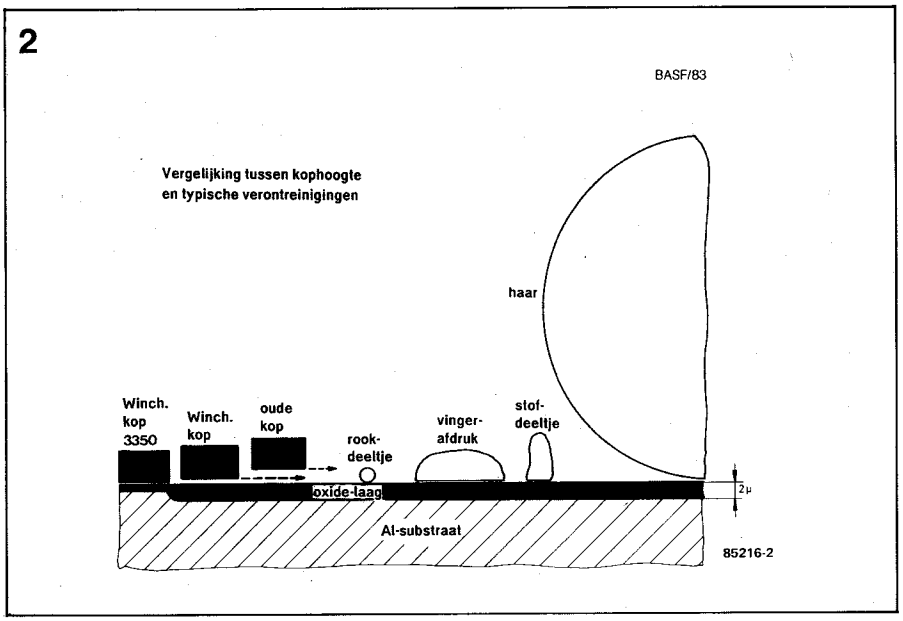




Figuur 1. Op deze foto's in het normaliter ontoegankelijke binnenste van twee moderne harddisk-drives te zien. Figuur 1a toont een BASF 6185 met drie schijven en normale afmetingen, plus een BASF 6188 met twee schijven en slim-line afmetingen. Figuur 1b toont de BASF 6185 nog eens vanuit een andere gezichtshoek. Op de voorgrond is de stappenmotor duidelijk herkenbaar.

Daaronder in "slimline"-uitvoering de BASF 6188, deze drive heeft slechts twee platen en een kapaciteit van 15 MByte ongeformatteerd. Figuur 1b toont eveneens de 6185, maar op een andere wijze "opengesneden". Hier komt een belangrijk detail naar voren: de schijven zijn dubbelzijdig, bij elke schijf behoren twee koppen. Alle koppen zijn op een gemeenschappelijke houder, de "kam", gemonteerd en bewegen dus gelijktijdig. Zoals gezegd is de harddisk bijzonder geschikt voor de opslag van grote hoeveelheden data. Ter vergelijking: een opslagkapaciteit van 30 Mbyte is voldoende om ongeveer 10000 vellen typemachinetekst van 80 x 40 tekens op te slaan. Dergelijke hoeveelheden komen bij privégebruik zelden voor, zodat het niet verwisselbaar zijn van de schijven op de koop toe kan worden genomen.
De grote hoeveelheid data op een harddisk heeft echter wel een aantal hogere eisen tot gevolg. In vergelijking tot een floppy-disk is het wenselijk dat de schrijf- en leessnelheden aanzienlijk hoger komen te liggen. Beide eisen zijn alleen door een veel hogere draaisnelheid te verwezenlijken. Dat heeft dan weer tot gevolg dat de lees-/schrijfkop niet in direkt kontakt met de schijf mag staan. Door de anders onvermijdelijke mechanische slijtage zouden zowel kop als schijfoppervlak binnen zeer korte tijd versleten zijn. Een luchtspleet tussen de kop en de schijf heeft echter weer tot gevolg dat de haalbare datadichtheid gereduceerd wordt. Hier ontstaat dus een vicieuze cir-

kel, die men alleen met een kompromis kan verlaten.
Het is gebleken dat de afstand tussen kop en schijf minder dan één mikron (= één miljoenste meter) zou moeten zijn om een redelijk kompromis te bereiken. In figuur 2 wordt deze problematiek verduidelijkt. In het binnenste van een harddisk-drive moet het extreem stofvrij zijn. Dit is bij de geheel open floppy-disk absoluut onmogelijk. De stofvrij-eis houdt in dat het gehele produktieproces tot de montage in een stofdichte behuizing in stofvrije ruimtes dient te gebeuren, terwijl bovendien speciale eisen worden gesteld aan de kleding van de produktiemedewerkers (figuur 3). Daarmee is echter nog niet verklaard hoe de lees-/schrijfkop stabiel op 1 mikron boven de schijf kan worden gerealiseerd. De oplossing kwam in de zeventiger jaren van IBM met de introduktie van de Winchester-technologie. De sleutel tot de oplossing van het probleem lag in het benutten van de luchtstroming boven de snel roterende schijf. Door de kop een speciale vorm te geven vormt zich tussen kop en schijf een luchtkussen, waarop de kop met behulp een precisiegeleiding kan zweven. De vereiste 1 mikron is zo bereikbaar. De nieuwste generatie lees-/schrijfkoppen laat zelfs vlieghoogtes van 0,2 mikron toe.
Het is duidelijk dat hier extreme eisen ten aanzien van de nauwkeurigheid worden gesteld. In het bijzonder het kompenseren van de thermische eigenschappen van de materialen (uitzettingskoëfficiënt) was lange tijd een groot probleem. Het gebruik van deze luchtkussenmetode stelt verder een vrijwel ideale oppervlaktestruktuur voorop. Elke oneffenheid is direkt van invloed op de stabiliteit en de vlieghoogte van de kop. Een licht doorhangen van de schijf (ten gevolge van de zwaartekracht) in de orde van tienden van een mikron leidt door de zeer kleine afmetingen van de kop tot variaties in de kop/schijf-afstand van enkele honderdsten mikrons. Bij een vlieghoogte van 0,2 à 0,3 mikron kan dit al amplitudevariaties van 10% tot gevolg hebben.
De aandrijving wordt meestal verzorgd door een gelijkstroommotor voor de schijf respektievelijk het schijvenpakket, en een stappenmotor of een "voice coil" voor de positionering van de kam binnen het schijvenpakket.
Een beslissende faktor voor de levensduur van een harddisk-drive is, dat de ruimte waarin de schijf draait ook na de montage nog vele jaren stofvrij moet blijven. Daarom wordt de lucht in het binnenste van de drive konstant gefilterd. Alle eventueel door wrijving of kop/schijfkontakt losgewerkte deeltjes (zie nogmaals figuur 2) worden op deze wijze uit de cirkulerende lucht verwijderd. Door een aangepaste konstruktie van het schijvenhuis kan de pompwerking van de schijven worden benut om een frekwente doorstroming van de lucht te bewerkstelligen. Een zogenaamd ademend filter zorgt voor nivellering van de drukverschillen met de buitenlucht. Door een juiste plaatsing van dit filter kan men bereiken dat de druk in het schijvenhuis iets hoger is dan de atmosferische druk. In geval van een lekkage zal dus lucht van binnen naar buiten stromen, en niet in omgekeerde richting. Het binnendringen van deeltjes uit de buitenlucht wordt daardoor verhinderd. Een van de potentiële lekkageplaatsen, de as van de aandrijfmotor, wordt afgedicht met een magnetische afdichtvloeistof.

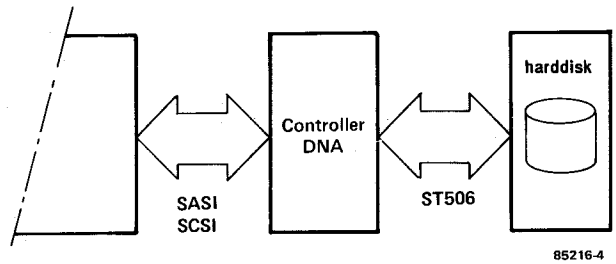


...en elektronica

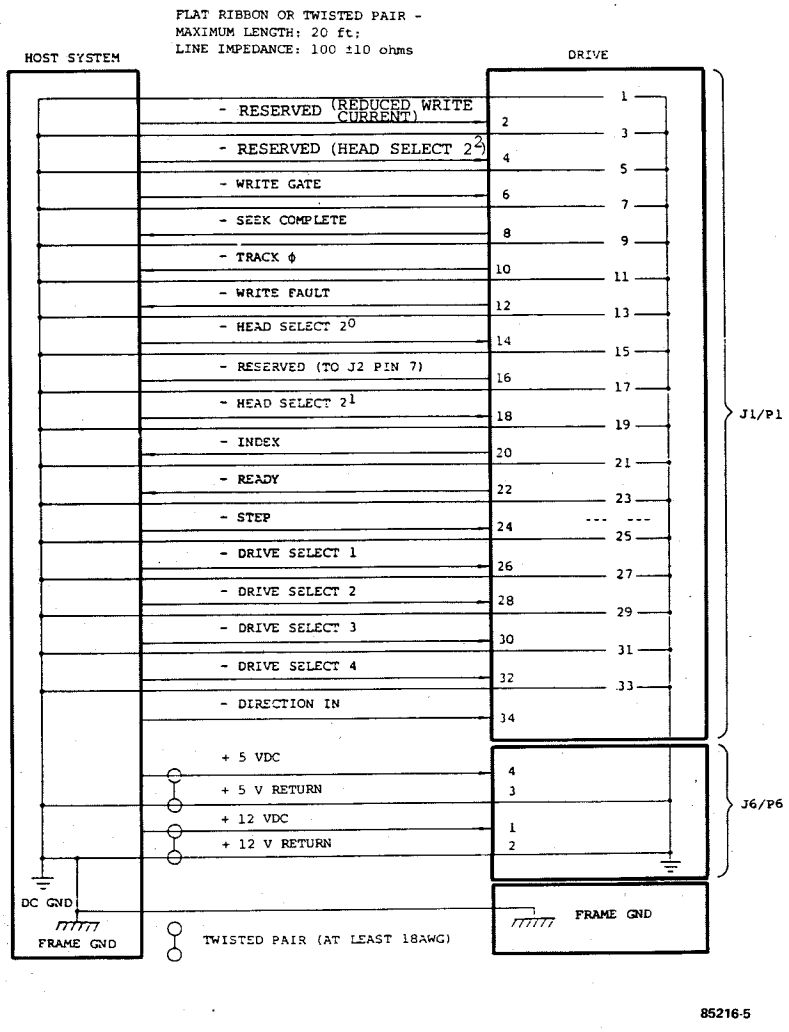
Achter de meestal sobere frontplaat verbergt zich, zoals uit het voorgaande is gebleken, een vebazingwekkend stukje mechanische technologie. Voor de gebruiker is het natuurlijk interessant om te weten hoe zo'n wonderding op zijn PC kan worden aangesloten. Behalve de harddisk-drive zelf is er ook nog een controller-module nodig, die niet alleen als interface naar de PC dient, maar ook diverse hulpfunkties voor het "management" van de data op de schijf voor zijn rekening neemt. De hoge omwentelingssnelheid maakt een dermate snel datatransport mogelijk, dat de verwerking van deze data door de centrale microprocessor erg inefficiënt zou verlopen. Daarom worden de belangrijkste en qua timing meest kritische funkties door de harddisk-controller zelf afgehandeld, zonder dat de microprocessor daarbij behoeft in te grijpen. Zulke funkties zijn bijvoorbeeld de herkenning van datablokken of het initialiseren van de drive. Het doelgerichte ontwerp van de controller verklaart waarom deze het snelle datatransport wel kan hanteren.
Ook bij het transport van de data tussen harddisk en werkgeheugen van de PC wordt zo veel mogelijk buiten de mikroprocessor om gewerkt. Meestal wordt hiervoor een DMA-IC gebruikt, dat door direkte toegang tot het geheugen (direct memory access) de data zeer snel kan overbrengen. Dit bespaart de omweg via de accumulator van de microprocessor, waardoor in een single-user-systeem een snelheidswinst van 200 tot 500% wordt bereikt. Figuur 4 toont een standaardkonfiguratie voor de kombinatie van PC, controller en harddisk-drive.
De interface tussen computer en controller wordt meestal aangeduid met de afkorting SASI (Shugart Associates Standard Interface) of SCSI (Small Computer-systems Standard Interface). De meeste fabrikanten hebben in hun eigen belang

de SASI als norm geaksepteerd. Hoe zo'n interface volgens deze pseudo-norm eruit ziet, is terug te vinden op pagina 56 van EC2, waar het SASI-gedeelte van de Big-Board-II staat afgedrukt. Ook de interface tussen controller en harddisk-drive heeft zich tot een uniforme standaard ontwikkeld. De meeste drives zijn tegenwoordig met een ST506-interface uitgerust. Deze interface vertoont duidelijke overeenkomsten met de interface voor floppydisk-drives. Ook hier hebben we weer te maken met een pseudo-norm. Het grote voordeel hiervan is dat de transmissiesnelheid, de kodering van de data en de opslagkapaciteit per spoor dwingend zijn voorgeschreven. Daardoor is de totale opslagkapaciteit van een drive uitsluitend afhankelijk van het aantal schijfoppervlakken en het aantal sporen per oppervlak. De aansluitvolgorde en de bijbehorende signalen van de ST506-interface zijn aangegeven in de figuren 5 en 6. Hieronder volgt een korte beschrijving van alle signalen van deze interface. Twee belangrijke opmerkingen vooraf: daar het hier niet om een norm gaat, maar om een pseudo-norm, zijn bij sommige drives detail-afwijkingen mogelijk. Verder geldt: alle signalen zijn "active low"!

4



5



Figuur 2. Uit deze afbeelding wordt direkt duidelijk waarom het binnenste van een harddisk-drive extreem stofvrij moet worden gehouden. Zelfs een rookdeeltje vormt een obstakel voor de op zeer kleine hoogte boven de aluminium schijf zwevende kop.

Figuur 3. "Klinisch schoon" is een wat simpele benaming voor de stofvrije ruimtes waarin de harddisk-drives worden gemonteerd. De produktiemedewerkers lijken hier wel wat op astronauten, alleen heeft hun kleding hier tot doel te voorkomen dat deeltjes uit de longen of van de huid in de werkruimte belanden. Om de werkruimte te bereiken moeten eerst meerdere sluizen worden gepasseerd. Tussen twee sluizen wordt dan de werkkleding aangetrokken.

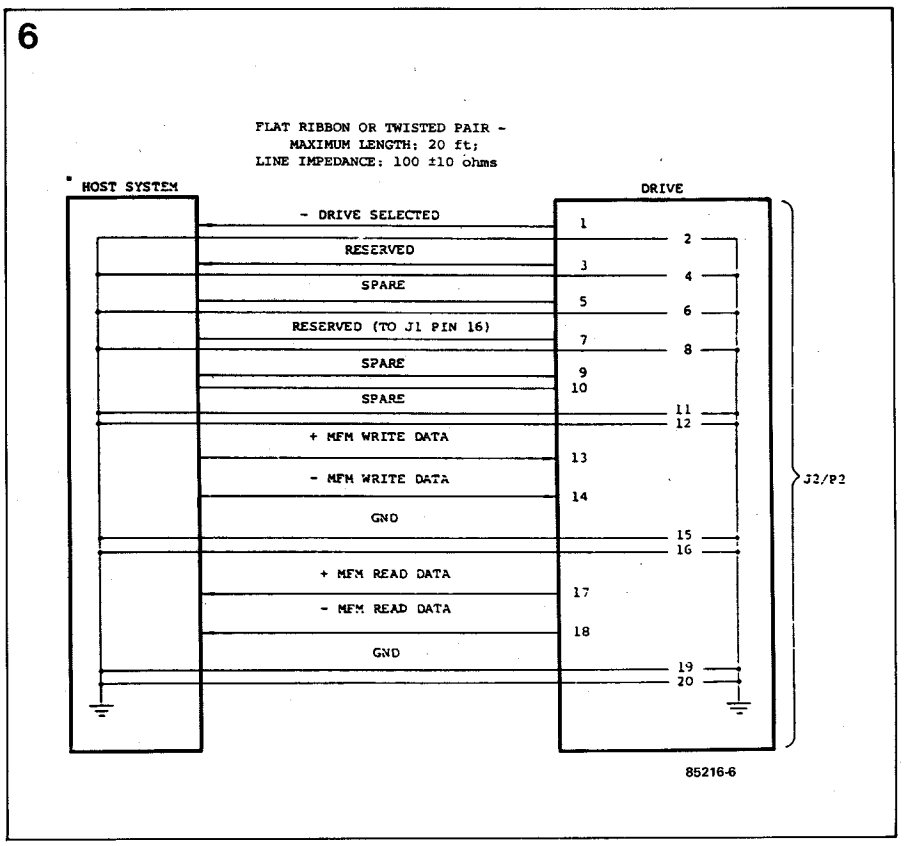
Figuur 4. De harddisk-controller vormt de verbinding tussen computer en harddisk-drive. De "intelligentie" van deze controller voorkomt dat iedere gewone PC volslagen "overwerkt" raakt bij de besturing van een harddisk-drive. De verwerkingssnelheid vormt daarbij het grootste probleem.

Figuur 5. De verbinding tussen controller en drive bestaat bij de ST506-interface uit twee bandkabels (flat-cables) . De hier met J1/P1 aangeduide konnektor voert de besturingssignalen (control-bus). De data worden via de tweede bandkabel verstuurd (zie figuur 6).

Tabel 1

Tabel 1. Met twee selektielijnen kunnen vier koppen worden gekozen. Drie lijnen zijn voldoende voor acht koppen. De tabel toont welke kop door welke nivo's op de selektielijnen wordt gekozen.

		Head Select 2↑0 (J1, Pin 14)	Head Select 2↑1 (J1, Pin 18)	Head Select 2↑2 (J1, Pin 4)
schijf 1	kop A (1)	Ø	1	1
	kop B (2)	1	1	1
schijf 2	kop A (3)	Ø	Ø	1
	kop B (4)	1	Ø	1
schijf 3	kop A (5)	1	1	Ø
	kop B (6)	Ø	1	Ø
schijf 4	kop A (7)	1	Ø	Ø
	kop B (8)	Ø	Ø	Ø



Figuur 6. De data worden via symmetrische verbindingen verzonden en ontvangen (dus twee paar aders). Daartussen liggen steeds weer massa-aders. De zogenaamde "Twisted pair GND" zou bij deze hoge frekwenties niet voldoende zijn (zie tekst).

Figuur 7. Dit blokschema toont hoe meerdere drives op een controller worden aangesloten. De hier als een aparte eenheid getekende data-separator maakt meestal deel uit van de controllerprint.

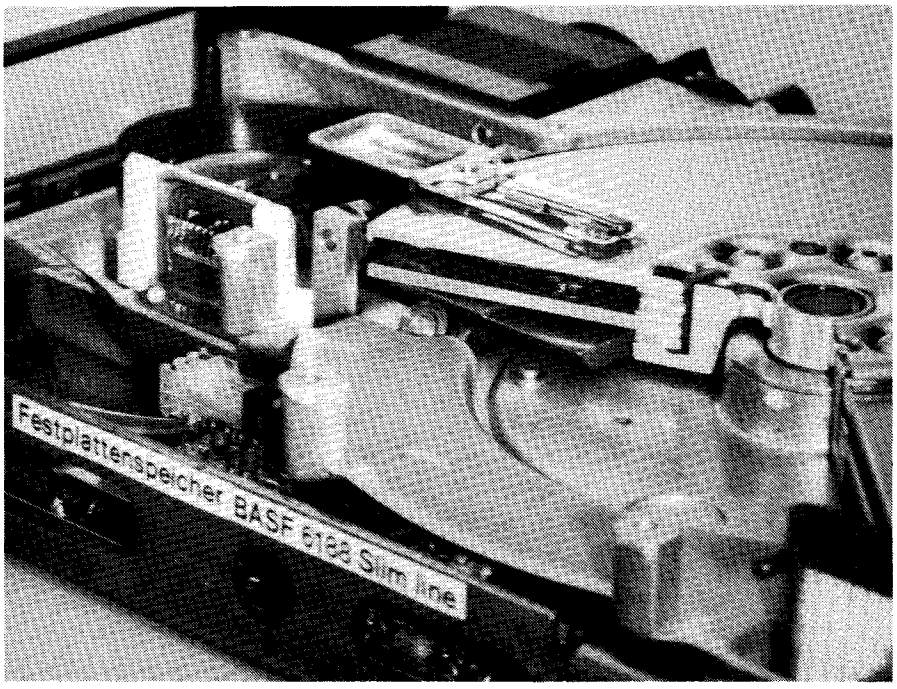
Figuur 8. Zo is de dataverbinding opgebouwd. In verband met de hoge transmissiesnelheid (5 Mbit/s) worden de data via symmetrische lijnen getransporteerd.

Track Ø (TRKØ):
Het signaal is "low" als de lees/schrijfkop zich op spoor nul bevindt, het buitenste spoor van de schijf.

Index (INDEX):
Dit signaal wordt iedere schijfomwenteling éénmaal aktief. Het dient om het begin van het spoor aan te geven.

Deze beide signalen vormen een essentiëel oriëntatiepunt voor het dataverkeer van en naar de schijf.

Head Select (HD SELØ...HD SEL 2):
De meeste harddisk-drives hebben meer dan één lees/schrijfkop. Ze kunnen met de head-select-signalen geselekteerd worden. Voor 4 koppen (= 2 schijven) zijn 2 signaallijnen voldoende, met 3 signaallijnen kunnen 8 koppen worden geselekteerd. Tabel 1 geeft hiervan een duidelijk beeld.



Drive Select 1 — 4 (DR SEL 1...DR SEL 4):
Een controller-IC kan meestal tot 4 drives besturen. De gewenste drive wordt gekozen met behulp van één der drive-select-lijnen. Een voorbeeld van een konfiguratie met meerdere harddisk-drives toont figuur 7.

Direction I (DIR IN):
Dit signaal bepaalt de bewegingsrichting van de "kam". Een logisch-nul-signaal laat de kop bij iedere "step"-impuls naar binnen, dus naar het midden van de schijf, bewegen. Een logisch-één-signaal heeft een beweging naar buiten, dus richting spoor Ø, tot gevolg (zie ook onder "step"). Het DIR IN-signaal mag niet van toestand wijzigen zolang het signaal 'Seek Complete' (zie hieronder) inaktief ("1") is.

Step (STEP) :
Iedere step-impuls verplaatst de koppenkam èèn positie naar binnen of naar buiten (zie bij DIR IN). Afhankelijk van de step-frekwentie (dus de snelheid waarmee de step-impulsen elkaar opvolgen) schakelen de meeste moderne drives automatisch om tussen de "single-step-mode" en de "ramp-mode". De BASF 6188-drive reageert bijvoorbeeld als volgt: wanneer de tijd tussen de step-impulsen tussen de 1,2 en 3,1 ms ligt, zal de kam gelijke tred houden met de step-impulsen : de single-step-mode. Ligt de tijd tussen de impulsen echter tussen de 10 en 200 μs en moet de de kam over een redelijk aantal sporen worden verplaatst, dan zal de kam zich met een bepaalde snelheid vloeiend naar het gewenste spoor bewegen. De kop kan zich daardoor in deze "ramp-mode" aanzienlijk sneller verplaatsen.
Het omschakelen tussen de beide modi wordt automatisch verzorgd door de drive-elektronica.

Seek Complete (SEEK COMPL):
Dit signaal wordt aktief, zodra de kam op het geselekteerde spoor is aangekomen

en de kop voldoende tijd heeft gekregen om te stabiliseren. Voor het laatste geldt een vaste tijd (inklusief veiligheidsfaktor), een gangbare waarde is 15 ms.

Write Gate (WRT GATE):
Het logische nivo op deze signaallijn bepaalt of er geschreven naar ("Ø") of gelezen van ("1") de schijf wordt.

Reduced Write Current (RED WRT CUR):
Op de binnenste sporen is de "vlieghoogte" van de koppen minder en de magnetische fluxdichtheid groter. Daarom wordt er op de binnenste sporen meestal met een lagere stroom (= reduced write current) geschreven. Bij de BASF 6188 met in totaal 360 cylinders wordt de RED WRT CUR-lijn vanaf spoor 256 geaktiveerd.

Open Cable Detect (OP CBL DET):
Beide konnektoren van de drive beschikken over een aansluiting waarmee de interne elektronica kan testen of de kabels juist zijn aangesloten.

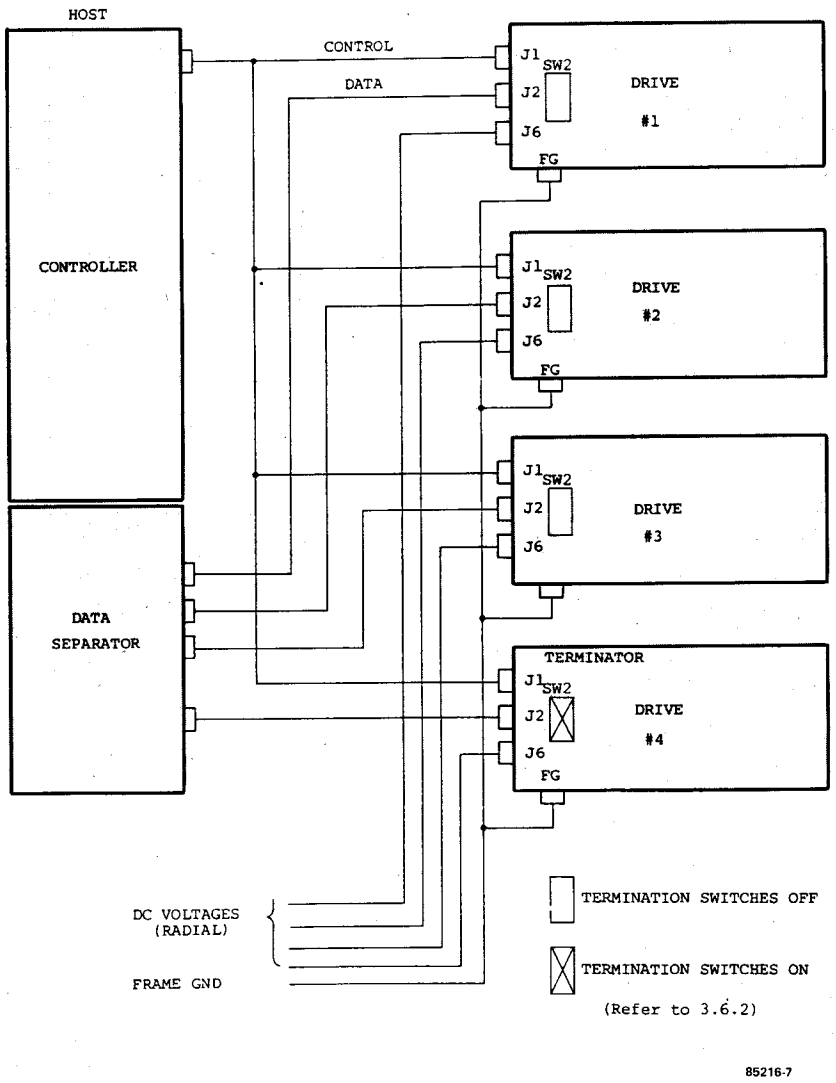
Drive Selected (DRV SELTD):
Deze signaaluitgang wordt aktief wanneer de drive geselekteerd is. Het signaal dient als verifikatie-signaal voor de controller.

Ready (READY):
Wanneer deze signaallijn en de "Seek Complete"-lijn beide "low" zijn, staat de drive klaar voor schrijven, lezen of een "seek operation", dus een beweging van de kam met de koppen. Is de ready-lijn "high", dan zijn deze aktiviteiten geblokkeerd.
Na het inschakelen van de voedingsspanning moet aan drie voorwaarden voldaan zijn, voordat de ready-lijn aktief wordt:
* De track-Ø-herkalibratie moet voltooid zijn,
* het motortoerental moet binnen de toegestane tolerantie (meestal 1%) liggen, en
* de lijn "Write Fault" mag niet aktief zijn (zie hieronder).

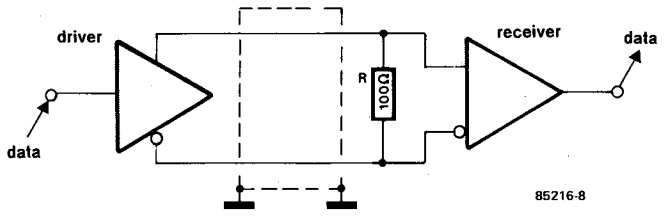
Write Fault (WRT FLT):
Een 'Ø'-signaal op deze lijn signaleert dat de drive een schrijf-operatie niet naar behoren kan uitvoeren. Dit kan de volgende oorzaken hebben:
* Er is meer dan één kop geselekteerd,
* er is geen schrijfstroom aanwezig,
* er is loopt een schrijfstroom tijdens het lezen,
* de voedingsspanning zit buiten de tolerantiegrenzen.
Teneinde dit signaal weer in zijn rusttoestand te laten terugkeren, moeten de signalen DR SEL en/of WRT GATE inaktief ("1") gemaakt worden.

MFM Write Data (MFM WRT DATA):
Als de lijn WRT GATE aktief is, wordt via dit paar datalijnen (!) naar de schijf geschreven. Elke impuls op deze lijnen heeft een magnetische impuls in de kop tot gevolg, die een "afdruk" op het magnetische materiaal van de schijf achterlaat.

7



8

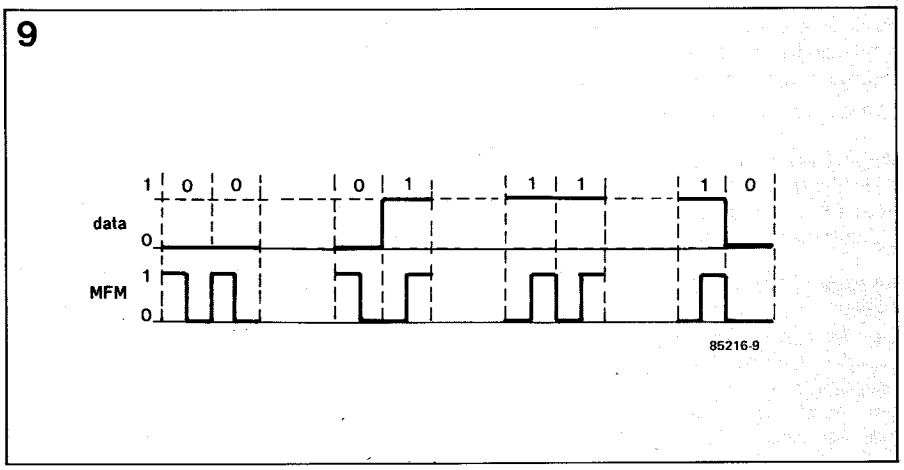


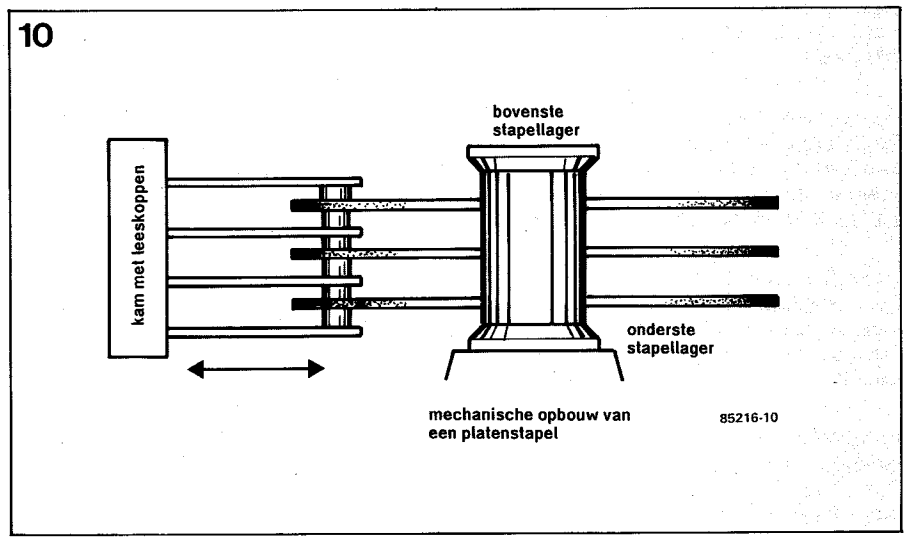
MFM Read Data (MFM RD DATA):
Via dit lijnenpaar worden de van de schijf gelezen data in de vorm van elektrische informatie aan de controller doorgegeven: De magnetische "afdrukken" op de schijf veranderen het elektromagnetische veld in de spoel van de geselekteerde kop. Deze veranderingen leveren een zwak elektrisch signaal, dat na versterking tenslotte als data doorgegeven wordt. Tijdens dit gebeuren moet de WRT GATE-lijn logisch één zijn.
De zeer hoge transmissiesnelheid over de bussen (bijv. 1 Mbit/s) stelt hoge eisen aan de signaallijnen. Daarom worden alle lijnen zeer laagohmig afgesloten met 220

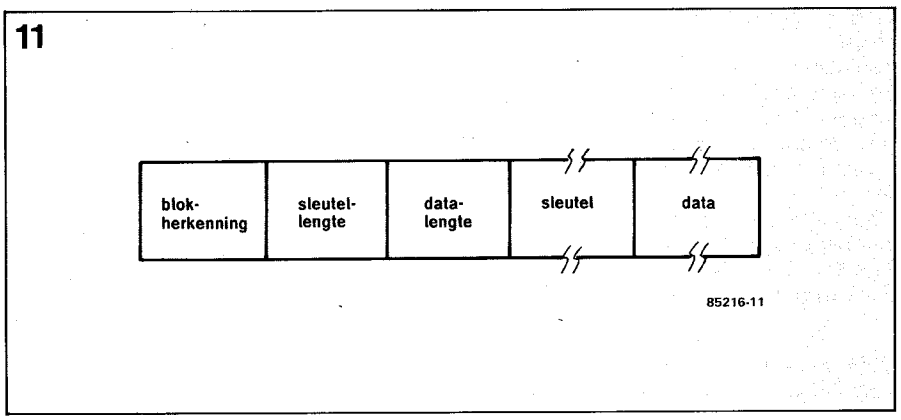
Ohm naar +5 V en 330 Ohm naar massa. De buffers die de lijnen moeten sturen zijn meestal open-kollektor-NAND's van het type 7438. Aan de ontvangstzijde worden over het algemeen schmitt-triggers met temperatuurgekompenseerde hysteresis toegepast van het type 7414. De transmissiesnelheid over de datalijnen ligt rond de 5 Mbit/s. Daarom worden deze signalen ter beperking van de storingsgevoeligheid over symmetrische lijnen verstuurd. Hiervoor worden speciale drivers van het type 3486/87 toegepast en vaak speciale verbindingen, zoals afgeschermde kabels. Dit principe wordt overigens om dezelfde reden ook in de studiotechniek bij mikrofoons toegepast.
De data worden serieel getransporteerd. Een apart kloksignaal is daarbij niet nodig, door toepassing van een speciale modulatietechniek. Bij deze MFM-methode (Modified Frequency Modulation), die we ook al kennen als "double density" bij de floppy-disk, worden klok en data gemeenschappelijk over één lijn getransporteerd. Met behulp van een "data separator" worden klok en data van elkaar gescheiden. Een modulatietechniek als MFM heeft t.o.v. de gewone FM-techniek het nadeel dat de dubbele bandbreedte vereist is. Figuur 9 geeft hiervoor de verklaring. De databits worden door het kloksignaal gemoduleerd, waardoor een frekwentieverdubbeling optreedt. Bij een 0-bit is het uitgangssignaal gelijk aan het kloksignaal en bij een 1-bit is het uitgangssignaal gelijk aan het geïnverteerde kloksignaal. Wat er gebeurt wanneer "0" en "1" elkaar afwisselen, wordt door figuur 9 beter verklaard dan met woorden is uit te leggen.
De data-separator maakt gebruik van een PLL-schakeling. Meestal worden hier analoge circuits toegepast.De digitale PLL (DPLL) is echter in opmars; Er zijn al komplete DPLL-chips beschikbaar. In TTL-technologie is dat bijvoorbeeld de SN 74297. Kwalitatief moet de DPLL het nu nog afleggen tegen zijn analoge broer. Een voordeel van de DPLL is dat er geen afregeling nodig is. De analoge PLL is echter al geruime tijd in een zelf afregelende uitvoering (van Western Digital) beschikbaar.
De geavanceerde harddisk-technologie heeft er samen met zeer veilige datatransmissie-methodes voor gezorgd dat de harddisk momenteel een van de veiligste opslagmedia is. De meeste fabrikanten geven als gemiddelde tijd tussen twee storingen al een MTBF (=Mean Time Between Failures) op tussen de 10000 en 20000 uur, overeenkomend met één tot twee jaar.

Data op de harde schijf

Het schijfoppervlak wordt niet spiraalvormig, zoals bij de grammofoonplaat, met data beschreven, maar in koncentrische cirkels, die sporen of "tracks" genoemd worden. Dezelfde methode dus die bij de floppy-disk wordt gehanteerd. Op deze manier wordt de schijf in 200 tot 400 sporen verdeeld. Bij een schijvenpakket zijn de koppen van elke schijf niet onafhankelijk van elkaar te positioneren. De koppen zijn op een gemeenschappelijke koppendrager gemonteerd, die qua uiterlijk aan een kam doet denken (figuur 10). Dit houdt in dat zonder beweging van de koppen dezelfde spoornummers op alle schijven tegelijkertijd toegankelijk zijn. De boven elkaar liggende cirkelvormige sporen vormen samen een cylinder. Daarom wordt bij de harddisk meestal de term "cylinder" i.p.v. spoor/track gebruikt. De sporen of cylinders worden verdeeld in sektoren, zoals dat ook bij de floppy gebruikelijk is. Spoor- en sektornummer zijn voor de controller belangrijke oriëntatiepunten.
Tot zover de mechanische aspekten van de harddisk. Hoe worden nu de data ondergebracht? Principieel worden alle programma's en databestanden in blokken op de schijf gezet. Een blok is de kleinste data-eenheid op de schijf waarin een bepaalde hoeveelheid informatie kan







Figuur 9. In deze figuur is het verband te herkennen tussen data en klok bij de MFM-metode. Klok en data zijn in één signaal gekombineerd. De door de industrie en vaak ook in de vakliteratuur gebruikte term "MFM-kode"(of MFM-gekodeerd) is onjuist; een modulatiemethode is nu eenmaal geen kode.

Figuur 10. De mechanische opbouw van een schijvenpakket met koppenkam is hier nog eens schematisch weergegeven. De onder elkaar liggende sporen vormen samen een "cylinder", in dit schema bestaat de cylinder dus uit zes sporen.

Figuur 11. Dit schema geeft de typische opbouw van een datablok op de schijf weer. De grote hoeveelheid informatie die op een harddisk kan worden opgeslagen, maakt het noodzakelijk om voor het operating system enige bytes te reserveren, teneinde een bepaald bestand gemakkelijk te kunnen lokaliseren.

worden gezet. Zelfs het kleinste programma heeft dus minimaal één blok aan ruimte op de schijf nodig. Wanneer de blokken voor alle bestandstypen uniform van afmeting zijn, kan de blokverdeling van een schijf bij het installeren d.m.v. een hulpprogramma gebeuren. Deze procedure wordt het formatteren van de schijf genoemd.
Zoals gezegd worden alle bestanden in blokvorm op de schijf gezet. De blokken moeten dus identificeerbaar zijn. Hierbij zijn twee type-aanduidingen gebruikelijk:

* Een type-aanduiding die bestaat uit een samenvoeging van cylinder-, kop- en bloknummer. Bij ieder blok is deze typering op een vaste plaats en met een vaste lengte aanwezig.
* Een type-aanduiding met een variabele lengte, die naar goeddunken aan het blok kan worden toegevoegd. Deze type-aanduiding heeft dan ook een variabele lengte en wordt ook wel sleutel of key genoemd.

In het algemeen is een blok dus opgebouwd volgens het in figuur 11 aangegeven schema. De aanduidingen "lengte sleutel" en "lengte data" zijn nodig om het begin respektievelijk het einde van deze variabele blokdelen vast te kunnen stellen. Een fout in één van de lengte-aanduidingen heeft dus fatale gevolgen.

Twee speciale bestanden

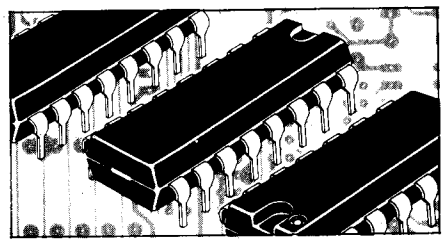
Schijfidentifikatie, schijfetiket
Ter identifikatie is iedere schijf van een aantal gegevens voorzien. Deze gegevens staan altijd in een apart blok, bijvoorbeeld op spoor 0. Dit identifikatie-blok bevat over het algemeen de volgende gegevens :

* Naam van het schijvenpakket
* Naam van de eigenaar
* Datum van ingebruikname
* Een verwijzing naar vervangende tracks
* Een verwijzing naar het directory
* Een verwijzing naar extra informatie

Inhoudsopgave, data-catalogue, directory
Het directory bevat alle bestands- en programmanamen plus alle bijbehorende gegevens. Gewoonlijk is het directory direkt na het idenfikatieblok op track 0, aan het eind (bijv. track 199) of in het midden van de schijf te vinden. In het laatste geval wordt het aantal bewegingen van de koppenkam enigszins beperkt.

Literatuur:
BASF 6188 Fixed Disk Drive Specification, BASF
Mitsubishi MR521/522 Disk Specification, Mitsubishi
Olivetti HD 661 Product Description, Olivetti

Met dank aan de Fa. BASF voor het beschikbaar gestelde illustratiemateriaal.



harddisk op de Big-Board-II

In dit artikel zullen we ons bezighouden met de werking van de SASI-interface op de Big-Board-II, hoe men via deze interface kan kommuniceren met een harddisk-controller en hoe men hiervoor een CP/M-systeem op de schijf installeert.

Voor opslag van data in het Big-Board-II-systeem kennen we de gewone floppy. Het is echter ook mogelijk om dataopslag-in-het-groot te doen met een harddisk-loopwerk. Wat hebben we daarvoor nodig?

- Allereerst een harddisk-loopwerk met een opslagkapaciteit tussen 5 en 50 Mbyte
- Een XEBEC-harddisk-controller
- Een diskette waarop zich het programma VARBIOS.MAC en de diverse LIB-files bevinden. Deze diskette wordt bij de print van de Big-Board-II geleverd. Wie het installeren van CP/M op de harddisk wat al te moeilijk vindt, hoeft niet te wanhopen. De firma "Twente Digital" heeft in samenwerking met de Duitse firma IEV een speciale diskette gemaakt die het installeerproces op de BBII automatisch afwerkt.

In totaal moeten er drie kabels worden aangesloten: een kabel tussen de XEBEC-controller en de SASI-interface van de BBII en twee kabels die de controller met het loopwerk verbinden. Op de schijf moet een CP/M- of ZCPR-2-systeem gegenereerd worden en de harddisk moet geformatteerd worden. Het installeren van CP/M op de harddisk doet de BBII praktisch automatisch.

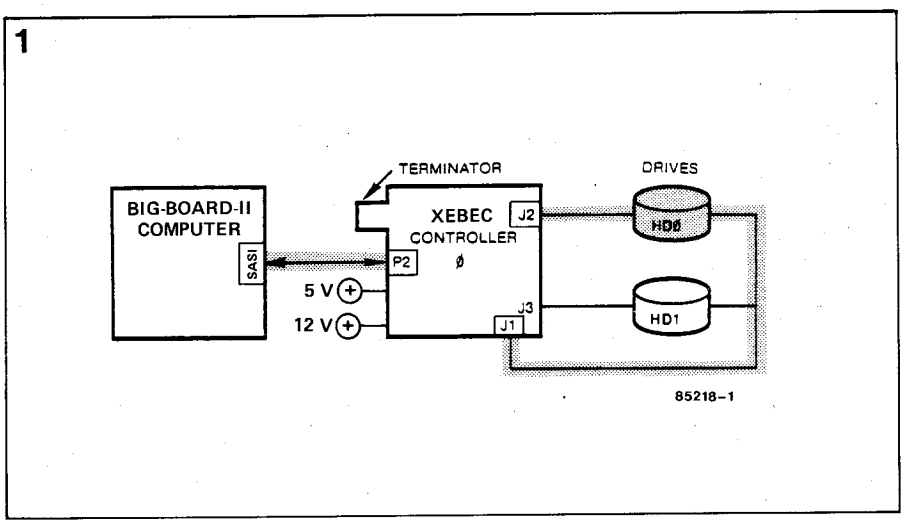
Computer — SASI — harddisk

Figuur 1 toont ons de opzet van het computersysteem met harddisk. De XEBEC-controller is via konnektor P2 verbonden met de SASI-interface op de computer. Een controller kan maximaal twee harddisks bedienen. Hier houden we ons bezig met het aansluiten van één drive "HD0" op de konnektors J1 en J2. De besturingssignalen voor de drive verlopen via J1. J2 bevat de lees/schrijfsignalen op TTL-nivo, die serieel naar de lees/schrijfkop-versterkers worden gevoerd. In verband met de zeer hoge transmssiesnelheid (5 Mbit/s) moeten deze signalen over symmetrische leidingen worden getransporteerd. J2 en J3 worden dus vanuit symmetrische drivers gevoed. In het vorige artikel hebben we de bekabeling van de konnektors J1 en J2 reeds besproken.

De SASI-interface

De SASI-kommunikatie tussen computer en controller verloopt via een 50-polige steker. De pennummering en benaming van de signalen is terug te vinden in tabel

Figuur 1. Het loopwerk is via twee kabels verbonden met de XEBEC-controller, die op zijn beurt met één kabel is aangesloten op de SASI-interace. Eén XEBEC-controller kan maximaal twee loopwerken bedienen.



1. Tabel 2 bevat uitvoeriger gegevens betreffende de status-signalen op de SASI-kabel. Alle SASI-signalen zijn op de even-genummerde pennen te vinden. De oneven pennen zijn met massa verbonden. We zullen ons nu eens bezighouden met de signalen zelf:

I/O: input/output
I/O is een open-kollektor-uitgang van de XEBEC-controller. Is deze lijn "0", dan zendt de controller data naar de SASI-poort over de datalijnen DATA0...DATA7. Is de lijn "1", dan verloopt het data-transport precies anderom: de computer zendt en de controller ontvangt. De computer gebruikt dit signaal om de SASI-busbuffer om te schakelen. Het signaal REQ maakt I/O geldig of ongeldig.

C/D: command/data
Dit is eveneens een open-kollektor-uitgang van de controller. Deze lijn informeert de computer over de aard van de data op de bus. "0" betekent dat het een kommando betreft en "1" wil zeggen dat er normale data (gegevens) over de databus "jagen". Ook hier geeft het REQ-signaal de geldigheid van C/D aan.

BUSY
BUSY is een open-kollektor-uitgang van de controller. De controller zendt dit signaal naar de computer nadat hij een "SELECT"-signaal over de databus heeft ontvangen. De computer wordt zo geïnformeerd of de controller klaar is voor data-uitwisseling.

MSG: message
Ook hier gaat het om een open-kollektor-uitgang van de controller. Is MSG "0", dan heeft de controller een kommando suksesvol uitgevoerd. Het I/O-signaal is altijd "0" als MSG aktief is. Hiermee kan de controller dus de busbuffer van de SASI-poort bedienen. REQ geeft de geldigheid van het signaal aan.

REQ: request
REQ is eveneens een open-kollektor-uitgang van de controller. Door deze lijn aktief te maken geeft de controller te kennen dat data vanuit de computer kan worden verzonden. Tevens bepaalt dit signaal de geldigheid van de signalen I/O, C/D en MSG.

ACK: acknowledge
Als antwoord op het REQ-signaal geeft de computer een ACK: acknowledge.
Als antwoord op het REQ-signaal geeft de computer een ACK-signaal terug naar de controller, ten teken dat de computer beschikbaar is voor het verzenden of het ontvangen van data. Na elke REQ moet een ACK volgen. ACK is dus een ingang van de controller.

RST: reset
Het RST-signaal wordt vanuit de computer gezonden en reset de controller. Alle signalen van en naar de harddisk-drive worden dan uitgeschakeld. De minimale lengte van het RST-signaal is 100 ns.

Tabel 1: de SASI-interface

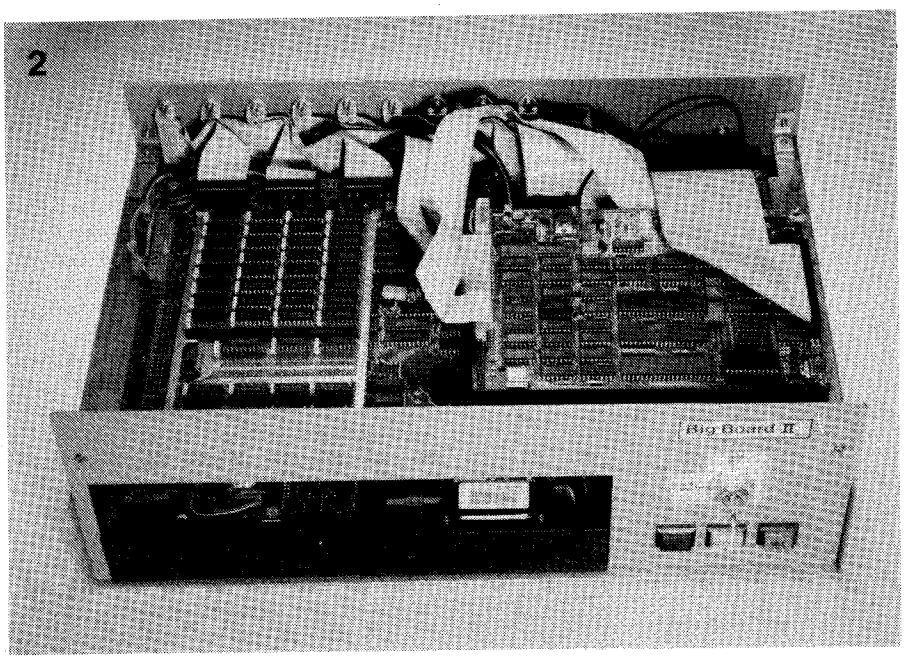
Pin	Mnemonic	Engels	Nederlands
2	DATA0	bit-0	
4	DATA1	bit-1	
6	DATA2	bit-2	
8	DATA3	bit-3	
10	DATA4	bit-4	
12	DATA5	bit-5	
14	DATA6	bit-6	
16	DATA7	bit-7	
18...34	NC		niet gebruikt
36	BUSY	Busy	bezig
38	ACK	Acknowledge	ontvangst-bevestiging
40	RST	Reset	reset
42	MSG	Message	boodschap
44	SEL	Select	selekteren
46	C/D	Command/Data	kommando/data
48	REQ	Request	bus-aanvraag
50	I/O	Input/Output	ingang/uitgang

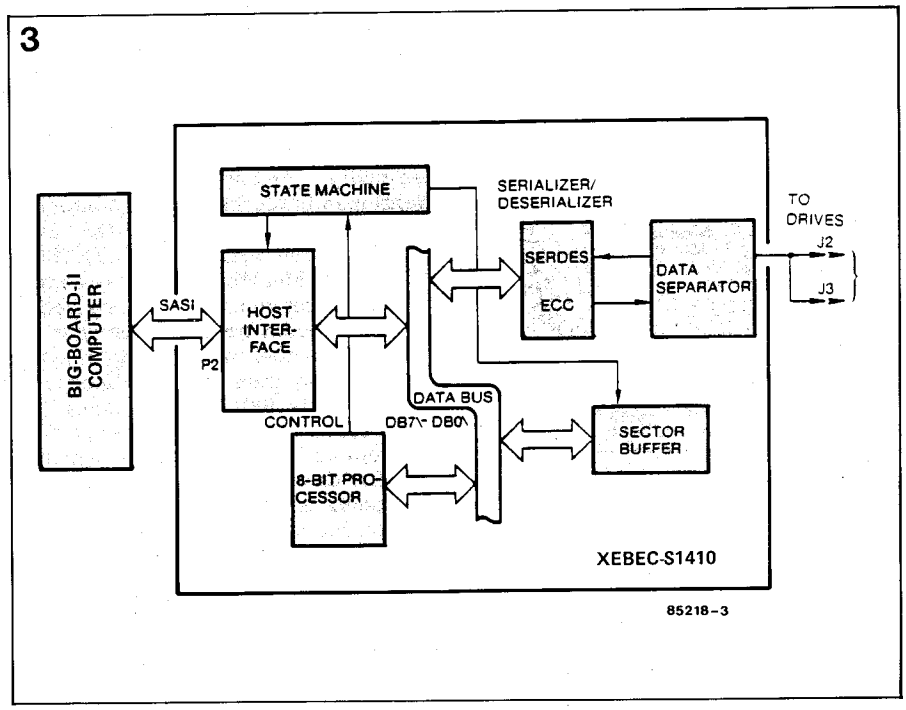
Tabel 2: de status-signalen op de SASI-interface

DRV DRH I/O	C/D	MSG	Definitie
1	0	1	De XEBEC-controller ontvangt een kommando van de SASI-interface
1	1	1	De XEBEC-controller ontvangt data van de SASI-interface
0	1	1	De XEBEC-controller zendt data naar de SASI-interface
0	0	1	De XEBEC-controller zendt een "ERROR"-byte naar de SASI-interface
0	0	0	De XEBEC-controller heeft een kommando korrekt uitgevoerd

SEL: select
Eén van maximaal acht aanwezige XEBEC-controllers kan met dit selekteersignaal gekozen worden. De controller mag niet ergens mee bezig zijn als de computer het SEL-kommando geeft. Na afloop van de momentele kommando's moet het SEL-signaal weer inaktief worden gemaakt.

DB0...DB7: SASI-databus
De data-uitwisseling tussen de computer en de harddisk-controller geschiedt over





Figuur 2. Een luchtfoto van de Big-Board-II. Rechts zien we de controller en links de RAM-disk. De Winchester-drive zit naast de floppy-disk-drive gemonteerd.

Figuur 3. Het blokschema van de XEBEC-controller herbergt een "dedicated" computer, zodat op een relatief kleine ruimte toch veel "intelligentie" past.

deze databus. Ook het selekteren van de controller wordt over deze bus gedaan: bit 0 = controller 0, bit 1 = controller 1, enzovoort. Omdat elke controller op zijn beurt twee drives kan bedienen, bedraagt het maximale aantal aan te sluiten drives 16 (!).
In tabel 2 vinden we verdere gegevens over een aantal signalen op de SASI-bus die de kommunikatie tussen de computer en de controller besturen.

De XEBEC-controller

Figuur 3 toont ons het blokchema van het inwendige van de controller. Het schema is tamelijk oppervlakkig; in werkelijkheid bevat de print de modernste hardware op een relatief kleine oppervlakte. De afmetingen van de print zijn zo gekozen dat deze netjes boven op de drive kan worden monteerd.
De "host-interface" verbindt de SASI-databus met de interne databus van de controller. Alle data-uitwisseling vindt dus plaats via deze interface. De intelligentie van het systeem zit verstopt in het blok "8-bit-processor". Deze maakt het mogelijk om met slechts enkele kommando's data over te zenden van een CP/M-machine naar een harddisk. Tevens bewaakt de interne processor alle kontrolesignalen op de diverse bussystemen van de controller.
Het blok "SERDES" (serializer/deserializer) zet in de ene richting parallelle signalen om in seriële en in de andere richting omgekeerd. Het gehele dataverkeer tussen de drive en de host-interface verloopt via SERDES. De "data seperator" scheidt het inkomende signaal in een data-signaal en de klok, voordat het signaal naar de SERDES toe gaat. Het blok "state machine" synchroniseert het datatransport tussen de host-interface, de SERDES en de "sector buffer". Deze laatste is nodig voor een snelle buffering van data. De transmissiesnelheid bedraagt immers maar liefst 5 Mbit/s! Zonder de sector buffer zou geen enkele computer deze snelheid bijhouden.

Kommando's en status

De data-uitwisseling tussen de computer en de harddisk-controller vindt dus plaats over de SASI-bus. We kunnen drie verschillende soorten data onderscheiden die via deze bus worden getransporteerd:

- kommando's voor de XEBEC-controller
- status-informatie van de XEBEC-controller
- data van en naar de harddisk

Voordat de data van de harddisk naar het geheugen in de computer kunnen worden gekopiëerd, moet de controller eerst een kommando ontvangen. De controller antwoordt hierop door direkt na elkaar twee status-bytes terug te zenden. Door deze bytes te testen, kan de computer te weten komen of de controller het kommando heeft geaksepteerd.
Het kommando voor de controller is samengesteld uit zes opeenvolgende bytes, samen een "device control block" oftewel "DCB". In figuur 4a zien we de samenstelling van een DCB.
Byte 0 is opgedeeld in twee delen: bit 7, 6 en 5 bevatten de kategorie of de klasse van de kommando's. De opcode vinden we in bit 4...bit 0.
Byte 1 is eveneens opgedeeld in twee groepen. Bit 7, 6 en 5 bevatten het nummer van de loopwerken (LUN = Logical Unit Number) die zijn aangesloten op J2 en J3 (zie ook figuur 1). Daar er maar twee loopwerken op een controller kunnen worden aangesloten, kan LUN 000 zijn ofwel 001. De rest van byte 1 bevat een deel van het drive-adres dat wordt voortgezet in byte 2 en byte 3. Dit adres wordt het logische drive-adres genoemd (LOGAD). Het is 21 bits lang en wordt op de volgende manier bepaald:

LOGAD = (CYADR * HDCYL + HDADR) * SETRK + SEADR

- LOGAD = logical drive-address
- CYADR = cylinder-address
- HDADR = head-address
- SEADR = sector-address

■ HDCYL = heads per cylinder
■ SETRK = sectors per track

Byte 4 bevat de "interleave count", een soort "versleutel"faktor voor de sektoren. Wat hiermee wordt bedoeld, kunnen we begrijpelijk maken aan de hand van het in figuur 4c geschetste voorbeeld. Net als bij de normale, verwisselbare diskettes wordt ook hier de schijf opgedeeld in sektoren. Figuur 4c stelt één spoor voor dat is opgedeeld in 32 sektoren. Aan de binnenzijde van de cirkel zien we de nummering van de sektoren zoals we die tegenkomen met de draairichting van de schijf mee (physical sector). Aan de buitenkant van de schijf staat de toekenning van het logische nummer (logical sector). In dit voorbeeld is de versleutelingsfaktor 5. Dit betekent dat de logische sektornummering steeds vijf "fysieke" plaatsen overbrugt. Tel maar eens vanaf sektor 0 (buitenring): na 5 posities verschijnt steeds de volgende logische sektor. Deze methode van sektor-toekenning heeft het voordeel dat de harddisk-controller tijd wint om de data te verwerken die van of naar de schijf toe gaan. De maximaal toegestane faktor is:
aantal sektoren per spoor - 1(je loopt dan precies de andere kant op). Een juiste keuze van de interleave count factor is zeer belangrijk voor de verwerkingssnelheid van het gehele harddisk-systeem.
Als laatste hebben we bit 5, het controlbyte. Hiermee kunnen we de controller voor verschillende soorten harddiskloopwerken optimaal aanpassen. Er zijn immers verschillen tussen de diverse fabrikaten. We zullen byte 5 eens bit voor bit doorlopen:

Bit 0: wanneer dit geset is, dan betekent dat: halve-stap-optie voor Seagate- en Texas-Instruments-drives.

Bit 1: indien geset: halve-stap-optie voor Tandon-drives.

Bit 2: buffer-step-optie voor drives van Computer Memories Inc. en Rotating Memory Inc.. Deze drives kunnen elke 200 μs één stepper-puls verwerken.

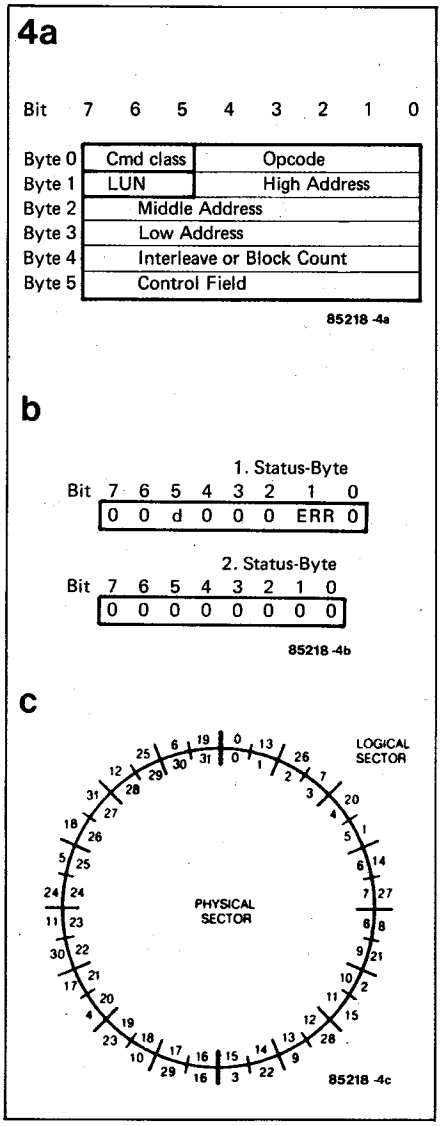
Bit 3...bit 5: deze bits worden niet gebruikt en zijn altijd nul.

Bit 6: dit bit dient normaliter "0" te zijn. Wanneer dit echter tijdens een sektor-read-kommando wordt geset, dan wordt de betreffende sektor bij een leesoperatie met fouten niet nog eens gelezen.

Bit 7: dit bit behoort in de normale situatie "0" te zijn. Bij een lees- of schrijffout op de schijf wordt de operatie vier keer herhaald. Door dit bit te setten dwingt men de controller om alle herhalingen niet uit te voeren.

De status-bytes

De XEBEC-controller zendt telkens twee status-bytes terug na een ontvangen kommando. In het eerste status-byte is bit 1 geset als er tijdens het uitvoeren van het kommando een fout is opgetreden. Bit 5 van dit byte geeft aan bij welke van de twee aangesloten harddisk-drives het kommando niet kon worden uitgevoerd. de resterende bits van het eerste statusbyte zijn "0". De bits in het tweede statusbyte zijn altijd gereset.



Figuur 4a. Een kommando dat vanuit de computer naar de controller wordt gezonden, bestaat uit een blok van 6 bytes. Deze bytes worden niet alijd volledig gebruikt.

Figuur 4b. Na een kommando antwoordt de controller met twee status-bytes. Er zijn maar twee bits in het eerste byte die informatie geven over een fout.

Figuur 4c. Net als bij de normale diskettes worden de sektoren op een spoor "versleuteld". Het voordeel hiervan is, dat de processor enkele mikrosekonden tijd heeft om de data te verwerken voordat een nieuwe sektor wordt aangesproken.

Figuur 5. Een listing van de file "SETUP.LIB". Deze file wordt gebruikt door zowel de harddisk-formatteerder als de VARBIOS. Door middel van het setten van flags (TRUE/FALSE) kan de Big-Board-II op uw speciale behoeften worden afgestemd. Voor het berekenen van de Winchester-parameters bewijzen een rekenapparaat en het handboek van de betreffende drive nuttige diensten.

De software

Na de hardware-beschrijving geven we nog een voorbeeld hoe men een willekeurige Winchester-drive op de Big-Board-II kan aansluiten. Bij onze computer hebben we gebruik gemaakt van een BASF-harddisk-drive in kombinatie met een floppy-disk-drive (DS/DD). Omdat CP/M maar 8 Mbyte per drive kan aansturen, is de beschikbare ruimte verdeeld over twee drives van elk 6 Mbyte (via software). Daarnaast is er ook nog een RAM-disk aanwezig van 256 KB. Voor het assembleren van VARBIOS gebruikten we de assembler/linker M80/L80 van Microsoft. Bij de start heeft de floppy het nummer "A" en is de Winchester aangesloten op J2 van de XEBEC-controller. CP/M zal dit loopwerk dan adresseren als zijnde "B" en "C". Hier zullen we dadelijk nog op terug komen.
Alle parameters voor het computersysteem zijn gedefiniëerd in de file SETUP.LIB. In figuur 5 zien we een gedeelte van deze file. Door middel van het setten van de flags "TRUE = waar" en "FALSE = niet waar" kan de VARBIOS eenvoudig gekonfigureerd worden. Wenst men bijvoorbeeld een Centronics-uitgang, dan moet "CENTRON EQU FALSE" worden veranderd in "CENTRON EQU TRUE". Met Wordstar of elke andere tekstverwerker kunnen dergelijke modifikaties eenvoudig worden ingebracht.
In het laatste deel van de listing in figuur 5 wordt de software voor de Winchesterdrive voor de BBII gekonfigureerd. De BASF-6188 heeft 360 cylinders (maxcyl) en vier koppen (maxhds). Experimenten met de BASF-drive hebben aangetoond dat alle sporen met dezelfde kopstroom kunnen worden beschreven. Om deze reden maken we gebruik van de optie: "gereduceerde kopstroom" voor alle sporen. Door middel van "reduce equ 360" en "precomp equ 360" wordt deze optie ingevoerd. Ook is de optie "fast step" bij de BASF mogelijk. Deze optie wordt ook gebruikt bij Tandon-drives. Uit de gegevens van het XEBEC-handboek volgt voor deze optie de overeenkomstige waarde: "stepopt equ 00000100b".
CP/M verwerkt de data op een floppy-disk of harddisk in blokken van elk 4 Kbyte. Daarom moet er bij de opdeling in blokken/sektoren wat gerekend worden. De sektor-opdeling op een harddisk is 256-, 512 of 1024 bytes per sektor. Op één spoor passen dus respektievelijk 32, 17 of 9 sektoren. Wanneer men dit terugrekent naar CP/M-blokken, dan krijgen we:

(256 * 32) / 4096 = 2.0
(512 * 17) / 4096 = 2.125
(1024 * 9) / 4096 = 2.25

Hier kiezen we voor "blocks per track": blptrk = 2.125. De resterende parameters voor "wblmax, maxblk, MAX...,REST..." zijn eenvoudig met behulp van de in de listing gegeven formules en met een rekenapparaat te berekenen.

5

```
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
;       ...DEFINE THE SYSTEM VARIABLES...
;
BNKBIOS EQU     true            ;WE PUT SOME BIOS STUFF TO
                                ;ANOTHER BANK. (Current = rom bank,
                                ; at location 3000h.)
ZCPR    EQU     true            ;ZCPR REPLACES CCP MODULE
CPMROM  EQU     FALSE           ;WE LOAD CCP(ZCPR2)&BDOS FROM ROM
CPMBNK  EQU     false           ;WE RELOAD CCP(ZCPR2)&BDOS FROM THE
                                ; ROM BANK (location 1000H-25FFH)
RAMDISK EQU     true            ;IF WE HAVE A RAMDISK
BBCOPT  EQU     TRUE            ;DO INCLUDE/EXCLUDE BBC CODING IN MYDEBLOK
                                ; CONDITIONAL FIR SIZE REASONS.
WARMB   EQU     true            ;WARM BOOT FROM WINCHESTER
SEC256  EQU     FALSE           ;XEBEC SECTOR SIZE=256
SEC512  EQU     true            ;XEBEC SECTOR SIZE=512
ADAPT   EQU     FALSE           ;ADAPTEC CONTROLLER WITH 1K SECTORS
MODXBC  EQU     FALSE           ;XEBEC CONTROLLER EPROM NOT MODIFIED.
NEWIO   EQU     true            ;MAKE USE OF I/O BYTE ???


        IF      BNKBIOS
OFFSET  EQU     1152
        ELSE
OFFSET  EQU     384
        ENDIF


CENTRON EQU     false           ;USED IF WE HAVE A CENTRONICS PRINTER
IOBYTE  EQU     0003H           ;INTEL CONFIGURATION BYTE
USRDRV  EQU     00H             ;USER/DRIVE TO LOG ON AFTER COLDSTART
;
DOUBLE  EQU     true            ;SINGLE/DOUBLE SIDED DRIVE ENABLE
SERIAL  EQU     FALSE           ;DISABLE SIO/CRT CONSOLE COMBINATION
LSTDVA  EQU     false           ;LIST DEVICE PORT (FALSE IS B)

        IF      RAMDISK
BANK2   EQU     8000H
        ENDIF

        IF      CPMROM
ROM2LOC EQU     2000H           ;ROM LOCATION
        ENDIF

CPMLEN  EQU     1600H           ;CPM LENGTH
SYS1    EQU     0C8H            ;ROM SWITCH PORT

        IF      CPMBNK
RAM1LOC EQU     1000H           ;START OF RAM AREA IN BANK-0
        ENDIF

        IF      BNKBIOS
BIOSBNK EQU     3000H           ;BANKED BIOS IN HERE
        ENDIF


  SYSTEM has been tested for the next system definitions.

 MSIZE   EQU     58

 BASE    EQU     (MSIZE-20)*1024
 CCP     EQU     3400H+BASE      ;CONSOLE COMMAND PROCESSOR
 BDOS    EQU     3C06H+BASE      ;OPERATING SYSTEM ENTRY POINT
 DBBUG   EQU     46D2H+BASE      ;DEBLOCK BUG LOCATION IN BDOS
 CBIOS   EQU     4A00H+BASE      ;BASE OF CUSTOM BIOS
 CPMSIZE EQU     (CBIOS-CCP)/128 ;NUMBER OF RECORDS NEEDED FOR CCP+BDOS
 ;
 ;
 MONITR  EQU     0F000H          ;BASE OF SYSTEM MONITOR
 SYSRAM  EQU     0FF00H          ;BASE OF SYSTEM VARIABLES
 SCRATCH EQU     0F700H          ;RAM AREA FOR BIOS/CPM BUFFERS
 XLPOOL  EQU     0FE20H          ;RAM AREA FOR XLATE TABLES (CONFIG)
 ;                               ;USED FOR OVERLAY CHECK ONLY.
 ;
 ;
;       ... DEFINE HOW MANY DRIVES IN SYSTEM ...
;
WINCH   EQU     2               ; NUMBER OF WINCHESTERS
NFLOPPY EQU     1               ;NUMBER OF FLOPPYS (1..4) IN SYSTEM


        IF      RAMDISK
        IF      WARMB
SWAPSEL EQU     FALSE           ;IF WE SWAP DRIVE'S ON SELECT
        ELSE                    ; BUT NOT IF WARM BOOT FROM WINCH.
SWAPSEL EQU     TRUE
        ENDIF                   ;WINCH
        ELSE
SWAPSEL EQU     FALSE           ;USUALY NOT IF NO RAMDISK
        ENDIF                   ;RAMDISK


        IF      RAMDISK
RAMDSK  EQU     NFLOPPY+WINCH   ;ONE HIGHER THAN # OF DISKS
        ENDIF
;

;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        IF      WINCH
        ;definitions for the winchester drive

maxcyl  equ     360             ;maximum number of cylinders
maxhds  equ     4               ;maximum number of heads
reduce  equ     360             ;start reduced write current cylinder #
precomp equ     360             ;start write precompensation cylinder #
maxecc  equ     11              ;maximum ECC data burst error length
stepopt equ     00000100b       ;TANDON FAST STEP OPTION.....!!
bloksiz equ     4096            ;use 4 Kbyte clusters
        if      sec256
secshf  equ     1               ;2 cpm sectors/ record
cpmspt  equ     64              ;32 sectors of 256 bytes
        endif
        if      sec512
secshf  equ     2               ;(base2) log (secsiz/128)
cpmspt  equ     68              ;17 SECTS OF 512 winspt*(secsiz/128)
        endif   ;sec512
        if      adapt
secshf  equ     3
cpmspt  equ     72
        endif
restrk  equ     1               ;one reserved track(s) (cyl 0)
wblshf  defl    0
x       defl    (bloksiz shr 7)
wblmsk  equ     x - 1
        rept    8
x       defl    x shr 1

        ift     x eq 0
        exitm
        endif
wblshf  defl    wblshf + 1
        endm
;blptr  equ     2.0             for 256 setor size
;blptr  equ     2.125           for 512 sector size (----XXXXXXXXXX
;blptr  equ     2.25            for 1024 sector size

wblmax  equ     2047            ;maximum cp/m blocksize for one drive
maxblk  equ     3856            ;((maxcyl * maxhds)-restrk) * blptrk
divblk  equ     maxblk/winch    ;max. blocks per winchester
;
MAX&0   EQU     1525            ;divblk-(restrk*blptr) - 1
REST&0  EQU     RESTRK
;
MAX&1   EQU     1525            ;same as MAX&0
REST&1  EQU     720             ;divblk/blptr + restrk
;
;MAX&2  EQU     1916            ;same as MAX&0
;REST&2 EQU     1708            ;(divblk/blptr)*2 + restrk
;
;       ... END OF WINCHESTER PARAMETER TABLE...

        ENDIF   ;WINCH
```

Het formatteren en het genereren van het systeem

Omdat er nog geen CP/M op de harddisk staat, kennen we de floppy-drive het nummer "A" toe en de Winchester de nummers "B" en "C". In de file SETUP.LIB moet dan op dit moment "WARMB EQU FALSE" zijn, omdat alleen op de diskette een systeem gegenereerd kan worden en niet op de harddisk!
Alvorens er data kunnen worden opgeslagen op de Winchester, zal eerst elke cylinder geïnitialiseerd moeten worden. Op de systeemdiskette voor de BBII staan hulp- en assembler-source-programma's, waarmee het formatteren van de harddisk en het genereren van het CP/M-systeem geen onoverkomelijk probleem hoeft te zijn. Allereerst wordt de Winchester-formatteerder geassembleerd:

M80 =WINFORMT (produceert de file: WINFORMT.REL)
L80 WINFORMT,WINFORMT/N/E (produceert de file: WINFORMT.COM)
De Winchester-formatteerder maakt ook gebruik van de file SETUP.LIB, waarin de technische gegevens van de harddisk-drive gedefinieerd zijn. Dan kan het programma WINFORMT.COM gestart worden. Na ca. 10 minuten zijn alle cylinders geformatteerd.

ZEX — WSYSGEN — VARBIOS

Is de schijf eenmaal geformatteerd, dan moet men de file WSYSGEN.MAC assembleren (WSYSGEN = Winchester-systemgeneration). Ook hier wordt weer gebruik gemaakt van de file SETUP.LIB. In SETUP.LIB is de situatie "WARMB EQU FALSE" nog altijd ongewijzigd. We gaan nu eerst het Winchester-systeem op floppy genereren en het daarna naar de harddisk kopiëren:

WSYSGEN CPM58.COM (voor een normaal CP/M-systeem)

Nu herhalen we de procedure met "WARMB EQU TRUE" in de file SETUP.LIB. De Winchester-drive is nu de eerste twee drives ("A" en "B"), met daarna de floppy-drive ("C") en de RAM-disk ("D", mits aanwezig). Meer bijzonderheden kan men vinden in de assembler-programma's op de diskette die bij de Big-Board-II-print wordt meegeleverd. Supereenvoudig is het installeren met behulp van het programma ZCPR-2, waarvan de gehele assembler-source op de diskette aanwezig is. Men typt dan:

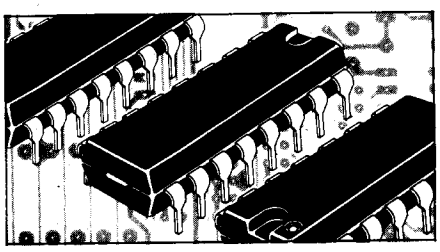
ZEX ZCPRG58 VARBIOS

en dan wordt volautomatisch het CP/M-systeem op de harddisk geschreven. Wanneer er wordt gewerkt met een RAM-floppy, dan moet in plaats van ZCPRG58 de filenaam ZCPRG58B worden ingegeven. De "G" voor "58" betekent systeemgeneratie en de "B" erachter staat voor "banked BIOS". Beide files zijn "ZEX"-files. Wanneer u de dokumentatie van ZCPR-2 doorneemt, dan zal snel duidelijk worden hoe het hulpprogramma ZEX werkt. Het is een vervanger voor de "good old" SUBMIT van CP/M. Figuur 6 toont ons de inhoud van ZCRPG58B.ZEX en geeft een indruk hoe het operating system op de harddisk wordt geïnstalleerd. Een uitvoerige dokumentatie van ZEX is op de diskette aanwezig. Typ maar eens "ZEX //" en u krijgt vele beeldschermpagina's informatie!

6

```
M80 =$1
;       Please type $^C if error(s) exists - */
era $1.bak
era $1.com
L80
/P:DC80
$1,$1/X/N/E
N
DDT CPM58.COM
M980,1F7F,4000
F100,3FFF,0
I$1.HEX
R3380
M4000,55FF,2000
IZCPR58.HEX
R5400
G0
VARSYSG
!
^*
```

Figuur 6. De file "ZCPRG58B.ZEX" bevat alle benodigde informatie om het operating system automatisch op de harddisk te zetten. Duidelijk is te zien hoe de assembler M80 en de linker L80 worden aangeroepen.



De EC-68K — deel 1

De hardware van de droomcomputer

De EC-68K is een moderne computer voor zelfbouw. Twee microprocessoren — een 68008 en een 6809 — garanderen hoge executiesnelheden. De 68008 behoort tot de 68000-familie en kent dezelfde machinekode als zijn grote broer. Bij de EC-68K is voor de 68008 gekozen omdat de bouwkosten vanwege de 8-bits databus drastisch verminderen en desondanks niet afgezien hoeft te worden van de 32-bits architektuur van de 68000. Statische RAM's in het geheugen maken deze computer bijzonder nabouwzeker en hebben ten opzichte van dynamische RAM's het voordeel dat in het geval van een hoge processorklokfrekwentie niet gestopt hoeft te worden tijdens iedere "refresh"-cyclus. Een test in het laboratorium heeft uitgewezen dat een 10-MHz-68008 met statische RAM ongeveer net zo snel werkt als een 8-MHz-68000 met dynamische RAM.

Het 1 Megabyte grote, ongesegmenteerde en lineaire adresgebied van de 68008 maakt het schrijven mogelijk van grote programma-modulen. Het programmeermodel is gelijk aan dat voor de 68000. De programmeur heeft in totaal 17 32-bits registers ter beschikking (figuur 1). De acht dataregisters (D0...D7) kunnen met 1-, 8-, 16- of 32-bits data werken. De zeven adresregisters A0...A6 kunnen als basisadresregisters worden ingezet, of als software-stackpointers. De adresregisters A7 en A7' zijn de systeem-stackpointers. Alle 17 registers van de 68000 kunnen als indexregister worden gebruikt. De snel te leren mnemonics, de vele adresseringsmogelijkheden, het geraffineerde koncept van "exception processing" (uitzonderingstoestand) en de kompatibiliteit met de 68000 maken de 68008 tot de ideale hoofdprocessor voor de EC-68K. Wie de hardware rond de 68000 nog niet kent, raden we aan om het artikel "De 68000 kort en bondig — 1" (in Elektuur Computing 2) te lezen.

Tweede processor (co-processor) in deze zelfbouwcomputer is de 6809. Om de 68008-hoofdprocessor werk uit handen te nemen, is een goedkope achtbitter met een interne 16-bits architektuur toegepast voor de besturing van de floppy-interface. Figuur 2 laat het programmeermodel van deze processor zien. Twee 16 bits brede indexregisters, twee stackpointers en twee 8-bits accumulatoren, die ook als één gekombineerde 16-bits accu kunnen worden gebruikt, zorgen ervoor dat de 6809 de ideale "tweede man" is voor de 68008.

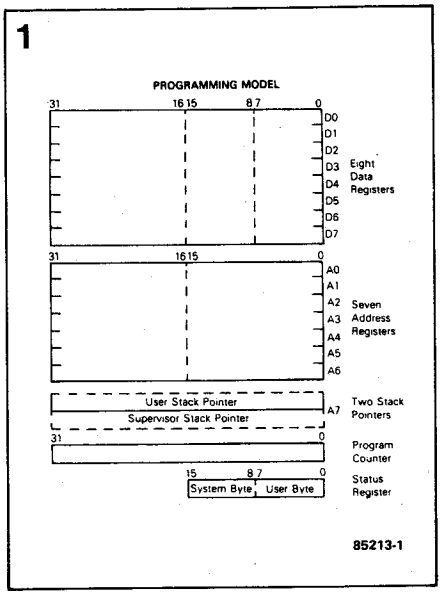
Er zijn in deze zelfbouwcomputer alleen normaal in de handel verkrijgbare TTL-IC's en MOS-komponenten gebruikt. Daarom kan de beschrijving van de schema's kort blijven. Op de busprint passen 13 64-polige konnektoren. De afmetingen zijn zo gekozen dat de computer probleemloos in een 19"-rack kan worden gemonteerd. Indien de EC-68K in de basisversie met floppy-interface is opgebouwd, zijn slechts vijf konnektorposities bezet. Figuur 3 toont de busaansluitingen. Op een bus met 13 konnektorposities kun je heel wat printkaarten kwijt. De eurokaarten die in dit artikel aan de orde komen, vormen tezamen slechts het basissysteem van de EC-68K. Als u dit leest heeft de producent van dit computersysteem, de Duitse firma "Dr. Böhm", inmiddels nieuwe eurokaarten klaar:

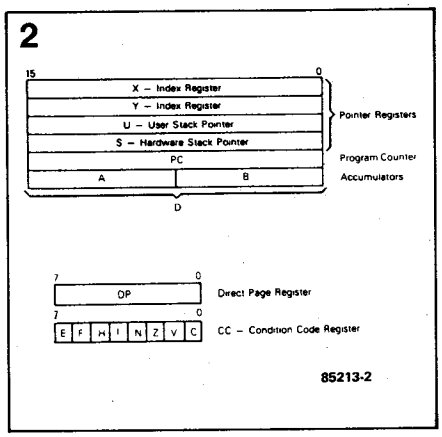
1) EPROM-simulator
2) Parallelle/seriële printer-interface
3) MIDI-interface

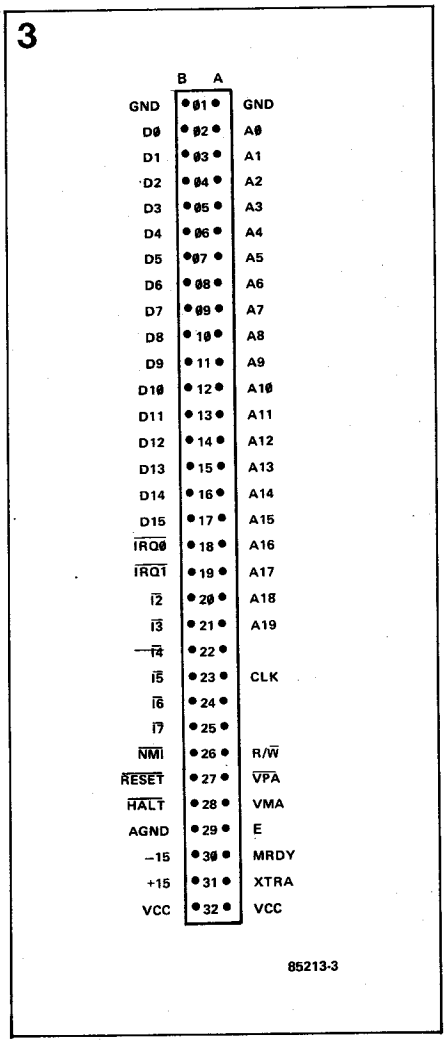
Verdere eurokaarten zijn in voorbereiding. Daarover kunt u zeker meer lezen in toekomstige uitgaven van Elektuur Computing!

De CPU-kaart

Figuur 4a toont het schema van de CPU-kaart. De 68008-processor kan men niet over het hoofd zien. Met 20 adreslijnen is



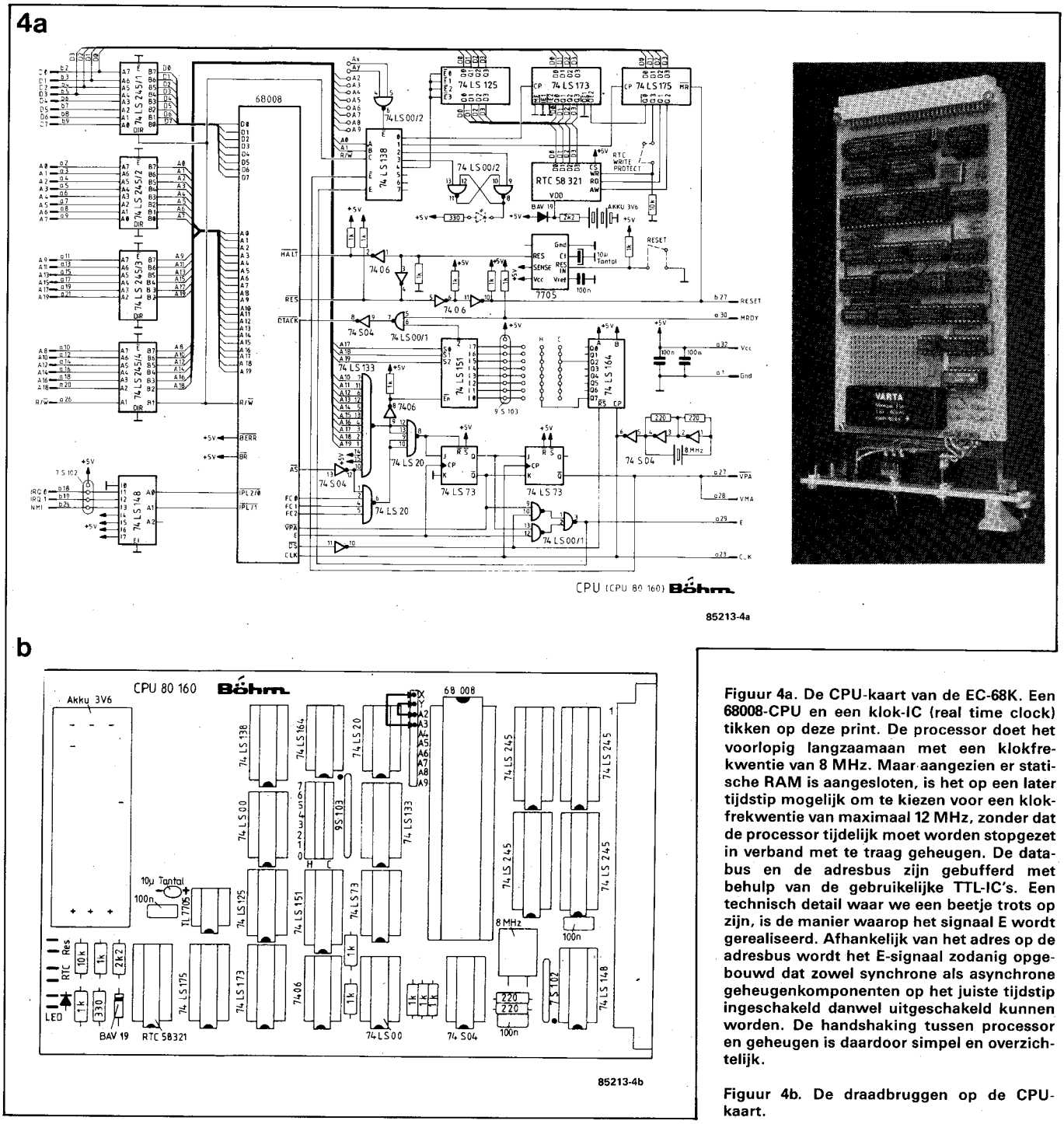




Figuur 1. De registerstruktuur van de 68008 is gelijk aan die van de 68000. Beide processoren gaan uit van dezelfde machinekode. Toch is er een verschil: de 68008 bezit een 8 bits brede databus, terwijl de 68000 een 16 bits brede databus bezit. Aangezien de EC-68K gebruik maakt van statische RAM's in het geheugen, komen wachtcyklussen, bekend van dynamische RAM's tijdens het refreshen, niet voor. Dit heeft tot gevolg dat de 68008 met een klokfrekwentie van 10 MHz ongeveer net zo snel werkt als een 68000 met 8 MHz klokfrekwentie en dynamische RAM's in het geheugen.

Figuur 2. In het floppy-subsysteem wordt een 6809-microprocessor gebruikt. Uit het programmeermodel blijkt dat deze processor intern een 16-bits registerstruktuur bezit.

Figuur 3. De aansluitgegevens van de EC-68K-computerbus. Op de busprint is plaats voor totaal 13 konnektoren ten behoeve van andere systeemkaarten.



Figuur 4a. De CPU-kaart van de EC-68K. Een 68008-CPU en een klok-IC (real time clock) tikken op deze print. De processor doet het voorlopig langzaamaan met een klokfrekwentie van 8 MHz. Maar aangezien er statische RAM is aangesloten, is het op een later tijdstip mogelijk om te kiezen voor een klokfrekwentie van maximaal 12 MHz, zonder dat de processor tijdelijk moet worden stopgezet in verband met te traag geheugen. De databus en de adresbus zijn gebufferd met behulp van de gebruikelijke TTL-IC's. Een technisch detail waar we een beetje trots op zijn, is de manier waarop het signaal E wordt gerealiseerd. Afhankelijk van het adres op de adresbus wordt het E-signaal zodanig opgebouwd dat zowel synchrone als asynchrone geheugenkomponenten op het juiste tijdstip ingeschakeld danwel uitgeschakeld kunnen worden. De handshaking tussen processor en geheugen is daardoor simpel en overzichtelijk.

Figuur 4b. De draadbruggen op de CPU-kaart.

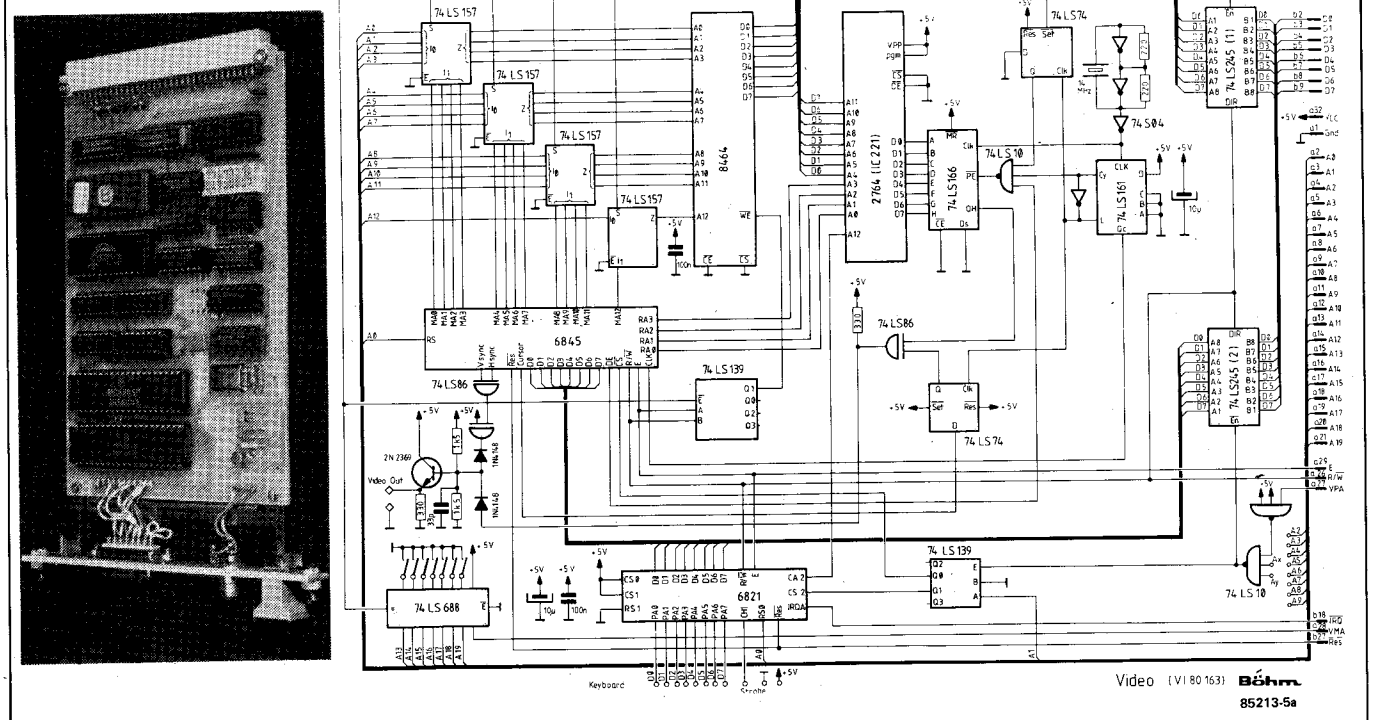
deze in staat om 1 MB geheugen te adresseren. In tegenstelling tot de 68000 is de adreslijn A0 bij de 68008 wèl extern beschikbaar, omdat de databus immers 8 bits breed is. Bij zijn grote broer 68000 is A0 uitsluitend intern beschikbaar voor de realisatie van de datastrobes $\overline{UDS}$ en $\overline{LDS}$. Alle adreslijnen en datalijnen zijn gebufferd met behulp van vier 74LS245-IC's. Drie interrupt-nivo's (IRQ0,IRQ1 en NMI) zijn via de interrupt-encoder 74LS148 op de computerbus van de EC-68K aangesloten. Rechts van de 68008 in figuur 4a zien we nog meer hardware. Een 7705-precisietimer realiseert het $\overline{RESET}$-signaal zodra de computer wordt ingeschakeld. De $\overline{HALT}$- en $\overline{RESET}$-lijnen worden dan gelijktijdig logisch nul gemaakt. De $\overline{RESET}$-lijn van de processor is bidirektioneel. De processor kan via de RESET-instruktie alle erop aangesloten I/O-komponenten initialiseren (software-reset).

De TTL-IC's 74LS238, 74LS125 en 74LS175 besturen het klok-IC (RTC). In het tweede deel, elders in deze Elektuur Computing, treft u een voorbeeldprogramma aan, waarmee u de klok kunt gelijkzetten en de datum kunt instellen.

Met drie inverters (74LS04) is de 8MHz-klokgenerator opgebouwd. Alle signalen op de computerbus zijn van het hieruit afkomstige kloksignaal afgeleid. Teneinde ook traag geheugen (bijvoorbeeld de synchrone 6800-komponenten PIA, VIA, ACIA, enzovoorts) op de processor te kunnen aansluiten, is er voorzien in het schuifregister 74LS164. Dit vertraagt gedurende enige klokperioden het $\overline{DTACK}$-signaal van de 68000. De datastrobe $\overline{DS}$ reset het schuifregister aan het begin van een dergelijke buscyklus. Om de 125 ns wordt een "1" aan de ingang van het schuifregister ingelezen. Na maximaal acht klokperioden is de "1" volledig doorgeschoven. De kortste vertraging wordt verkregen indien alle ingangen van de multiplexer 74LS151 open zijn. De langste vertraging treedt op indien Q7 van de 74LS164 met één van de ingangen van de 74LS151 is verbonden. Aangezien de multiplexer 74LS151 acht ingangen bezit, staan acht 128k-adresbereiken ter beschikking, waarbinnen naar keuze "langzame logika" kan worden gelokaliseerd.

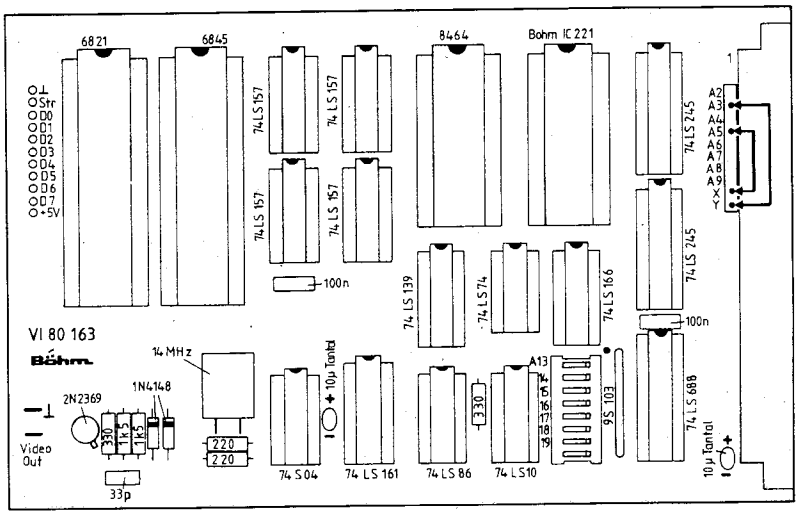
Een NAND-poort 74LS133 dekodeert een adresblok van 1k aan het geheugeneinde ($FFFC0...$FFFFF). Dit geheugenbereik is gereserveerd voor de synchrone I/O-komponenten 65XX en 68XX. Twee NAND-poorten 74LS20 en twee flipflops 74LS73 zorgen ervoor dat de asynchrone

5a



Figuur 5a. Het beeldscherm en het toetsenbord worden op de video/keyboard-print aangesloten. Zoals het schema laat zien is de schakeling konventioneel maar nabouwzeker. Een 8K x 8-karaktergenerator maakt het mogelijk om minstens twee verschillende karaktersets ter beschikking te hebben. De omschakeling van de ene naar de andere set verloopt onder software-besturing. Er kunnen meerdere van deze kaarten op de busprint worden aangesloten. Het is dus mogelijk dat meerdere gebruikers van dit computersysteem gebruik maken.

b



Figuur 5b. De draadbruggen op de video/keyboard-print.

68008 de synchrone komponenten kan besturen. Ook de interrupt-acknowledgecyklus (FC2...FC0=111) wordt bekeken bij de realisatie van het besturingssignaal E voor de synchrone I/O-komponenten; dit omdat deze bouwstenen zèlf geen vektornummer kunnen leveren. De 68008 schakelt dan om op de level-x-interruptautovector.

Teneinde de handshaking-procedure (terugmelding) te vereenvoudigen, wordt het synchronisatiesignaal E tevens gebruikt voor de besturing van asynchroon geheugen. De omschakeling tussen synchroon en asynchroon E-signaal wordt gerealiseerd met een 74LS00/1. Ook de gebruikelijke handshakesignalen $\overline{\text{MRDY}}$, $\overline{\text{VPA}}$ en $\overline{\text{VMA}}$ zijn op de computerbus aangesloten, zodat ook de kaarten van andere fabrikanten, die met deze signalen werken, kunnen worden gebruikt. Figuur 4b laat de draadbruggen op de CPU-kaart zien.

Video- en keyboard-kaart

Op de videokaart (figuur 5) is een poort aanwezig (type 6821) ten behoeve van een toetsenbord. Op het eerste gezicht ziet dat er een beetje vreemd uit! Er schuilt echter een goed idee achter. Indien namelijk meerdere van deze kaarten aangesloten zijn, dan kunnen er meerdere toetsenborden en beeldschermen worden aangesloten! Maar daar gaan we hier niet verder op in.

De video-signalen komen uit een 6845-CRT-controller, die na een RESET wordt geprogrammeerd. Via vier multiplexers (74LS17) bestuurt de 6845 het videogeheugen (6164). In deze 8K-x-8-RAM zijn alle ASCII-data opgeslagen die in beeld verschijnen. Zodra de 68008 een ASCII-teken voor weergave verstuurt, worden de vier multiplexers omgeschakeld en het bitpatroon op de adresbus komt in het videogeheugen terecht. De lokale databus van dit geheugen leidt naar de karaktergenerator (2764). Deze vertaalt het ASCII-teken in een passend patroon van oplichtende beeldpunten. Op het beeldscherm is een teken uit meerdere video-lijnen opgebouwd. De CRT-controller bepaalt via de adreslijnen A0...A3, die naar de karaktergenerator voeren, welke video-lijn van het karakter momentaan wordt geschreven. Het synchrone schuifregister (74LS166) zet het parallelle beeldpuntenpatroon op de uitgang van de karaktergenerator (D0...D7) om in een serieel videosignaal, dat op de uitgang QH aanwezig is. De klok voor dit schuifregister komt uit een 14-MHz-kwarts-oscillator, die tevens een teller

(74LS161) stuurt. Ieder karakter op het beeldscherm is acht beeldpunten breed. Indien een karakter na acht klokpulsen het schuifregister is uitgeschoven, triggert de teller via uitgang Qc de 6845. Daarmee adresseert de CRT-controller dus het volgende weer te geven karakter. Via twee dioden en vier EXOF-poorten worden de synchronisatiesignalen aan het videosignaal toegevoegd. Een transistor zorgt voor de 50-Ω-uitgangsimpedantie van het komplete (composite) videosignaal.

Een 74LS688 en een 74LS139 nemen de adresdekodering voor hun rekening (onderste helft van het schema). Het basisadres van de 6845 is $FFC28 en het basisadres van de 6821 is $FFC2A. Verder is nog een dubbele flipflop (74LS74) gebruikt. Een helft hiervan zorgt ervoor dat de cursor-adressering synchroon loopt met de karakterteller. De andere helft hiervan onderdrukt beeldruis tijdens de momenten dat de 68008 het videogeheugen opfrist.

De tweede helft van de 74LS139 stuurt de lijn $\overline{WE}$ van het videogeheugen. Het R/$\overline{W}$-signaal van de CPU wordt gesynchroniseerd met E (RAM-R/$\overline{W}$). Op deze wijze komen er slechts geldige data in het videogeheugen terecht. De beide busdrivers (74LS245) zorgen voor mooie, eenduidige signalen op de databus. Figuur 5b laat zien welke draadbruggen er op deze kaart moeten worden aangebracht.

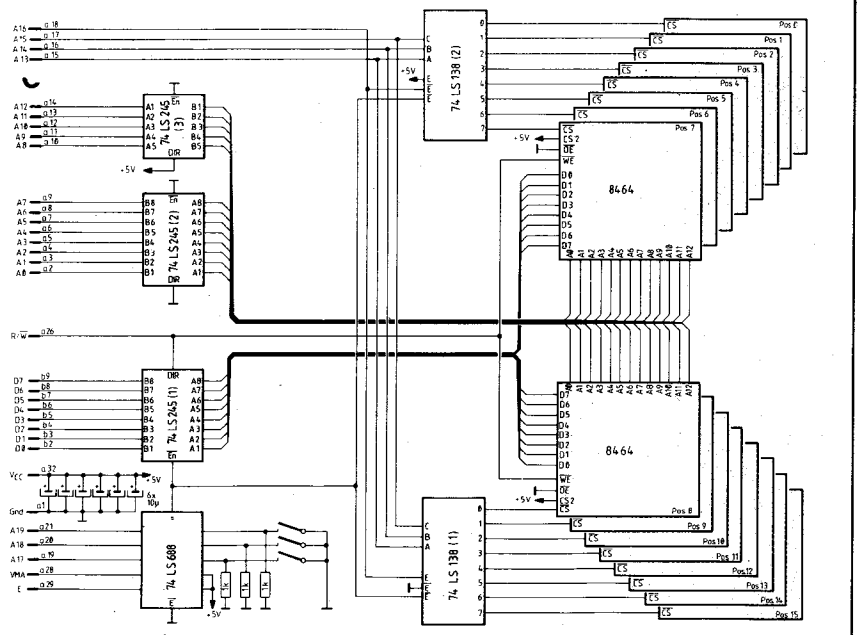
De geheugenkaarten voor de EC-68K

Zoals u in het schema (figuur 6) van de universele geheugenkaart kunt vaststellen, gaat het hier om een rechttoe-rechtaan-ontwerp. Veel valt er niet over te vertellen. De schakeling is zodanig ontworpen dat RAM- en/of EPROM-geheugenchips op de kaart kunnen worden geplaatst; kombinaties van RAM en EPROM zijn dus ook mogelijk. Dat heeft als allergrootste voordeel dat er slechts één print nodig is voor al uw geheugenbehoeften. In totaal passen er 16 IC's met een geheugeninhoud van 8K x 8 (bits) op deze geheugenkaart. Dit resulteert in een totale geheugenkapaciteit van 128 Kbyte per geheugenkaart. Zoals gezegd: RAM en/of EPROM. Twee IC's van het type 74LS138 dragen zorg voor de selektie van de 16 geheugen-IC's. De adresbus en de databus zijn op deze kaart gebufferd (3 x 74LS245). De bidirektionele buffers in de adreslijnen worden slechts in één richting gebruikt; kostenoverwegingen hebben aanleiding gegeven tot de keuze van dit buffertype. De adresdekodering is in handen van de 74LS688. Met drie DIL-schakelaars kan de geheugenkaart in een willekeurig 128K-adresblok van het 1 Mbyte grote adresbereik worden gelokaliseerd. De signalen E en $\overline{VMA}$ zijn bij de CPU-kaart besproken. Figuur 6b laat zien in welke standen de drie DIL-schakelaars voor de eerste RAM-kaart moeten worden gezet. Figuur 6c toont de situatie voor EPROM's.

Het floppy-subsysteem

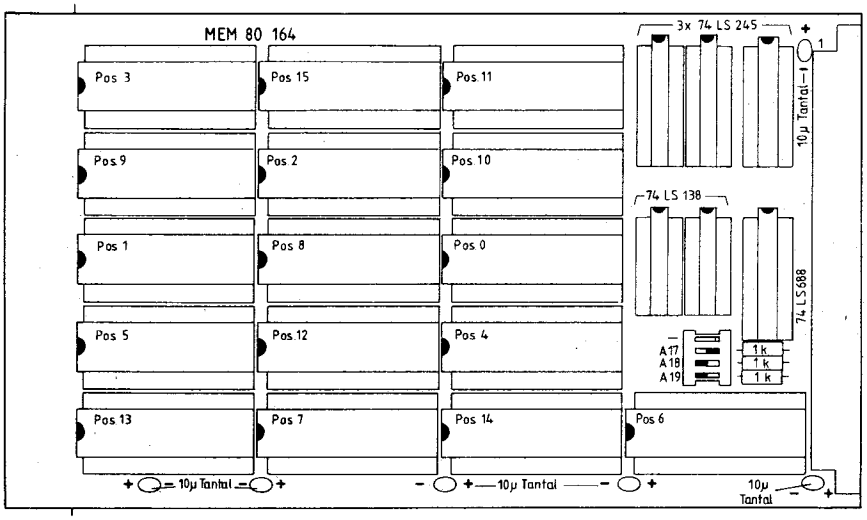
Het schema van het floppy-subsysteem is in figuur 7 te zien. Aangezien geen enkele industriële floppy-controller goed genoeg

6a



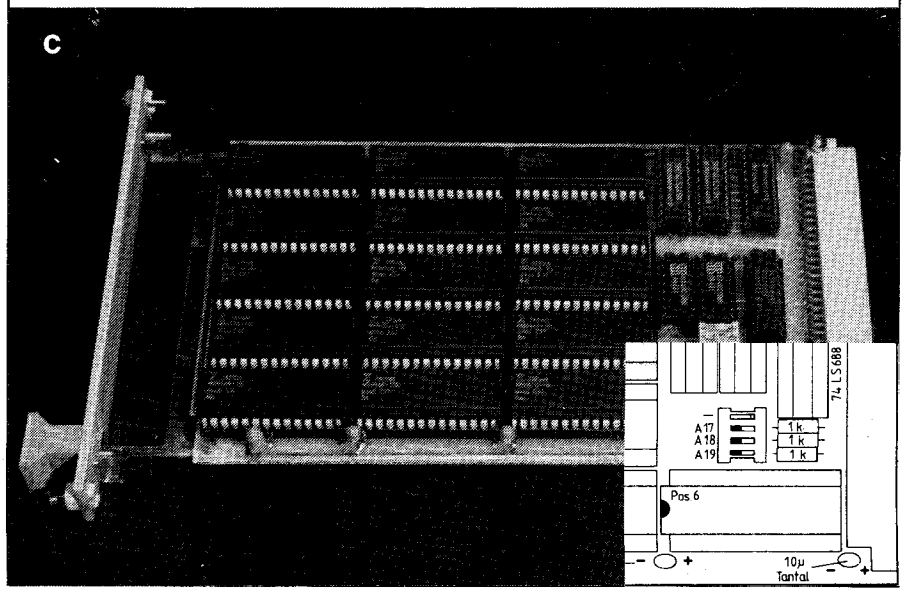
85213-6a

b

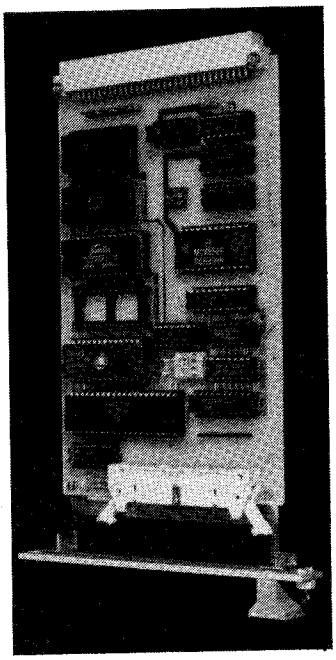


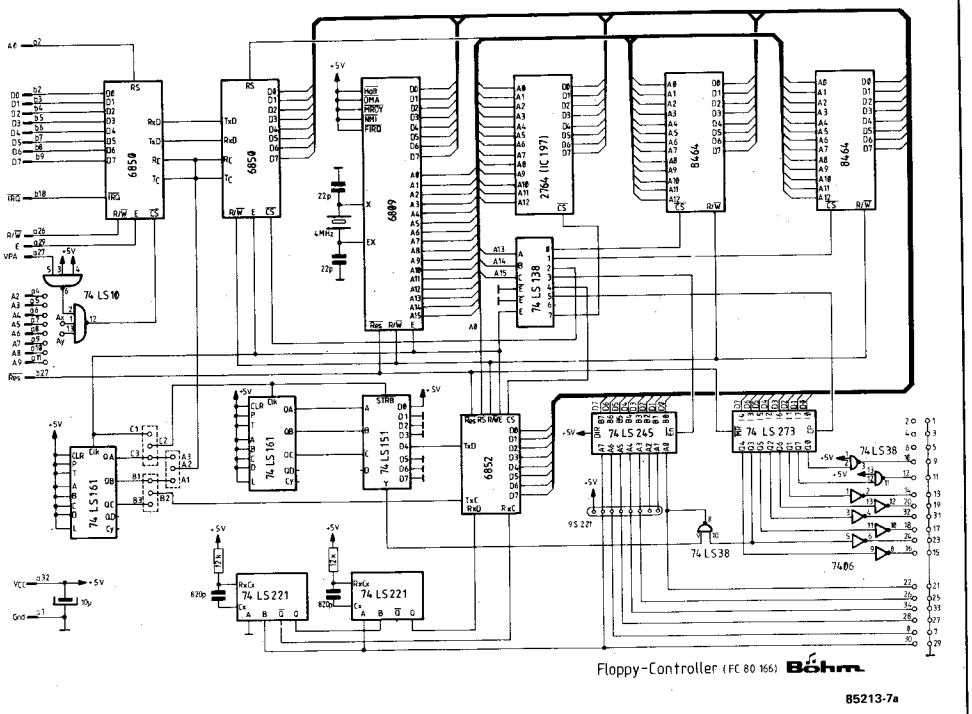
85213-6b

c

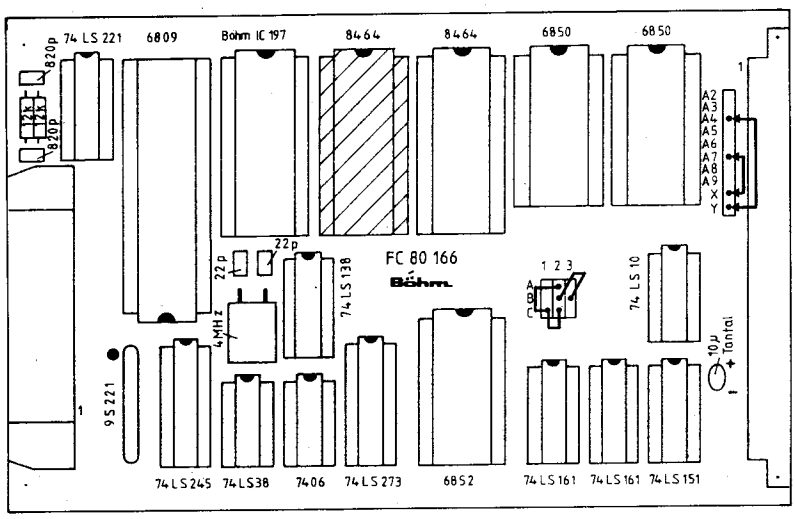


7a





b



Figuur 6a. De universele geheugenkaart van de EC-68K. Aangezien er statische geheugen-IC's worden gebruikt, kunnen ROM en RAM op eenzelfde kaart worden gekombineerd. Dat is bijzonder voordelig wanneer software die van andere systemen afkomsig is, snel moet worden aangepast aan de EC-68K. Vektoren en andere sprongadressen hoeven dan niet te worden gewijzigd, omdat de ROM en de RAM op dezelfde adressen zijn gelokaliseerd als bij de oorspronkelijke computer het geval is.

Figuur 6b. De posities van de DIP-schakelaars op de "eerste" RAM-kaart.

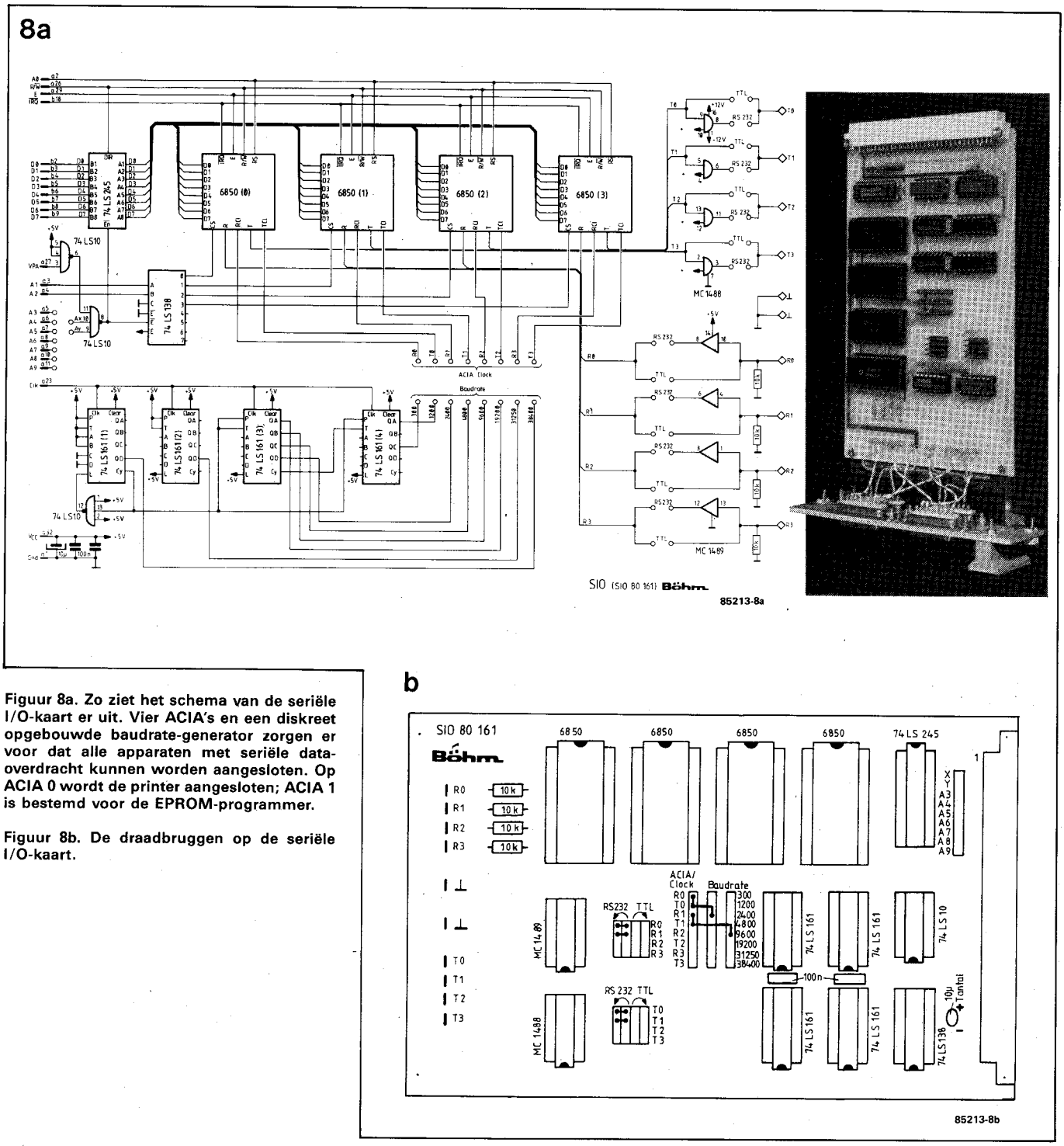
Figuur 6c. De posities van de DIP-schakelaars op de EPROM-kaart. Alleen de monitor-ROM dient op een bepaalde vastgelegde positie te worden ingestoken, zodat de processor de RESET-vektor en de initiële waarde voor de supervisor-stackpointer kan vinden. De software in de andere EPROM's is zodanig geschreven dat deze in het geheugen verplaatsbaar (relocatable) is. De andere EPROM's kunnen dus op willekeurige posities worden ingestoken. Na een RESET gaat de EC-68K automatisch op zoek naar de software in de EPROM's.

Figuur 7a. De EC-68K bezit een super-intelligent floppy-subsysteem. De 6809-processor neemt de hoofdprocessor (68008) taken uit handen, zoals de snelle toegang tot data op de diskette. Een RAM-buffer die tot maximaal 16 Kbyte kan worden uitgebreid, neemt de data van de diskette op voordat door de hoofdprocessor naar die data wordt gevraagd.

Figuur 7b. De draadbruggen op de floppy-kaart.

bevonden werd voor de EC-68K, moest er naar een andere oplossing worden gezocht. Teneinde de 68008 te bevrijden van "domme" taken, zoals het lezen van en schrijven op de floppy, werd het noodzakelijk om een intelligente floppy-interface te ontwerpen. Een 6809-microprocessor stuurt zelfstandig het dataverkeer naar de drive (loopwerk). Primitieve routines zoals: "plaats de kop op een track", "lees een sektor van de diskette", enz., zijn ondergebracht in een 2764-EPROM. Rond dit soort routines is een overkoepelend programma aangebracht: de DOS-kommando-interpreter. De kommando's die de hoofdprocessor aan de interpreter delegeert, zullen we in de volgende uitgave van Elektuur Computing uitvoerig beschrijven. Maar een voorproefje van die bespreking zullen we u niet onthouden:

* aktiveren van een drive
* uitschakelen van een drive
* het openen van een databestand
* het sluiten van een databestand
* sekwentiële toegang tot een databestand
* toegang tot de "record" van een random-databestand
* het lezen van een sektor
* het beschrijven van een sektor
* de inhoudsopgave (directory) van een diskette opvragen
* het wissen van een databestand
* de naamsverandering van een databestand
* het resetten van het aantal benutte databestanden
* het resetten van het aantal gebruikers van een databestand
* de status van een diskette opvragen
* het formatteren van een diskette



Figuur 8a. Zo ziet het schema van de seriële I/O-kaart er uit. Vier ACIA's en een diskreet opgebouwde baudrate-generator zorgen er voor dat alle apparaten met seriële data-overdracht kunnen worden aangesloten. Op ACIA 0 wordt de printer aangesloten; ACIA 1 is bestemd voor de EPROM-programmer.

Figuur 8b. De draadbruggen op de seriële I/O-kaart.

Tot zover de DOS-kommando's die de 6809-processor op de floppy-kaart in huis heeft.

Naast de EPROM hebben nog twee statische RAM-chips onderdak gevonden op de floppy-kaart (type 6164, 8K x 8). Daarvan wordt er momenteel slechts één gebruikt. De RAM wordt gebruikt om er data in op te slaan die gebruikt worden bij de verwerking van de DOS-kommando's. Tevens doet de RAM dienst als databuffer tijdens het lezen of beschrijven van een sektor van een diskette. Naast de 6809 bevindt zich nog een eigenaardige schakeling: een FIFO, opgebouwd met twee 6850-chips. Via deze FIFO verloopt de data-uitwisseling tussen de hoofdprocessor en het floppy-subsysteem. Natuurlijk had men een industriële FIFO op deze plaats kunnen zetten. Maar dat was dan wel een stukje duurder uitgevallen dan de hier gekozen oplossing, die rustig onkonventioneel mag worden genoemd. (Een FIFO is een First-In-First-Out-datamedium.)

Om data van de diskette te kunnen lezen of op de diskette te kunnen schrijven, moeten de parallelle data van het 6809-subsysteem in seriële data worden omgezet. De synchrone seriële data-adapter (SSDA) 6852 neemt deze taak voor zijn rekening. Hij vervangt het lees/schrijfregister in de gebruikelijke floppy-controllers. Op de ingang TxC is het kloksignaal van de zender aanwezig; dit signaal is afgeleid van het E-signaal van de 6809. De zenderdata worden via de uitgang TxD uitgeschoven. Via de ingang RxD leest de 6852 de floppy-data serieel in. Het schuifkloksignaal voor de ontvanger wordt aan de ingang RxD toegevoerd. De seriële data die het loopwerk levert, kan de 6852 niet direkt verwerken omdat de data nog in het kloksignaal verpakt zit. De datascheider is opgebouwd met twee monoflops (74LS221) en zorgt voor de scheiding van data en kloksignaal. De beide signalen worden aan de SSDA toegevoerd.

Ook de van de 6852 afkomstige seriële data kan het loopwerk niet zonder meer verwerken. Voordat het loopwerk de zenderdata kan verwerken, moeten het kloksignaal en de data tot één signaal worden gekombineerd. Twee tellers (74LS161) en een multiplexer (74LS151) moduleren het seriële datasignaal op het kloksignaal. De eerste teller deelt het 1MHz-E-signaal van de 6809 door acht. De tweede teller stuurt de multiplexer. Het bitpatroon op de lij-

nen A, B en C legt vast welke van de acht ingangen met de uitgang Y wordt verbonden. De logische nivo's op de ingangen van de multiplexer zijn zodanig gekozen dat op de uitgang Y het volgende signaal onstaat:

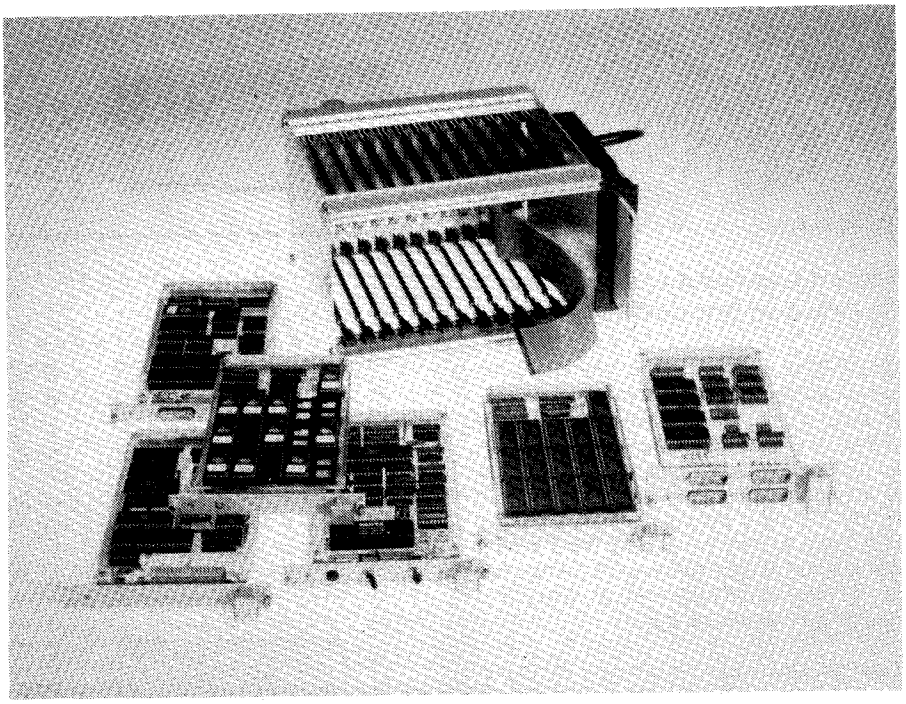
een klokbit
drie nulbits
een databit (logisch nul of één)
drie nulbits

Een in de uitgang opgenomen poort (74LS38) verhindert dat data "zonder toestemming" op een diskette wordt geschreven. De leiding WE moet logisch één zijn, wil er sprake zijn van het schrijven van data naar de diskette.
Naast de schrijf/leesdata heeft het loopwerk nog andere besturingssignalen nodig. Zo moet de kop bijvoorbeeld een stap naar binnen of naar buiten worden geschoven. Of het loopwerk moet worden medegedeeld dat de voor/achterzijde van een diskette is geselekteerd, of dat data gelezen danwel geschreven moet worden. Deze besturingssignalen schrijft de 6809-processor naar een tussengeheugen (latch type 74LS273). De drivers die in de uitgangen van dit tussengeheugen zijn opgenomen (met open-kollektoruitgang) zijn aangepast op de "totempaal"-uitgangen van de tussengeheugens, zoals de gestandaardiseerde Shugart-bus van floppy-loopwerken die kent.
De adresdekodering van en voor het floppy-subsysteem is in handen van een 74LS138. Via het adres $FFC90 adresseert de 68008, de hoofdprocessor van de EC-68K, de FIFO op de floppy-kaart.
Figuur 7b toont de draadbruggen op de floppy-kaart.

De seriële I/O

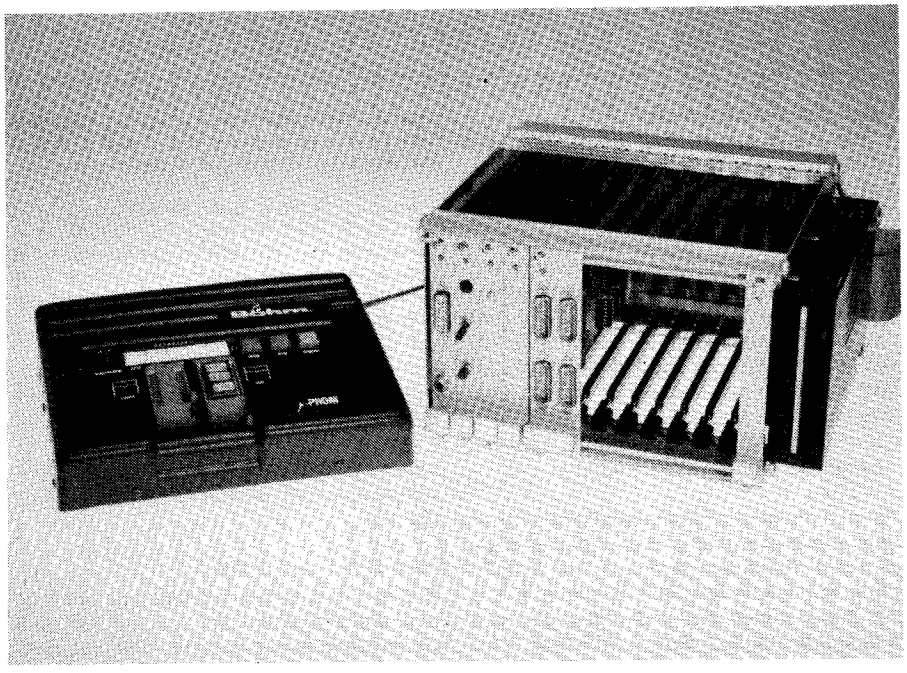
Voor de EC-68K staan momenteel drie verschillende I/O-kaarten ter beschikking:

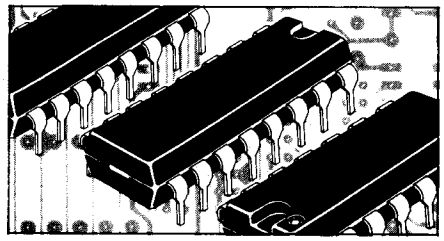
1) Vier RS232/V24-kanalen op één kaart



2) Twee MIDI-kanalen op één kaart
3) Een universele seriële/parallelle printer-interface

We zullen hier uitsluitend de seriële RS232/V24-interface bespreken. De MIDI-interface, inmiddels ingeburgerd in de moderne muziek-elektronica en daar niet meer weg te denken, zullen we in een van de volgende uitgaven van Elektuur Computing beschrijven.
Figuur 8 toont het schema van de vier seriële RS232-kanalen. Zoals u in het schema snel zult ontdekken, kunnen de beide 12-V-spanningen op de seriële uitgangen worden doorgeschakeld. In bijzondere gevallen kan ook gekozen worden voor +5V en massa. Een dergelijk klein detail blijkt in de praktijk zeer nuttig te zijn.
Elke ACIA (6850) bezet twee geheugenplaatsen. Alle vier 6850's zijn volledig, dat wil zeggen zonder hiaten, gedekodeerd. Het basisadres van ACIA 0 is $FFC30. Een 74LS138 en een 74LS10 nemen de adresdekodering voor hun rekening. De databus is in twee richtingen gebufferd met behulp van een 74LS245. In het onderste gedeelte van het schema treft u de baudrate-generator aan. Deze bestaat uit vier tellers (74LS161) en levert acht verschillende transmissiesnelheden waaruit kan worden gekozen (300 baud...38,4 Kbaud).
Ook de aansluitingen van de konnektoren zijn duidelijk zichtbaar in het schema. Momenteel worden slechts ACIA 0 en ACIA 1 gebruikt. Op de eerste ACIA kan een seriële printer worden aangesloten. De tweede ACIA is gereserveerd voor de "Dr. Böhm-EPROM-programmer". Alle EPROM's van 2716 tot en met 27256 kunnen hiermee worden geprogrammeerd.
Figuur 8b toont de positie van de draadbruggen op de seriële I/O-kaart.





EC-68 — de Elektuur-Flex-computer

Een professioneel systeem voor de prijs van een home-computer

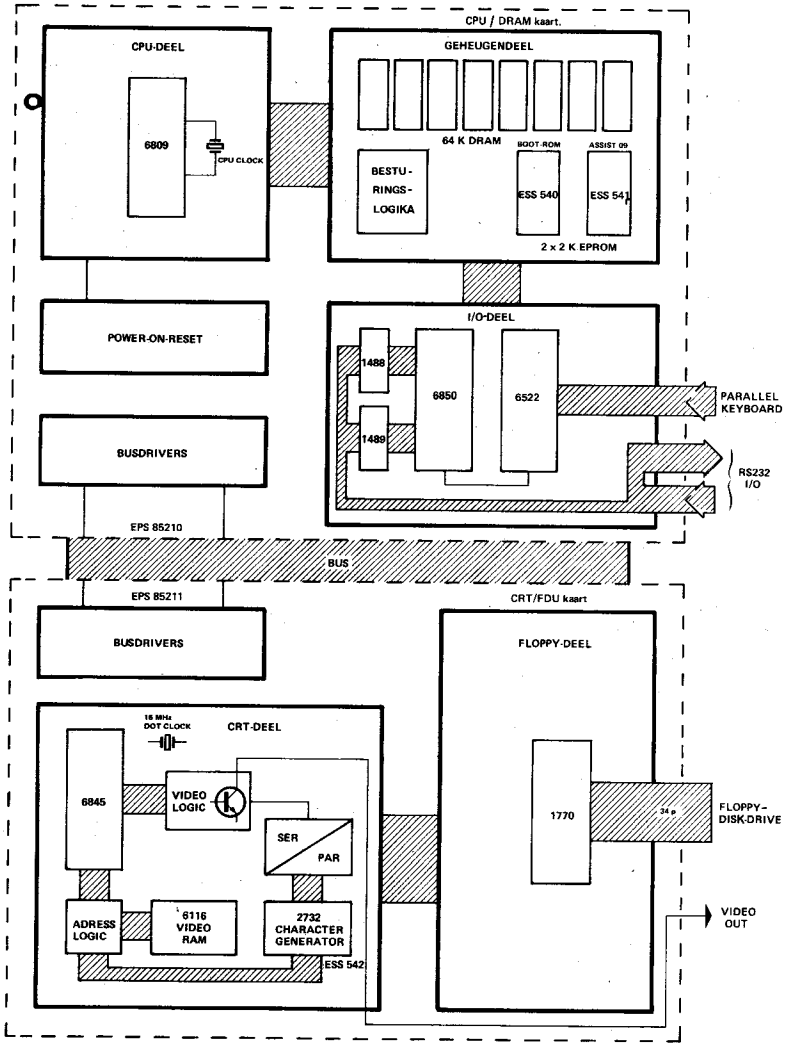
De EC-68 is een klein maar krachtig computersysteem, dat ook voldoet aan professionele eisen. Dit systeem biedt een bijzonder gunstige prijs/kwaliteitsverhouding als we kijken naar de beschikbare hoeveelheid software en de daaruit voortvloeiende gebruiksmogelijkheden. Een klein systeem met zeer veel mogelijkheden dus. Dat zult u na het lezen van de diverse Flex-artikelen in deze EC beslist met ons eens zijn.

Zoals de titel al laat zien, werkt de EC-68 met het operating system Flex. Deze naam zal de meeste van onze lezers waarschijnlijk weinig of niets zeggen. Daarom besteden we in het software-gedeelte van deze Computing-uitgave (en in de komende nummers) de nodige aandacht aan dit interessante systeem. Op de computermarkt is, zoals we reeds hebben opgemerkt, erg veel software voor Flex verkrijgbaar, waaronder diverse bekende programma's. Maar alvorens nog meer tekst wordt gewijd aan die software, moet eerst de computer zelf worden beschreven.

De systeem-hardware

We zullen in dit gedeelte niemand vermoeien met uitgebreide verhandelingen waarbij de funktie van elk weerstandje uit de doeken wordt gedaan. We volstaan met de bespreking van het uitgebreide blokschema.
Een computersysteem voor huishoudelijk of semi-professioneel gebruik bestaat altijd uit een vaste set funktionele bouwstenen. Voor een Flex-computer zijn de volgende delen onontbeerlijk: een CPU-deel, een geheugen-deel, een videodeel en een koppeling naar een snel magnetisch opslagmedium (een floppy-disk-interface dus). Met deze opsomming zijn ook de hoofdbestanddelen van het blokschema gegeven. Lezers die al vaker een bouwbeschrijving van een computersysteem hebben gelezen, komen, na het maken van een eenvoudig rekensommetje, tot de konklusie dat het systeem naar alle waarschijnlijkheid zal bestaan uit een viertal eurokaarten. Fout! Door een uitgekiend ontwerp zijn voor de bouw van deze computer slechts twee printen van eurokaartformaat nodig. Op één print zit het CPU- en geheugendeel, op de tweede de floppy-disk-interface en het videogedeelte.
Het blokschema van de Flex-computer is gegeven in figuur 1.
Natuurlijk bestaat een computer niet alleen uit twee eurokaarten met elektronica. Verder hebben we nog nodig:

1



* Een parallel-keyboard (ASCII).
* Een monitor met 75-Ω-ingang en een minimale bandbreedte van 16 MHz.
* Minimaal een floppy-drive met 40 of 80 tracks en een Shugart-bus. Het loopwerk moet dubbelzijdig werken, aangezien de systeemdiskette dubbelzijdig is.
* Een netvoeding die de volgende spanningen levert: 5 V/3 A, +12 V/2 A (voor de floppy-drives), +12 V/250 mA en —12 V/250 mA (voor de RS-232-interface). Een voeding die wat meer stroom kan leveren, kan natuurlijk nooit kwaad.
* Een bus-print met konnektors, bijvoorbeeld de Omnibus (EPS 83102).

Aangezien we hier te maken hebben met een eurokaarten-computer, gelden voor de opzet dezelfde regels als we in EC1 hebben beschreven voor de Octopus 65. Dat geldt overigens ook voor de kast waarin de EC-68 wordt gebouwd. Het enige verschil met de Octopus, kwa opbouw, is het gebruik van dubbelzijdige drives.

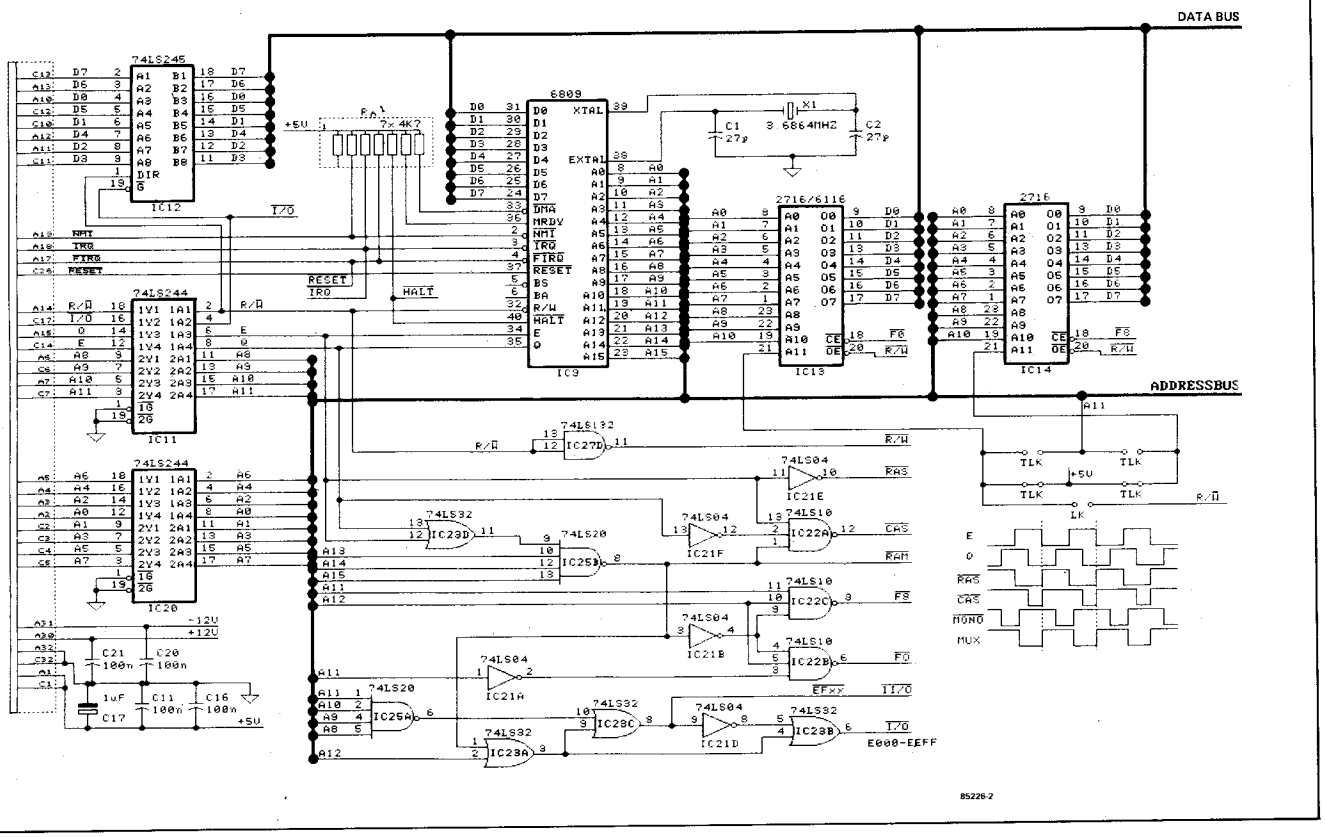
Figuur 1. Dit blokschema geeft een overzicht van de diverse funktie-delen in de EC-68 en de verdeling hiervan over twee eurokaarten.

Figuur 2...4. Drie door de computer getekende schema's zijn nodig om de hele schakeling van het CPU-DRAM-gedeelte weer te geven. Het is wel duidelijk dat op de print flink met de beschikbare ruimte moest worden gewoekerd.

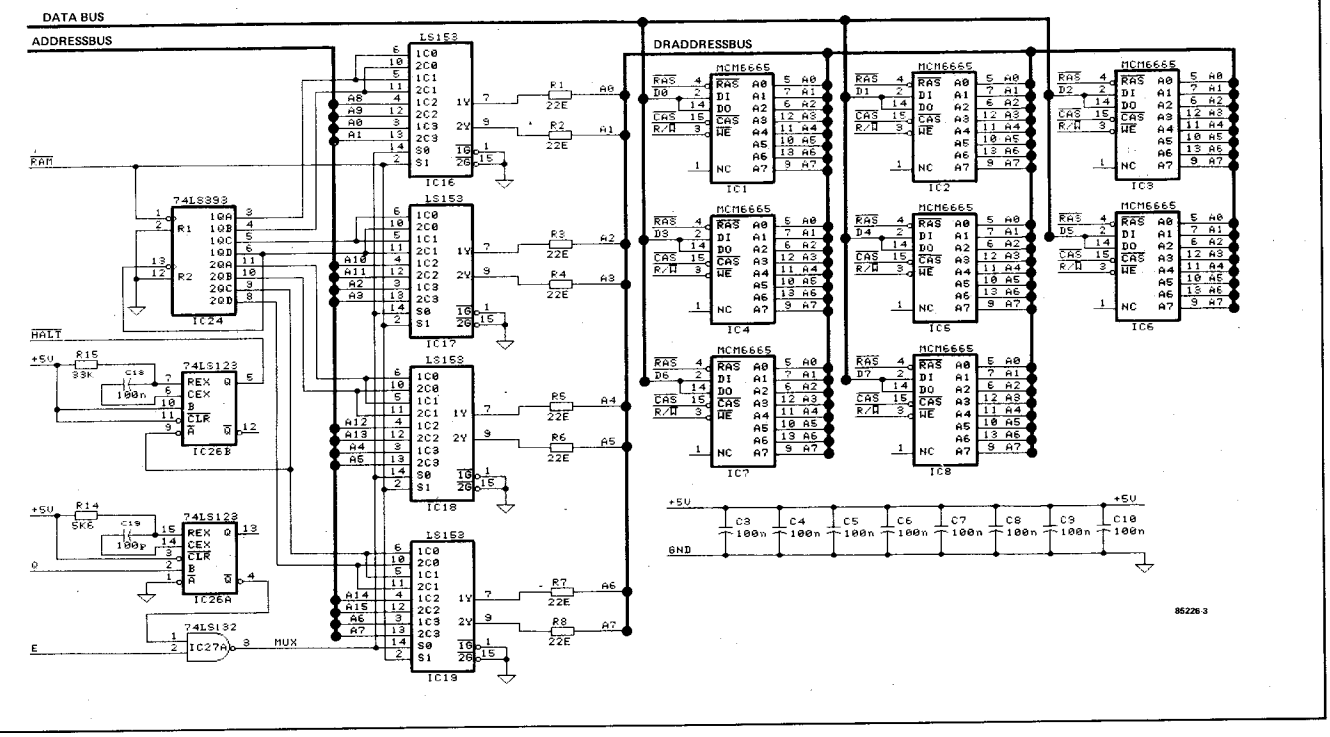
Het CPU-gedeelte

Het hart van elke computer is de microprocessor. Flex is een operating system dat is geschreven voor de 6800 en 6809.

2



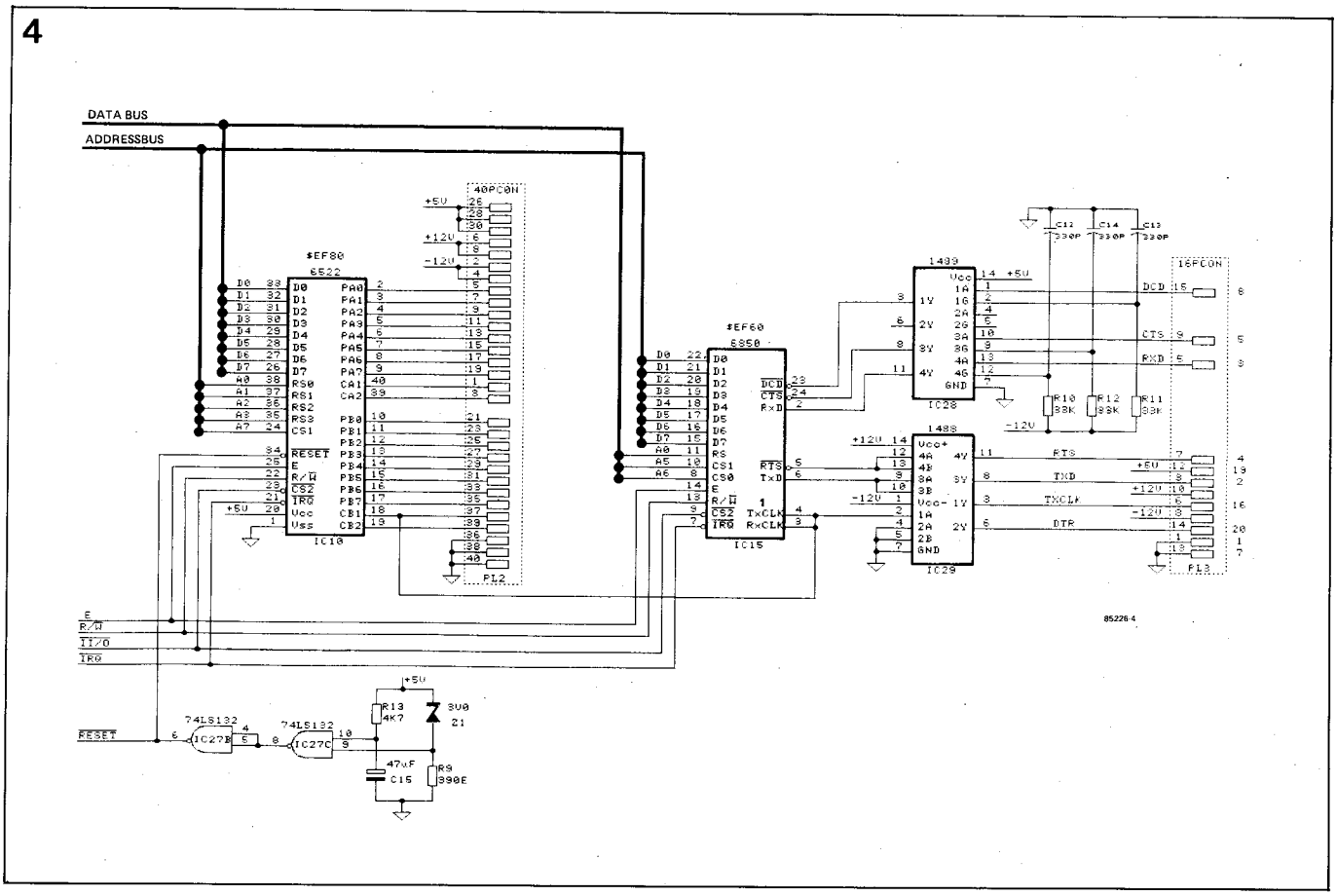
3



De CPU in de Elektuur Flex-computer is de 6809 (IC9). De Flex-versie hiervoor heet dan ook toepasselijk "Flex-9". Figuur 2 toont het schema van het gedeelte rond de 6809. Zonder software kan ook de 6809 niet werken. De firmware (Assist-09 en Bootrom) bevindt zich in twee EPROM's van het type 2716 (IC13 en IC14). Standaard hebben we te maken met een systeem dat draait op een klokfrekwentie van 3,6864 MHz. Men heeft hier gekozen voor deze frekwentie, omdat hiervan ook de baudrates van de AC1A worden afgeleid. Door de frekwentie van de systeemklok te verdubbelen (de schakeling rond X1), het toepassen van B-types voor de 68xx-IC's, het gebruik van snellere dynamische RAM's en het veranderen van de timing van die RAM's kan men pogen over te schakelen op een 2-MHz-systeem. We sluiten de mogelijkheid van omschakeling naar een hogere systeemklok niet uit, maar raden het op dit moment echter af vanwege de te verwachten problemen met de timing.

In het begin van deze hardwarebeschrijving werd vermeld dat het hele systeem is opgebouwd rond een tweetal printen van eurokaartformaat. Het gebruik van bus-georiënteerde printen maakt het aansluiten van uitbreidingen (bijvoorbeeld RAM-disk, PROM-programmer) bijzonder eenvoudig. IC11, IC12 en IC20 zijn

4



de bus-drivers die adres-, data- en controlbus-signalen van en naar de andere gedeeltes van het systeem sturen. Bij nadere bestudering van figuur 2 blijken nog twee gedeeltes niet besproken te zijn, de draadbruggen rechtsonder en de verzamelingen poorten middenonder.
Met de draadbruggen kan men kiezen tussen een 2716 of 6116 op de plaats van IC13. Het is namelijk mogelijk deze CPU/DRAM-kaart voor andere toepassingen in te zetten dan als CPU/RAM voor de Elektuur Flex-computer. In onze toepassing wordt voor IC 13 een 2716 gebruikt. In het geval dat een 6116 wordt gebruikt, of een ander type EPROM, moet de adressering (en mogelijk het R/W-signaal) met behulp van draadbruggen aangepast worden. De in figuur 2 aangegeven draadbruggen geven de situatie weer bij gebruik van twee EPROM's van het type 2716. Over de inhoud van de twee 2716's volgt in het software-gedeelte van deze EC uitgebreidere informatie.
De serie poorten dient voor het maken van een aantal chip-select-signalen en de RAS- en CAS-signalen voor de dynamische RAM's.

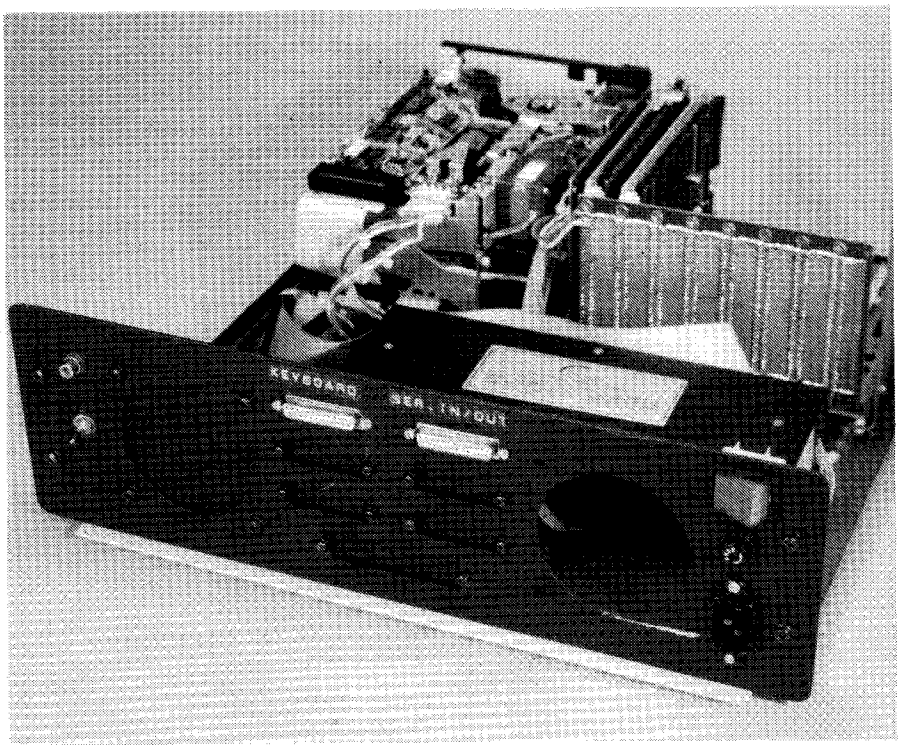
Het geheugen en de I/O

Zoals zojuist werd verteld, worden op de CPU/RAM-print dynamische RAM's (4164) ingezet. In figuur 3 is het DRAM-deel in tekening gebracht.
Het eigenlijke geheugen (kapaciteit 64 K) omvat de IC's 1 t/m 8. De rest van de IC's in figuur 3 zijn besturings-IC's voor de dynamische RAM's. Een timing-diagram vindt u in figuur 2 (rechtsonder).
Verder was er op de eerste print nog plaats voor I/O-chips. In figuur 4 is aangegeven hoe die I/O in elkaar zit. De kommunikatie met de buitenwereld kan op twee manieren plaatsvinden. De data worden parallel ontvangen (bijvoorbeeld van een keyboard) en verzonden (Centronics-interface voor een printer) of serieel ontvangen en verzonden (RS-232 interface).
Voor de menselijke kommunikatie met de computer wordt gebruik gemaakt van een parallel keyboard. De kommunikatiechip die de informatie van het toetsenbord ontvangt, is de 6522 (IC10 in figuur 4). Eén 6522 heeft een tweetal 8-bits-parallelpoorten; die tweede poort komt goed van pas voor het aansturen van een zogenaamde Centronics-printer. Over het gebruik van de VIA (6522) meer in een ander artikel. De seriële kommunikatie vindt plaats met behulp van IC15, IC28 en IC29. IC15 is een 6850, een zogenaamde ACIA (een afkorting van Asynchronous Communications Interface Adaptor). Dit IC zet de parallel-informatie van de computer om in een seriële datastroom (en omgekeerd). IC28 en IC29 zijn speciale IC's die de aanpassing verzorgen tussen de RS-232-lijnen en de 6850 (IC28 voor binnenkomende RS-232-signalen en IC29 voor uitgaande RS-232-signalen). De seriële aansluiting kan onder andere gebruikt worden voor data-overdracht via een modem. Op de systeemschijf is daartoe ook een modem-programma aanwezig (MODEM.CMD).
Kijken we nog even naar de klokingangen van de 6850 (pen 3 en pen 4 van IC15), dan zien we ook waarom er gekozen is voor zo'n merkwaardige kristalfrekwentie. De VIA-poort CB1 (pen 18 van IC10) wordt door de firmware als klokdeler geprogrammeerd, waardoor de systeemklok wordt gedeeld tot ACIA-klok. Dat kan alleen maar als de systeemklok een geheel veelvoud is van de gangbare ACIA-frekwenties. Daar de 6809 de kristalfrekwentie door vier deelt en de systeemklok zo dicht mogelijk in de buurt van 4 MHz moet liggen, volgde hieruit de frekwentie van 3,6864 MHz. Door deze opzet zijn een aparte klokgenerator en deler voor de ACIA niet nodig. Dat bespaart niet alleen geld, maar ook plaats op de print. Alleen door gebruik te maken van zulke slimme ideeën is het mogelijk om zoveel hardware op één eurokaart onder te brengen.
De aansluitgegevens van de 40-polige konnektor met de parallel-poorten voor keyboard en printer zijn afgebeeld in figuur 5. Figuur 6 geeft de aansluitingen van de 16-polige konnektor voor de seriële interface. In het figuurbijschrift vindt u de aansluitgegevens voor de 25-polige D-konnektor die gewoonlijk bij RS-232-interfaces wordt toegepast.
Omdat we te maken hebben met een bus-georienteerd systeem is het niet overbodig de signalen en spanningen op de bus te geven. Zie hiervoor figuur 7. Ter afsluiting merken we nog op dat de computer bij het opstarten een systeem-reset genereert. Daarvoor is de schakeling rond IC27 (poort B en C) verantwoordelijk (zie figuur 4). U hoeft dus niet te beginnen met het drukken van de reset-knop na het inschakelen van de voedingsspanning.
Feitelijk staat ons niets meer in de weg om de CPU/DRAM-print van komponenten te voorzien. Hierbij moet, in verband met de grote pakkingsdichtheid, wel nauwkeurig gewerkt en gesoldeerd worden. Door die grote dichtheid kon ook niet worden vastgehouden aan de typische Elektuur-layout voor de printen. (zie de print-layout met komponentenopstelling in figuur 8).

Vergeet vooral de vijf draadbruggen niet! Het spreekt voor zich dat u de met gelijke letters aangegeven punten met elkaar moet verbinden (dus A met A, B met B, etc). Bij IC13 worden de draden direkt aan de aansluitpennen van het EPROM-voetje gesoldeerd, waarna de overgebleven uiteinden van de draden tussen IC20 en IC21 worden vastgesoldeerd. Let hierbij goed op! Als u de print met de komponentenzijde naar u toe houdt, met de buskonnektor naar links, dan zijn de onderste vijf soldeerplaatsen tussen IC20 en IC21 de aansluitingen A...E. Het daarboven liggende soldeerpunt is alleen een doorgemetalliseerd gat en mag verder niet gebruikt worden! Het is verder raadzaam IC-voeten te gebruiken, alsmede de in de onderdelenlijst aangegeven konnektoren. De twee EPROM's kunnen in de Elektuur Software Service geprogrammeerd worden, waarbij het linker IC (IC13 dus) moet worden voorzien van het programma Bootrom en het rechtse (IC14) van Assist-09. De ESS-nummers vindt u op de inhoudspagina onder de kolofon.

Video- en floppy-interface

Op de tweede print bevindt zich het resterende stuk elektronica dat nodig is voor het basissysteem, het videogedeelte en de floppy-disk-interface.
Een onontbeerlijk IC voor het verwerken van digitale signalen tot videobeelden is de CRTC (Cathode Ray Tube Controller).



Hiervoor is de bekende 6845 genomen. Helaas zijn er echter nog wel wat meer IC's nodig om alles op de juiste manier te kunnen verwerken (zie figuur 9 en 10).
Het beeldscherm is opgebouwd uit 24 regels van elk 80 tekens. Wel geteld gaan er dus op een scherm 80 x 24 = 1920 tekens. Ergens in de computer moet de beeldscherminhoud opgeslagen worden. Daarvoor is een 2-Kbyte-RAM nodig. Het ligt voor de hand als 2-K-RAM een 6116 te kiezen. In figuur 9 is dat IC 20. In dit IC worden de tekens opgeslagen in de vorm van ASCII-kodes. Het opslaan geschiedt via IC1 en IC10, terwijl de adressering gebeurt door IC17, IC18 en IC19.

5

CA-1	o - 1	2 - o	- 12 volt	
CA-2	o - 3	4 - o	+ 12 volt	
PA-0	o - 5	6 - o	+ 12 volt	
PA-1	o - 7	8 - o		
PA-2	o - 9	10 - o		
PA-3	o - 11	12 - o		
PA-4	o - 13	14 - o		
PA-5	o - 15	16 - o		uit
PA-6	o - 17	18 - o		
PA-7	o - 19	20 - o		
PB-0	o - 21	22 - o		
PB-1	o - 23	24 - o		
PB-2	o - 25	26 - o	+ 5 volt	
PB-3	o - 27	28 - o	+ 5 volt	
PB-4	o - 29	30 - o	+ 5 volt	
PB-5	o - 31	32 - o		
PB-6	o - 33	34 - o		
PB-7	o - 35	36 - o	Gnd	
CB-1	o - 37	38 - o	Gnd	
CB-2	o - 39	40 - o	Gnd	

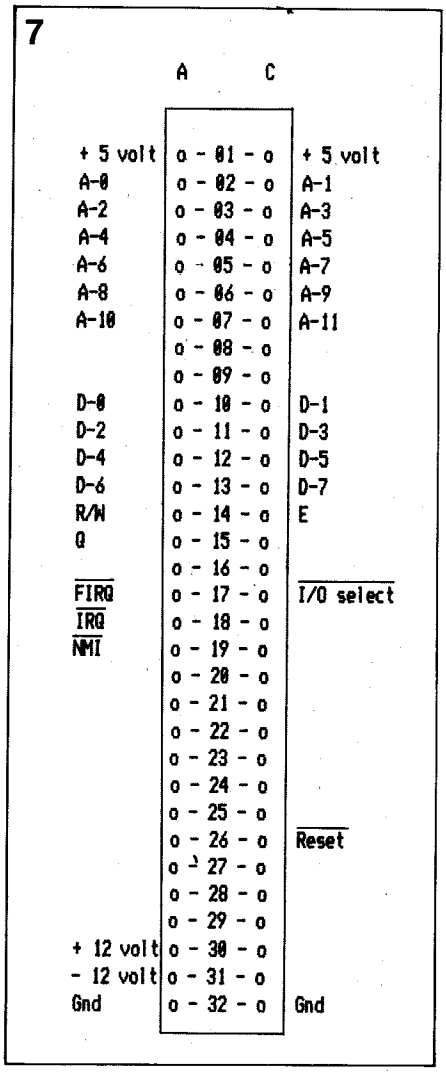
Figuur 5. De parallel-interface. De parallelport PA is de keyboard-ingang. Op PA-0 staat bit 0, op PA-1 bit 1, enz. Het strobe-signaal staat op CA-1 (data worden inlezen bij de eerste flank van de strobe-puls). Via CB-1 krijgt de ACIA (6850) zijn kloksignaal. Deze lijn is dan ook verbonden met pen 3 en 4 van de 6850. Port PB kan als parallel-uitgang voor een printer worden gebruikt (Centronics-interface).

6

GND	o - 1	2 - o	
TxD	o - 3	4 - o	
RxD	o - 5	6 - o	TxClk
RTS	o - 7	8 - o	- 12 volt
CTS	o - 9	10 - o	+ 12 volt
	o - 11	12 - o	+ 5 volt
GND	o - 13	14 - o	DTR
DCD	o - 15	16 - o	

Figuur 6. Dit zijn de spanningen en signalen zoals ze op konnektor 2 van de CPU/DRAM-kaart voorkomen. Deze aansluiting wordt gebruikt voor datakommunikatie volgens de RS-232-standaard. De standaard-RS-232-konnektor (die op de kast van de computer wordt gemonteerd) moet als volgt worden aangesloten:
01 = Gnd (ground)
02 = TxD (transmit data)
03 = RxD (receive data)
04 = RTS (ready to send)
05 = CTS (clear to send)
06 = niet gebruikt
07 = Gnd (ground)
08 = DCD (data carrier detect)
09
...
15 = niet gebruikt
16 = TxClk (transmit clock)
17 = niet gebruikt
18 = niet gebruikt
19 = +5 volt
20 = DTR (data terminal ready)
21
...
25 = niet gebruikt

7



Met het inlezen van de RAM zijn we slechts halverwege de klus, omdat de inhoud van de RAM nog in voor ons leesbare vorm op het beeldscherm moet worden omgezet. In IC 21 (een EPROM van het type 2732) ligt voor elk ASCII-karakter de punt-matrix opgeslagen. IC12 (74LS377) dient als tussengeheugen. De ASCII-kode wordt gebruikt als adres voor de 2732, zodat het bijbehorende puntpatroon aan de uitgangen van de EPROM verschijnt.

De output van de 2732 wordt vervolgens omgezet in een serieel signaal (IC 22), gesommeerd met een aantal andere signalen en vervolgens versterkt tot een voor een monitor geschikt videosignaal (T1). De zogenaamde dot-clock-frekwentie (de frekwentie voor de klok-ingang van IC 22) is 16 MHz en wordt gegenereerd door een oscillator die bestaat uit een aantal inverters (IC13). De CRTC is IC2; dit IC koördineert de diverse besturingssignalen voorhet video-proces. Verder maakt

Onderdelenlijst CPU/DRAM-kaart

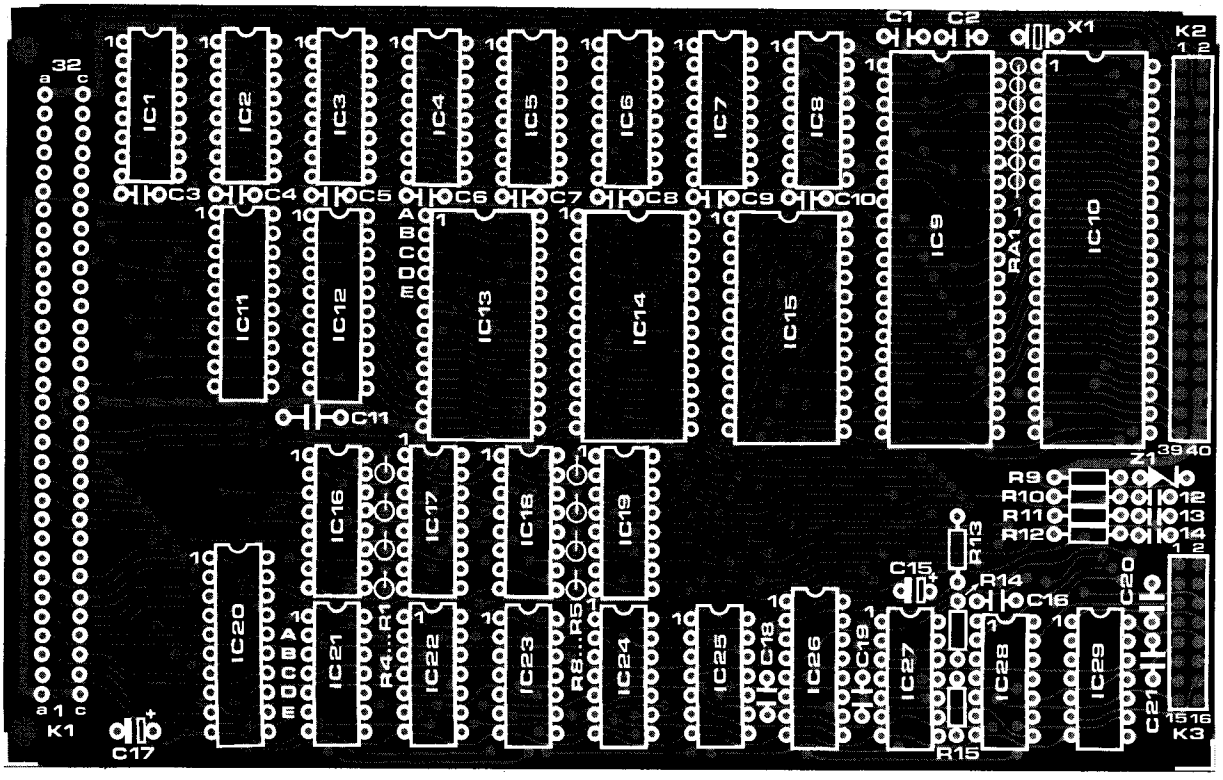
Weerstanden (allemaal 1/8 W):
R1...R8 = 22 Ω
R9 = 390 Ω
R10...R12,R15 = 33 k
R14 = 5k6
Ra1 = 7 × 4k7 (SIL-array of 7 aparte weerstanden)

Kondensatoren:
C1,C2 = 27 p
C3...C11,C16,C18,C20,C21 = 100 n ker.
C12...C14 = 330 p ker.
C15 = 47 µ/16 V
C17 = 1 u/16 V
C19 = 100 p ker.

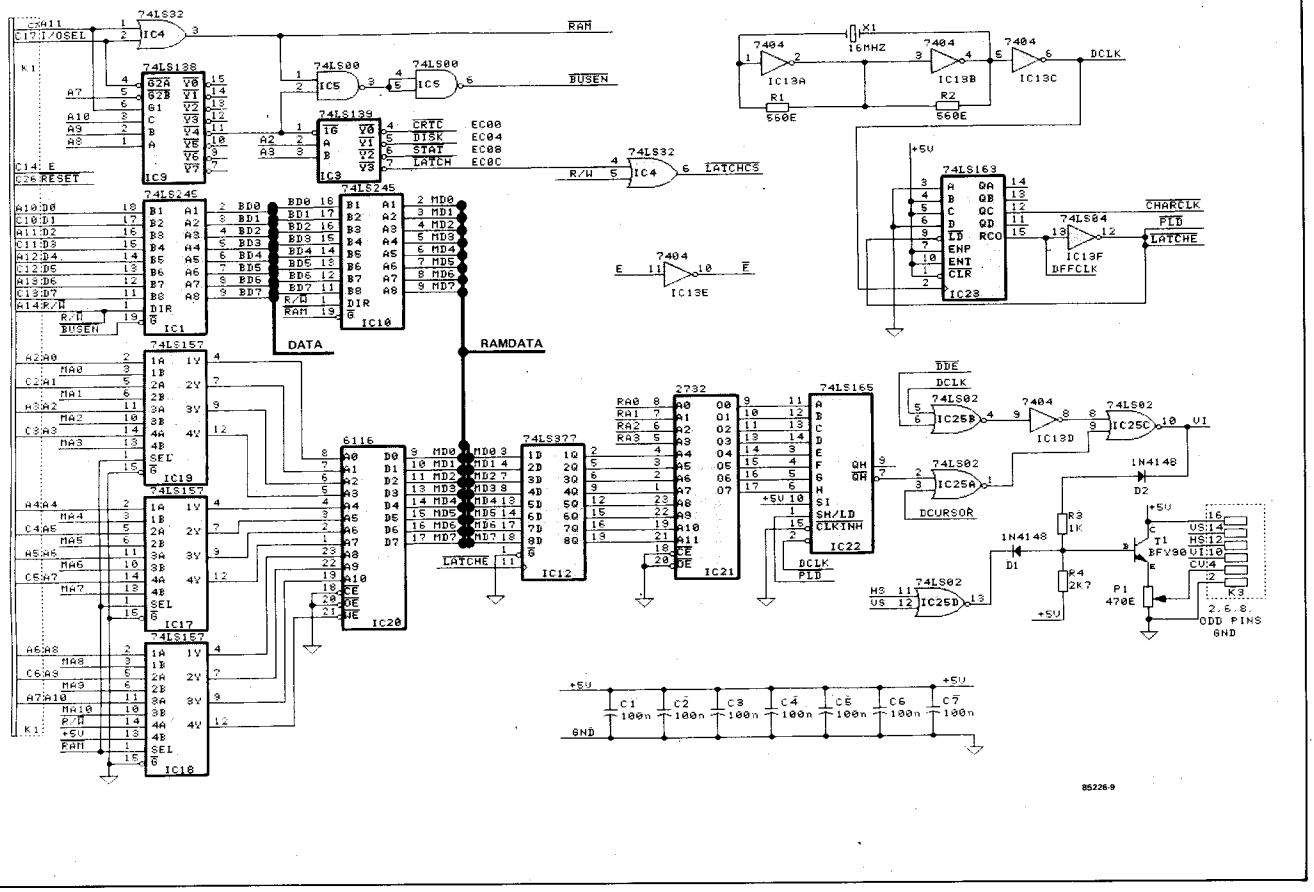
Halfgeleiders:
Z1 = 3 V/400 mW zenerdiode
IC1...IC8 = 4164 (200 ns voor basisversie)
IC9 = 6809
IC10 = 6522
IC11,IC20 = 74LS244
IC12 = 74LS245
IC13 = 2716 (ESS 540)
IC14 = 2716 (ESS 541)
IC15 = 6850
IC16...IC19 = 74LS153
IC21 = 74LS04
IC22 = 74LS10
IC23 = 74LS32
IC24 = 74LS393
IC25 = 74LS20
IC26 = 74LS123
IC27 = 74LS132
IC28 = 1489 (75189)
IC29 = 1488 (75188)

Diversen:
X1 = kristal 3,6864 MHz (klein model)
K1 = 64-polige buskonnektor, male, volgens DIN 41612
K2 = 40-polige male konnektor voor flat-cable en bijbehorende steker, bijv. Molex nr. 5342-40-GS1 en 5320-40-BGS1
K3 = 16-polige male konnektor voor flat-cable en bijbehorende steker, bijv. Molex nr. 5342-16-GS1 en 5320-16-BGS1

8



9



Figuur 7. De aansluitgegevens van de bus voor het Elektuur-Flex-systeem. Verschillende pennen zijn nog beschikbaar voor toekomstige uitbreidingen. Let wel even goed op: De bus-aansluitingen van de EC-68 komen *niet* overeen met die van de bekende Elektuur-bus.

Figuur 8. De komponentenopstelling van de CPU/DRAM-kaart. In totaal 29 IC's zitten dicht naast elkaar op deze print.

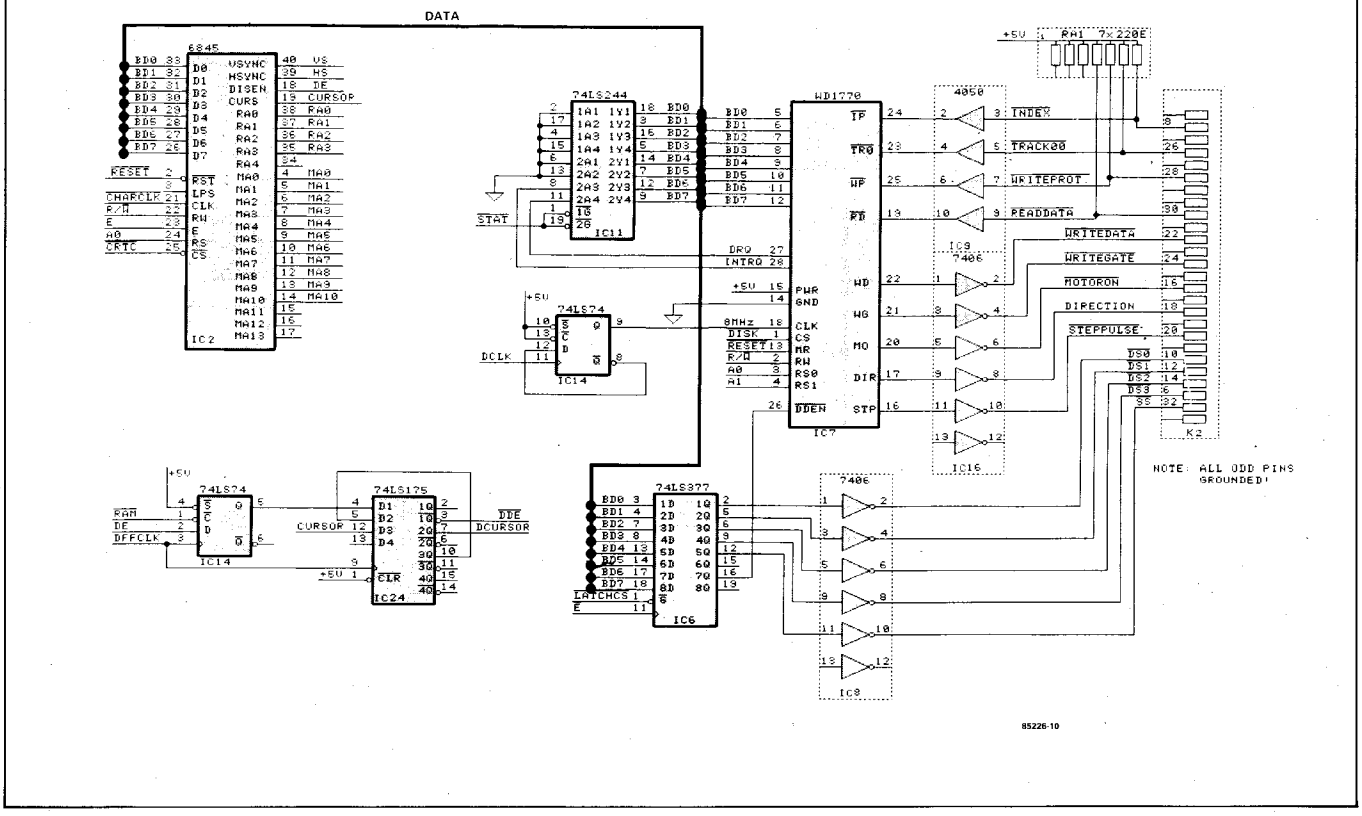
Figuur 9 en 10. Ook de schema's van de FDC/CRT-kaart zijn door de computer getekend. Er zitten op deze print iets minder IC's dan op de CPU/DRAM-kaart. Dit is vooral te danken aan de toepassing van een sterk geïntegreerde floppy-controller.

de 6845 de horizontale en vertikale synchronisatiesignalen voor de monitor. De CRTC wordt op zijn beurt weer bestuurd door de CPU (6809); hij is namelijk geen intelligente video-controller en kan dus niet zonder een CPU werken.
De CPU adresseert de CRTC net zoals de andere periferie-chips (de VIA en ACIA). Voor de adresdekodering zijn IC9 en IC3 (linksboven in figuur 9) verantwoordelijk. De floppy-disk-controller staat getekend in figuur 10. Ook hier is de eigenlijke controller slechts één IC (de WD 1770 van Western Digital); IC nummer 7 dus. De besturingselectronica om IC 7 is een stuk eenvoudiger van opzet dan die van het videogedeelte. Dit komt omdat de WD 1770 een zeer gecompliceerd IC is. Door de verregaande integratie zijn bijvoorbeeld de instelpotentiometers (zoals die soms nog in oudere ontwerpen met FDC's voorkomen) achterwege gebleven, daar de zogenaamde data-separator ook in de FDC is geïntegreerd. Derhalve is geen enkele afregeling nodig. De 1770 is verantwoordelijk voor het schrijven en lezen van data van en naar de floppy-disk. Ook hier wordt met seriële kommunikatie gewerkt. De diverse besturingssignalen worden gebufferd door IC8, IC9 en IC16. Tevens beveiligen ze de dure WD 1770 tegen calamiteiten, wanneer in- en uitgangen zouden worden kortgesloten door foutief aanbrengen van de konnektor die de floppy-disk-drive met de floppy-disk-interface verbindt. Dit gevaar bestaat omdat alle oneven genummerde aders van de bandkabel die floppy-disk drive en CRT/FDC-print verbindt, aan massa liggen. Dit is gedaan ter voorkoming van storing tussen de signalen onderling. Om diezelfde reden is het ook af te raden de bandkabel onnodig lang te maken.
Uit figuur 10 blijkt dat de Elektuur Flex-computer EC-68 met maximaal 4 floppy-disk-drives kan werken (DSØ t/m DS3 betekent drive select 0 t/m 3).
Bij het op de juiste manier inklemmen van de bandkabel in de bandkabel-konnektor (zit ader nr. 1 ook op pen 1 van de bandkabel-konnektor?) kan er met de diverse verbindingen eigenlijk niets meer misgaan, omdat de door ons aangegeven Molex-konnektoren allemaal polarisatienokken hebben en dus niet fout in het "male" konnektorgedeelte kunnen worden gestoken. Het is trouwens raadzaam voor het aansluiten van het videosignaal geen bandkabel te gebruiken, maar koax. De aansluitgegevens van de FDC-konnektor en de video-konnektor zijn gegeven in de figuren 11 en 12.
Ook de tweede print kan nu van onderdelen voorzien worden. De komponentenopstelling is gegeven in figuur 13. Hiermee is de bouw van de hardware beëindigd. Reden genoeg om over te gaan op de beschrijving van de software voor het systeem zelf: de firmware.

De firmware

Zoals reeds eerder geschreven, kan hardware niet werken zonder software (omgekeerd trouwens ook niet!). De software voor de EC-68 kan men kwa plaatsing onderverdelen in een tweetal groepen: de software op de printen in de vorm van EPROM's en de software op floppy-disks. Typisch voor het operating system Flex is, dat nagenoeg alle software op floppy-disk staat (dit in tegenstelling tot de vele home-computers die diverse EPROM's bevatten met BASIC-interpreters, tekstverwerkers, etc.). Bij Flex is het aantal EPROM's tot het noodzakelijke minimum beperkt; de rest staat op schijf. Hierin ligt de kracht van Flex. De software is zeer flexibel, omdat ze zeer makkelijk kan worden aangepast en veranderd, en er wordt niet meer ruimte dan nodig van het totale geheugenblok in beslag genomen door EPROM's. We geven graag toe dat hierdoor het systeem trager werkt dan een systeem met veel EPROM's aan boord. De access tot een EPROM is immers stukken sneller dan de access tot een floppy-disk-drive. In het

10



verhaal over firmware gaan we het uitsluitend hebben over de software in de EPROM's van de EC-68. De software op schijf komt nog ruim voldoende aan de orde in de diverse andere verhalen over Flex. Bij de aanschaf van de boot-diskette ontvangt u een manual waarin het operating systeem Flex zeer grondig uit de doeken wordt gedaan en waarin ook de diverse submodules, waaruit Flex is opgebouwd, uitgebreid worden verklaard.

Op de twee eurokaarten bevinden zich een drietal EPROM's, twee stuks 2716 en een 2732. De 2732 is vrij snel afgehandeld, omdat in dit IC eigenlijk geen programma zit. Het is de karaktergenerator, die deel uitmaakt van de video-elektronica. Op de door de firma's Schnijder Microsystems en BSC geleverde floppy disk bevindt zich een tekst-file met de naam CGTXT. Dit is een file die aangeeft hoe de karaktergenerator is opgebouwd. Met het kommando LIST CGTXT kan de file bekeken worden. Dit ter informatie voor degenen die vinden dat het nodig is de presentatie van bepaalde karakters op het beeldscherm te veranderen. Er blijven vervolgens nog een tweetal EPROM's over. Daarin moeten dus wel de programma's zitten. Dit zijn Bootrom en Assist-09. We zullen de inhoud van de EPROM's vluchtig doornemen en wel in dezelfde volgorde als ze door de microprocessor gebruikt zullen worden.

Assist-09 is een monitorprogramma dat van origine is geschreven door de firma Motorola. Inmiddels is Assist-09 vrij voor gebruik door iedereen. Dat maakt het ontwikkelen van een monitorprogramma een stuk makkelijker. De ontwerper van de EC-68 is ook uitgegaan van Assist-09 en heeft daarin een paar wijzigingen aangebracht om het programma aan te passen aan de EC-68. Bij het inschakelen van de EC-68 begint de 6809 met programma-executie vanaf adres FFFE (dat is de koude-start-vektor die in elke 68090-chip vast is ingebakken). De 6809 springt naar het begin van de EPROM op F800 (die namelijk de bovenste twee KByte van de memorymap bezet).

Het belangrijkste doel van de Assist-09-monitor is het "booten" van het operating system. Na het inschakelen van de EC-68 verschijnt op het beeldscherm de tekst ASSIST 09, gevolgd door de monitor-prompt (voorgesteld door het teken >). Door achter dit teken de letter F (goed geraden, de F van Flex) te tikken,

11

Gnd	o - 1	2 - o	
Gnd	o - 3	4 - o	
Gnd	o - 5	6 - o	DS3
Gnd	o - 7	8 - o	Index
Gnd	o - 9	10 - o	DS0
Gnd	o - 11	12 - o	DS1
Gnd	o - 13	14 - o	DS2
Gnd	o - 15	16 - o	Motor On
Gnd	o - 17	18 - o	Direction
Gnd	o - 19	20 - o	Steppulse
Gnd	o - 21	22 - o	Write Data
Gnd	o - 23	24 - o	Write Gate
Gnd	o - 25	26 - o	Track 00
Gnd	o - 27	28 - o	Write Protect
Gnd	o - 29	30 - o	Read Data
Gnd	o - 31	32 - o	SS
Gnd	o - 33	34 - o	

12

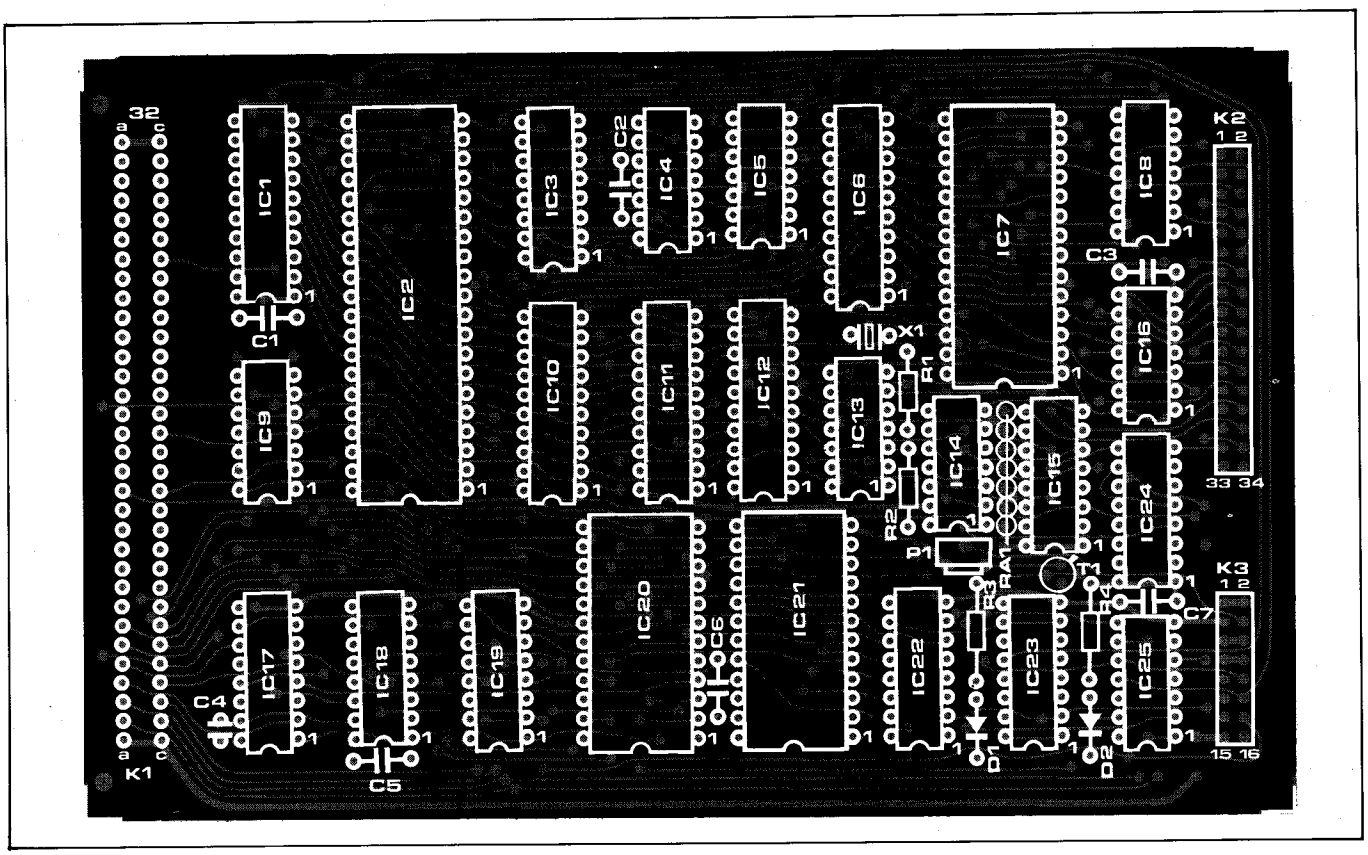
Gnd	o - 1	2 - o	Gnd
Gnd	o - 3	4 - o	CV
Gnd	o - 5	6 - o	Gnd
Gnd	o - 7	8 - o	Gnd
Gnd	o - 9	10 - o	VI
Gnd	o - 11	12 - o	HS
Gnd	o - 13	14 - o	VS
Gnd	o - 15	16 - o	+ 5 volt

Figuur 11. De floppy-interface houdt zich aan de wijd verbreide Shugart-standaard. Daardoor is het bijzonder eenvoudig om gangbare floppy-drives aan de EC-68 te koppelen. Nog een belangrijke opmerking: de toegepaste drives moeten beslist dubbelzijdige typen zijn (40 of 80 tracks).

Figuur 12. De konnektor voor de monitor. Gewoonlijk gebruikt men hiervan alleen het composite-video-signaal en de massa-aansluiting. Via een 75-Ω-koax-kabel wordt het signaal dan naar de monitor gevoerd. Op de konnektor staan verder nog het geïnverteerde video-signaal en de twee synchronisatiesignalen.

Figuur 13. De komponentenopstelling voor de FDC/CRT-print. "Slechts" 25 IC's zitten op deze tweede kaart.

Tabel 1. De belangrijkste systeem-adressen en de geheugen-indeling van de EC-68.



Onderdelenlijst CRT/FDC-kaart

Weerstanden (allemaal 1/8 W):
R1,R2 = 560 Ω
R3 = 1 k
R4 = 2k7
Ra1 = 7 × 220 Ω (SIL-array of 7 aparte weerstanden)

Kondensatoren:
C1...C7 = 100 n ker.

Halfgeleiders:
D1,D2 = 1N4148
T1 = BFY 90
IC1,IC10 = 74LS245
IC2 = 6845
IC3 = 74LS139
IC4 = 74LS32
IC5 = 74LS00
IC6,IC12 = 74LS377
IC7 = 1770 (Western Digital)
IC8,IC16 = 74LS06
IC9 = 74LS138
IC11 = 74LS244
IC13 = 74LS04
IC14 = 74LS74
IC15 = 4050
IC17...IC19 = 74LS157
IC20 = 6116
IC21 = 2732 (ESS 542)
IC22 = 74LS165
IC23 = 74LS163
IC24 = 74LS175
IC25 = 74LS02

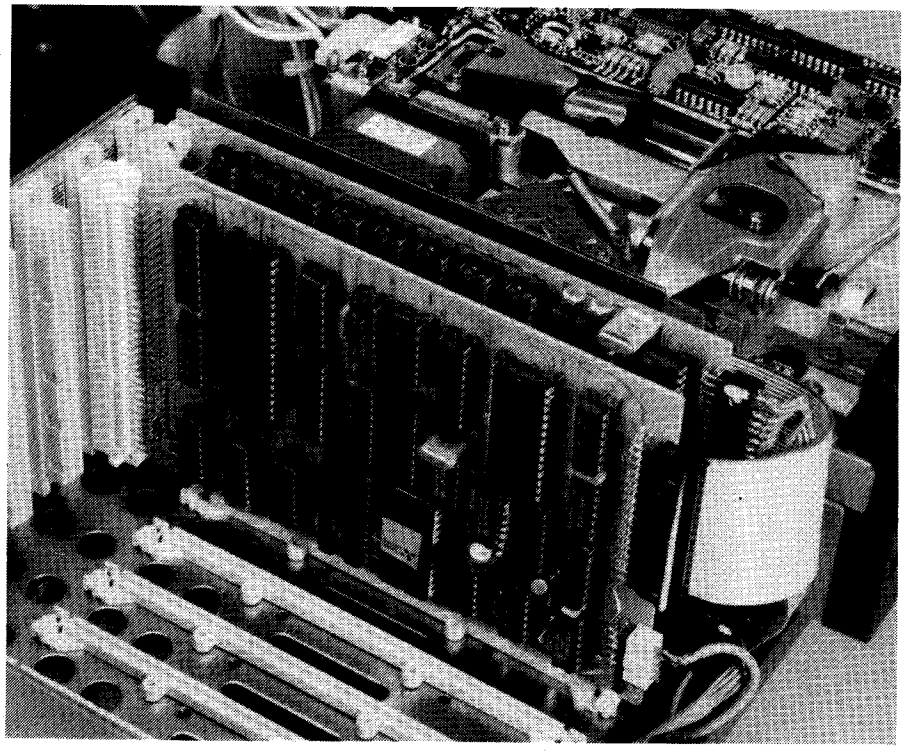
Diversen:
X1 = kristal 16 MHz (klein model)
K1 = 64-polige buskonnektor, male, volgens DIN 41612
K2 = 34-polige male konnektor en bijbehorende flat-cable-steker, bijv. Molex nr. 5342-34-GS1 en 5320-34-BGS1
K3 = 16-polige male konnektor en bijbehorende flat-cable-konnektor, bijv. Molex nr. 5342-16-GS1 en 5320-16-BGS1

Tabel 1.

:F800...FFFF:Assist-09 (EPROM);	:EC06:Floppy Disk Controller (sektor-register);	:CC00...D3FF:DOS;
:F000...F7FF:Bootrom (EPROM);	:EC05:Floppy Disk Controller (track-register);	:C980...CBFF:System Files Area;
:E800...EFFF:I/O;	:EC04:Floppy Disk Controller (status- en kommandoregister);	:C840...C97F:System FCB (File Control Block);
:ED00...EFFF:User-I/O;	:EC00:CRTC;	:C700...C83F:Scheduler & Printer Spooler (niet geïmplementeerd);
:EF00...EFFF:ACIA, VIA;	:E800...EBFF:User-I/O;	:C100...C6FF:Utility Command Area;
:EF80:VIA (parallele I/O);	:E000...E7FF:Videomemory;	C080...C0FF:Input Buffer;
:EF60:ACIA (seriële I/O);	:C000...DFFF:FLEX;	C000...C07F:Stack Area (SP wordt op C07F geïnitialiseerd);
:EC00...ECFF:DISK, VIDEO;	:DE00...DFFF:Disk Drivers;	:C000:Flex-operating-system-laadadres;
:EC07:Floppy Disk Controller (data-register);	:D400...DDFF:FMS (File Management System);	:0000...BFFF:User-RAM;

gevolgd door een carriage return, zal Assist-09 routines uit Bootrom oproepen en wordt het operating system Flex in het geheugen van de computer geladen. Basisfunkties in Assist-09 zijn onder andere een kommando waarmee men de inhoud van de verschillende registers kan bekijken en modificeren (het zogenaamde "Registers"-command), een kommando waarmee men de inhoud van een geheugenblok kan bekijken en veranderen (de "Memory"- en "Display"-kommando's). Deze en nog diverse andere kommando's liggen op machinetaalnivo. Dit soort kommando's kan ook worden

uitgevoerd door Flex-utilities op schijf. Het zou hier te ver voeren de diverse routines van Assist-09 en Bootrom die hiervoor nodig zijn uit te spitten. Voor degenen die alles over Assist-09 en Bootrom willen weten, verwijzen we weer naar de bootdisk, deze keer naar de tekst-files ASSIST09.TXT en BOOTROM.TXT.
Assist-09 bevat een aantal hulpprogramma's voor manipulaties op machinenivo. Omdat bij het kommuniceren met de computer een aantal machine-specifieke (lees: hardware-afhankelijke) programma's dienen te worden uitgevoerd, is er nog gebruik gemaakt van een tweede EPROM: Bootrom. Hierin staan de video-, floppy- en I/O-drivers die specifiek zijn voor de EC-68. Specifiek, omdat ze gebruik maken van informaties en kommando's die ze op bepaalde adressen wegschrijven of inlezen. Uit de vijf schema's heeft de oplettende lezer al een aantal adressen kunnen opmaken, omdat ze bij de diverse IC's (waarop iets dergelijks van toepassing is) vermeld zijn. Zo wordt keyboard-input bijvoorbeeld ingelezen via de PIA op adres $EF80, seriële input en output gaat via de ACIA op adres $EF60, de floppy-disk-controller zit op adres $EC04, etc.
De diverse drivers zijn uitgebreid beschreven in de diverse op schijf staande files. Als u echt tot op bitnivo wilt nagaan wat er gebeurt bij het opstarten van de EC-68, moet u de diverse .TXT-files maar eens listen.
Tot slot vindt u een zeer globale memory-map (met de interessantere adressen) in tabel 1.



Verkrijgbaarheid

Hieronder volgt een lijstje van firma's die moeilijk verkrijgbare delen voor de Flex-computer leveren.
*De boot-diskette met het operating system Flex en de diverse info-files is verkrijgbaar bij:

Schnijder Microsystems
Schooteindseweg 8a
5756 BD Vlierden
tel. 04930-13666
en
BSC B.V.
Lochemseweg 28
7244 RS Barchem
tel. 05734-570

*De floppy-disk-controller WD1770 wordt geïmporteerd door:
B.V. Diode
Hollantlaan 22
3526 AM Utrecht
tel. 030-884214

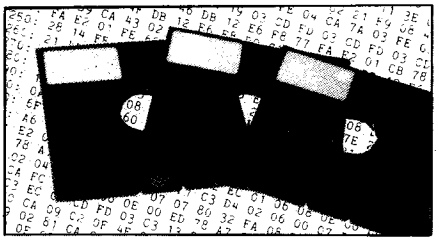
*De Molex-konnektoren worden geleverd door:
Molex B.V.
Visserstraat 13
5612 BS Eindhoven
tel. 040-450565

We danken de HCC-68XX-gebruikersgroep België voor het ter beschikking stellen van de video-driver voor dit Flex-systeem.

Binnenkort in Elektuur Computing nummer 4:

- de EC-65K — een nieuwe zelfbouwcomputer met 65816-CPU op 4 MHz
- Flex-software-overzicht en diagnostics
- Pascal-tekstverwerker
- Astrologie-programma
- Harddisk voor IBM-PC's

SOFTWARE



de EC-68K — deel 2

Systeembeschrijving en handleiding

U wilde een computer die tot nu toe alleen in uw dromen op uw werktafel stond? En software-pakketten gebruiken die meer dan konkurrerend zijn met de op dit moment op de markt verkrijgbare pakketten? Dan is de EC-68K voor u een goede partner. Wilt u bovendien de wereld van de 16/32-bit-multiprocessor-computers betreden, omdat u ook op de hoogte wilt blijven van de nieuwste ontwikkelingen? Dan raden wij u alleen nog aan de EC-68K zo snel mogelijk te bouwen en ermee aan de slag te gaan. Deze computer werd in samenwerking met de Duitse firma "Dr. Böhm" ontwikkeld, een firma die met name op het gebied van elektronische orgels erg aktief is. Een gebied dat, zoals bekend verondersteld mag worden, al lang niet (alleen) meer beheerst wordt door analoge elektronica. De historische achtergrond van onze kompagnon in deze garandeert bovendien een harmonische samenwerking tussen de twee processors, een 68008 en een 6809, die in deze machine huizen. Met dit paar wordt de EC-68K wat snelheid betreft een echte formule-I-computer. En dat niet alleen! Om ervoor te zorgen dat in assembler net zo komfortabel als in een hogere programmeertaal gewerkt kan worden, zijn floating-point-rekenoperaties en IN- en OUTPUT-procedures als extra mnemonics ingebouwd. Het operating system is niet voor de computer, maar voor de gebruiker van deze moderne 16/32-bits machine geschreven.

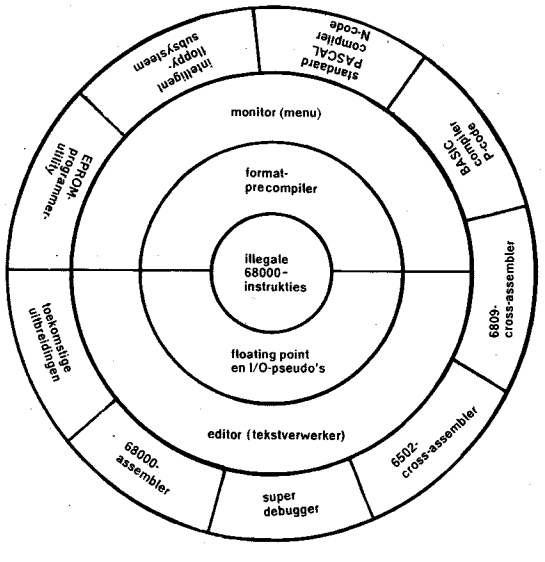
Onder de motorkap

De EC-68k is een uiterst nabouwzekere 16/32-bit computer die met twee processors werkt: een 68008 en een 6809. De 68008 is een processor die wat instruktieset betreft overeenkomt me de bekende 68000, maar hij heeft in tegenstelling tot zijn "grote" broer een 8-bits brede databus. U hoeft zeker niet bevreesd te zijn dat dit smalle datapad tot een traag geheel leidt. Uitvoerige tests in het lab hebben bewezen dat een 8-MHz-68000-computer met een 16 bits brede databus en dynamische RAM's trager was in het afwikkelen van zijn programmatuur dan onze EC-68K op 10 MHz en uitgerust met statische RAM's. Zoals bekend moet bij een dynamische RAM om de 15 mikrosekonden de geheugeninhoud opgefrist worden, een handeling die enkele klokcicly vraagt. Bij de EC-68K worden minder eisende statische RAM's gebruikt, die de processor verder geen tijd meer uit handen nemen.

Als u de EC-68K inschakelt, dan komt er meteen een menu op het scherm. Vanuit dit menu kunt de de full screen editor, de 68000-assembler en de debugger direkt oproepen. Wie in een hogere programmeertaal wil werken, kan vanuit het menu de BASIC-compiler starten. Binnenkort zal ook een Pascal-compiler beschikbaar komen. BASIC heeft een P-code-compiler, terwijl Pascal met een echte source-code-compiler werkt. Wie ooit met Microsoft-BASIC gewerkt heeft, zal met de BASIC op deze machine zo uit de voeten kunnen. Het spreekt vanzelf dat de BASIC op de EC-68K in overeenstemming is met de nieuwste software-standaarden. Ook cross-assemblers voor de 6502 en de 6809 zijn op de EC-68K beschikbaar; beide werden er overigens ook op ontwikkeld. Met de tekstverwerker kunt u bijvoorbeeld een 6502- of een 6809-assembler-programma ingeven. De cross-assembler produceert vervolgens voor de desbetreffende processor de benodigde machinekode en slaat deze ergens in het geheugen op. Via de seriële I/O-kaart kan de machinekode naar een aangesloten EPROM-programmer verzonden worden en aldus in EPROM (van 2716 tot en met 27256) vastgelegd worden.
De 68000-source-code voor de drie assemblers, de editor, de debugger de EPROM-programmer en het disk operating system is op diskette leverbaar. Ook de beide compilers zijn in 68000-assembler geschreven.
Een diskette met de software voor de compilers zal pas over enkele maanden verkrijgbaar zijn, omdat nog wordt gewerkt aan de dokumentatie van dit gedeelte.
Figuur 1 toont de overzichtelijke opbouw van het operating system van de EC-68K.

Figuur 1. Zo eenvoudig en overzichtelijk is het operating system van de EC-68K opgebouwd. Dit programma is gehuisvest in tien 2764-EPROM's. Met enige trots willen we toch even vermelden dat dit operating system geheel in overeenstemming is met de huidige stand der techniek.

1



De niet-geïmplementeerde 68000-instrukties vormen de kern van het systeem, het centrum van de cirkel. Door deze truuk kan men op assemblernivo met floating-point-getallen, goniometrische funkties en IN- en OUTPUT-procedures werken. Met andere woorden: het assemblerprogrammeren op de EC-68K gaat met het komfort van een hogere programmeertaal terwijl de efficiëntie gewaarborgd blijft, zodat ook tijdkritische zaken met sukses geprogrammeerd kunnen worden.
Om floating-point-getallen, integers, numerieke uitdrukkingen en tekst geformatteerd naar buiten te kunnen geven, is in een format-precompiler voorzien, de tweede ring in het schema van het operating system. Deze precompiler is een ongeveer 8 Kbyte groot pakket en laat niets te wensen over. In het artikel "de 68000 kort en bondig (deel 2)", elders in dit blad, zijn voorbeelden opgenomen hoe met deze precompiler komplexe toestanden eenvoudig kunnen worden opgelost.
De monitor en de tekstverwerker (editor) vormen de derde ring van het operating system. De monitor meldt zich na de (inschakel)reset met een menu. Vanuit dit menu kan bijvoorbeeld de editor of andere systeem-software aangeroepen worden. Het eigenlijke centrum der bezigheden is echter de editor. Van daar uit kunnen zowel het DOS (disk operating system), de assembler of de hogere programmeertalen aangeroepen worden. Omdat de editor de cursor-besturing voor zijn rekening neemt, zijn de kommando's hiervoor altijd hetzelfde, of men nu met BASIC, de debugger of een ander programma werkt.
In de buitenste ring zien we de hogere programmeertalen en andere utility-programma's. Bedenk dat het geheel rond de 80 K programmatuur, gezeteld in tien 2764-EPROM's, omvat. Na het inschakelen van de machine is het niet nodig een systeemprogramma vanaf disk te laden, dit zit reeds gebruiksklaar onder de knoppen. Dat dit ten nadele gaat van de grootte van het nog vrij te adresseren geheugen, is slechts zeer betrekkelijk; de 68000 is in staat 1 Mbyte direkt te adresseren. Zonder problemen kunnen dus meerdere programma's gebruiksklaar in het geheugen worden gehouden. Om ook het operating system daaronder te laten vallen, lijkt ons — voor een dergelijk dagelijks gebruiksgoed — nauwelijks een luxe. Een ander voordeel van een dergelijk groot adresseringsbereik: een enkele drive is voor de EC-68K eigenlijk voldoende.

De editor

Om teksten (dat kunnen natuurlijk ook programma's zijn) komfortabel in te kunnen voeren, is een krachtige (zoals dat in computerjargon nu eenmaal heet) tekstverwerker aanwezig. Met de cursor kunt u zich naar een willekeurige plaats in de tekst begeven om wijzigingen aan te brengen of iets tussen te voegen danwel te wissen. U kunt een regel in meerdere regels omvormen of blokken tekst markeren. Uiteraard zijn ook blokken tekst te kopiëren (copy) of te verplaatsen (move).

Tabel 1:
Editor-stuurkommando's

teken	hex	mnemonic	beschrijving
↑A	$01	TAM	plaats Tekst-"A"-Markering
↑B	$02	TBM	plaats Tekst-"B"-Markering
↑C	$03	STOP	proces-onderbreking
↑D	$04	SU	Scroll Up, schuif beeld naar boven
↑E	$05	SD	Scroll Down, schuif beeld naar beneden
↑F	$06	NXT	NEXT, ga naar de volgende, te gebruiken bij "FIND/REPLACE"
↑G	$07	IL	Insert Line, voeg een regel tussen
↑H	$08	BS	Back Space, cursor een plaats naar links, teken onder de cursor wordt gewist
↑I	$09	BSM	Block Start Markering plaatsen
↑J	$0A	BEM	Block Eind Markering plaatsen
↑K	$0B	KL	Kill Line, regel waar de cursor op staat wissen
↑L	$0C	KBM	Kill Block Markers, beide blokmarkeringen verwijderen
↑M	$0D	CR	Carriage Return
↑N	$0E	IS	Insert Space, spatie tussenvoegen en regel opschuiven
↑O	$0F	DC	Delete Character op plaats van cursor
↑P	$10		niet gebruikt
↑Q	$11	CL	Cursor Left
↑R	$12	CR	Cursor Right
↑S	$13	CU	Cursor Up
↑T	$14	CD	Cursor Down
↑U	$15	FU	Frame Up, 16 regels naar boven
↑V	$16	FD	Frame Down, 16 regels naar beneden
↑W	$17	CLI	Copy Lines, kopieer de voorstaande regels in de volgende regel, te gebruiken met "IL"
↑X	$18	EL	Erase Line, wis de regel waar de cursor op staat
↑Y	$19	RL	Restore Line, regel niet veranderen en oude toestand herstellen
↑Z	$1A	SPL	Split Line, de inhoud van één regel opdelen in twee regels

Tevens kunnen strings worden opgezocht. De editor-kommando's zijn in drie groepen te verdelen:
1) editor-stuurkommando's (tabel 1)
2) editor-makrokommando's (tabel 2)
3) editor-floppykommando's (tabel 3)
Met de kommando's I en J kan een tekstblok gemarkeerd worden. Dat is van belang als u het desbetreffende blok bijvoorbeeld ergens anders wilt kopiëren. Plaats de cursor op de plek in de tekst waar u het gemarkeerde blok wilt hebben. Druk vervolgens op <ESC> en geef het kommando <COPY>. Ook als al een flinke lap tekst in het geheugen zit, zult u verbaasd zijn over de enorme snelheid waarmee uw wens wordt uitgevoerd. Tabel 1 spreekt verder voor zich, het lijkt ons niet nodig daar een nadere toelichting op te geven.
De editor-makrokommando's worden in tabel 2 nader verklaard. Bij de meeste kommando's hoeven slechts de eerste twee of drie letters ingegeven te worden, dat maakt het typewerk aanmerkelijk korter.
Hoewel de tekstverwerker een full-screen-editor heeft, dus vrije toegang geeft tot het hele scherm, genereert hij automatisch regelnummers. Voor degene die representatieve korrespondentie wil verwerken, hoeft dit geen bezwaar te zijn;

Tabel 2:
Editor-makrokommando's (ingeleid met <esc>)

kommando	beschrijving
assemble	assembleer de source-code in het geheugen
copy	kopieer de tekst tussen de twee blokmarkeringen voor de regel waar de cursor op staat
debug	roep de debugger aan
delete	wis de tekst tussen de twee blokmarkeringen
exit	keer terug naar het menu van de monitor
find/string/	zoek de betreffende string
move	verplaats de tekst tussen de twee blokmarkeringen tot voor de regel waarop de cursor staat
number	voorzie de regels die nog geen nummer hebben van regelnummers
refresh	formatteer het beeldscherm en plaats de cursor in het midden
resequence	voorzie de file van nieuwe regelnummers
run	start het zojuist geassembleerde programma
size	geef de lengte van de file
switch n	schakel naar een ander geheugenblok adres = $28000 + n * $8000

Tabel 3:
Editor-floppy-kommando's (ingeleid met <esc>)

kommando	parameters	beschrijving
add	unit, naam	laad een file vanaf disk en voeg deze toe aan de file in het geheugen
compare	unit, naam	vergelijk de geheugeninhoud met een file op disk
csave	unit, naam	schrijf de file naar disk en verifieer de data op disk met de data in het geheugen
format	unit, naam, nummer	formatteer een nieuwe diskette
files	unit	geef de inhoudsopgave van een diskette
get	unit, naam	laad een file vanaf disk
pw	unit	schakel loopwerk in
po	unit	schakel loopwerk uit
remove	unit	wis een file
save	unit, naam	schrijf een file naar disk
test	unit	test loopwerk en diskette
title	naam	definieer de werktitel voor een file
version	unit	geef alle werktitels

de regelnummers kunnen eenvoudig weggelaten worden. Een demoprogramma hiervoor zal in de nabije toekomst gepubliceerd worden. Misschien klinkt het wat ongeloofwaardig, maar in het genereren van die regelnummers zit de meeste rekentijd van de processor. Toch laten we hem dat werk doen omdat bij lange programma's regelnummers erg handig kunnen zijn voor het lokaliseren van fouten.
Vaak worden tussen twee regels meerdere regels tekst tussengevoegd, en dan kan het handig zijn de regels van nieuwe nummers te voorzien. Als u zojuist een listing op papier heeft uitgedraaid, dan zal het duidelijk zijn dat de regelnummers op die listing na hernummeren niet meer korresponderen met de regelnummers op uw beeldscherm. Het kommando "NUMBER" zal daarom alleen met beleid nieuwe regelnummers uitdelen, bestaande regelnumers worden niet veranderd en mocht een nieuw blok tekst meer regels bevatten dan de negen die maximaal tussen twee bestaande regels ingepast kunnen worden, dan zullen er tekstblokjes gevormd worden waarin meerdere regels slechts één nummer krijgen toebedeeld. Op deze wijze hoeft u na elke hernummering niet een kompleet nieuwe listing af te laten drukken, de meeste regelnummers blijven immers onveranderd. Alleen het kommando "RESEQUENCE" zal de tekst wel van nieuwe regelnummers voorzien in het 10,20,30-stramien.
Een ander zinvol kommando is "SWITCH n" (n=1,2,3...). Hiermee wordt het geheugen van de EC-68K opgedeeld in segmenten van elk 32 Kbyte. Meerdere programma's of tekstblokken kunnen zo in het werkgeheugen gehouden worden zonder dat ze steeds vanaf diskette geladen hoeven te worden.

Editor-floppykommando's

De belangrijkste kommando's:
* get = laad een file vanaf disk
* save = schrijf een file naar disk
* remove = wis een file
* format = formatteer een nieuwe diskette
* files = geef een inhoudsopgave

In tabel 3 zijn alle editor-floppy-kommando's terug te vinden. Alvorens er een file naar disk geschreven kan worden, moet de nieuwe diskette geformatteerd worden. Dit gebeurt met het format-kommando. Bij dit kommando kunnen drie parameters toegevoegd worden:

1) unit-nummer (unit)
2) naam van de diskette (naam)
3) serienummer (nummer)

Als maar een enkele drive is aangesloten, dan is het unit-nummer 1. Worden op de in deze drive aanwezige diskette bijvoorbeeld assembler-programma's weggeschreven, dan kunnen we deze de naam assembler geven. Als serie- of diskettenummer moet een viercijferig getal gegeven worden, bijvoorbeeld 0004.
Met het kommando "GET FILE-NAAM" kan een file vanaf disk geladen worden. Let erop dat de punt achter de file-naam tot gevolg heeft dat het disk operating system weet dat de laatste versie van de desbetreffende file geladen moet worden. Wil men aansluitend de gemodificeerde file weer wegschrijven naar disk, dan hoeft die file niet met naam gespecificeerd te worden. Enkel het kommando "save" is voldoende. De computer weet de naam nog van het get-kommando en zal, aannemende dat de weg te schrijven file een nieuwe versie is, het versienummer automatisch verhogen. Aan dit versienummer kunt u, na bijvoorbeeld met "files" een inhoudsopgave gevraagd te hebben, uw meest aktuele file herkennen.
In tegenstelling tot het save-kommando zal het remove-kommando juist het oudste versie-nummer van een file verwijderen. Ook hier hoeft, als daarvoor een get- of save-kommando gebruikt is, geen naam gespecificeerd te worden. Het DOS gaat er van uit dat de bewerking dient te worden uitgevoerd op de file die u eerder al onder handen had. Is het get, save of remove echter van kracht op een nieuwe file, dan kunt u met "TITLE" een nieuwe naam opgeven.

68000-assembler-pseudo-instrukties

Wie de EC-68K-assembler onder de vin-

Tabel 1. De stuurkommando's van de editor.

Tabel 2. De macro-kommando's van de editor.

Tabel 3. De kommando's van het disk operating system.

gers heeft gehad, zal waarschijnlijk tot geen enkele andere assembler nog te bekeren zijn. De assembler en de debugger zijn zo krachtig dat, wil men toch in een hogere programmeertaal werken, het schrijven van een compiler niet veel problemen op zal leveren. Overigens zijn de BASIC- en standaard-Pascal-compiler ook op de EC-68K geschreven en getest.
Wat het programmeren op de EC-68K zo komfortabel maakt, is een slimme maar eigenlijk heel eenvoudige truuk.
Bij de 68000 zijn de opcodes 16 bits breed. Instrukties waarvan de opkode begint met een A of een F, dus $AXXX of $FXXX, zijn er niet. Dit zijn dus illegale instrukties. Stoot de 68000 op een dergelijke instruktie, dan zal hij via een fout-vektor het op dat moment lopende programma verlaten. Elk van de beide groepen illegale instrukties heeft zijn eigen vektor (1010-emulator en 1111-emulator). Ze zijn beide in ROM ($0...$3FF) aanwezig. Normaal gesproken zouden deze vektoren wijzen naar een programmaatje dat de gebruiker erop attendeert dat de processor een illegale instruktie is tegengekomen, maar wij doen het anders.
De 1010- en de 1111-emulator gebruiken wij om floating point- en IN- en OUTPUT-instrukties als nieuwe mnemonics te implementeren. Met andere woorden: via deze vektoren emuleert de processor nieuwe 68000-instrukties die we zelf gekreëerd hebben. Deze nieuwe instrukties zijn als afzonderlijke routines in mnemonics ingegeven, waarna de assembler voor ons de voor de processor uitvoerbare machinekode gegenereerd heeft. Met de op deze wijze in het leven geroepen extra instrukties laat de instruktieset van de EC-68K asembler zich onderverdelen in de volgende hoofdgroepen:

1) normale 68000-instrukties
2) normale 68000-pseudo-instrukties
3) floating-point-instrukties via 1111-emulator
4) in/output-instrukties via 1010-emulator
5) floating-point-formatteringsinstrukties

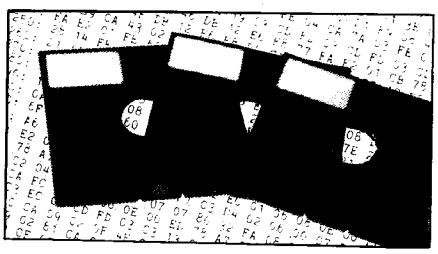
Welke voordelen hebben de uitgebreide 68000-pseudo-instrukties van de EC-68K nu ten opzichte van andere assemblers? We mogen aannemen dat u wel eens eerder met een assembler heeft gewerkt, en u kunt zich wellicht herinneren dat dat "werken" wel heel letterlijk geldt als bijvoorbeeld een karakter van het toetsenbord ontvangen moet worden of een getal geformatteerd naar het beeldscherm geschreven moet worden. Ergernis alom, omdat de source-code van het operating system van uw computer niet verkrijgbaar was, en dat u zich zonder verdere hulpmiddelen in dit detailprobleem moet storten. Heeft u de source-code wel, dan is het tenminste mogelijk om systeemspecifieke subroutines op te roepen. Het nadeel van het werken op deze manier is het worstelen met stapels papier om de startadressen van de desbetreffende routines te vinden.
Door te grijpen naar een hogere programmeertaal laten deze problemen zich omzeilen, maar dan loopt u vaak vast op het feit dat de BASIC of de PASCAL te traag blijkt te zijn om een tijdkritisch probleem adekwaat tot een goed einde te brengen. Met name bij het werken met grafische beelden met een hoge resolutie is een assembler (voor de snelheid) met goniometrische funkties (voor het gemak) eigenlijk onontbeerlijk. Het omrekenen van rechthoekige koördinaten naar pcol-koördinaten laat zien hoe belangrijk het is ook met floating-point-getallen in een assembler te kunnen werken. Het maken van een snel bewegend grafisch plaatje, zelfs in kleur, komt met deze assembler binnen het bereik van de programmeur. En elke lezer die zich vaak met analoog/digitaal-omzettingen bezig houdt, vindt in de "EC-68K"-assembler een ideale partner. Vele gangbare statements, zoals "INKEY", "PRINT" en "PRINT-USING" om er maar een paar te noemen, werden als instrukties in onze asssembler opgenomen, alleen onder een andere naam om het typewerk tot en minimum te beperken. Tabel vier laat alle 68000-pseudo-instrukties zien die op het

Tabel 4:
68000-assembler-pseudo-instrukties

pseudo-instruktie	beschrijving	bijzonderheden
bp 0...5	set "Break Point"	
code	genereer object-code	
date.x	lees datum uit real-time-klok	
dc.x	definieer een konstante	
disp	(display) stuur een string naar het beeldscherm	
dr.x	definieer een konstante relatief	
ds.x	definieer een geheugenruimte	
entry	definieer een programma-startadres	
fadd	floating-point-optelling	<ea2> = <ea2> + <ea1>
fsub	floating-point-aftrekking	<ea2> = <ea2> – <ea1>
fmul	floating-point-vermenigvuldiging	<ea2> = <ea2> × <ea1>
fdiv	floating-point-deling	<ea2> = <ea2> / <ea1>
float	"long integer"/floating-point-konversie	<ea2> = float <ea1>
fix	floating-point/"long integer"-konversie	<ea2> = fix <ea1>
fint	gebruik alleen het gedeelte voor de decimale punt van het floating-point-getal	<ea2> = int <ea1>
fmod	floating-point-modulo-deling	<ea2> = <ea2> mod <ea1>
fneg	floating-point-negatie	<ea2> = NOT <ea1>
ffrac	gebruik alleen het gedeelte achter de decimale punt	<ea2> = frac <ea1>
finput	floating-point-invoer vanaf toetsenbord	<ea2-waarde> = finput ea1-ASCII
fsin	floating-point-sinus	<ea2> = sin <ea1>
fcos	floating-point-cosinus	<ea2> = cos <ea1>
ftan	floating-point-tangens	<ea2> = tan <ea1>
fsqrt	floating-point-wortel	<ea2> = sqrt <ea1>
fexp	floating-point-machtsverheffing (grondgetal e)	<ea2> = exp <ea1>
fln	natuurlijke floating-point-logaritme	<ea2> = ln <ea1>
farctan	floating-point-arc-tangens	<ea2> = arctan <ea1>
fracsin	floating-point-arc-sinus	<ea2> = arcsin <ea1>
farccos	floating-point-arc-cosinus	<ea2> = arccos <ea1>
flog	floating-point-logaritme (grondgetal 10)	<ea2> = log <ea1>
fcmp	floating-point-vergelijking	Z = <ea2> – <ea1>

Tabel 4. De pseudo-kommando's van de assembler. Deze maken het samenstellen van een computerprogramma bijzonder eenvoudig. Als u eenmaal hebt gewerkt met de assembler van de EC-68K, zult u er snel achter komen dat dit niet veel moeilijker is dan het werken in een hogere programmeertaal.

Figuur 2. De opbouw van een floating-point-getal voor de assembler-pseudo-instrukties. Om ervoor te zorgen dat voor de floating-point-berekeningen dezelfde regels gelden als voor de overige 68000-instrukties, moesten de floating-point-getallen worden ondergebracht in de 32-bits-registers van de processor.



het operating system Flex

In dit nummer van Elektuur Computing presenteren we een 6809-systeem op twee eurokaarten: de EC-68. En daar hoort natuurlijk ook software bij. Wij hebben hier gekozen voor het in de V.S. wijd verbreide operating system Flex-9. Het software-aanbod voor Flex-9 is bijzonder groot. In dit artikel vertellen we wat meer over Flex-9, de software-basis van de EC-68.

Het operating system Flex is een diskette-gebaseerd systeem voor CPU's van het type 68xx (bijvoorbeeld 6800, 6802, 6809). Dit operating system is onder de computerhobbyisten nauwelijks verspreid. In de industrie is Flex vrij bekend. Momenteel wordt Flex daar echter verdrongen door zijn opvolger UNI-FLEX. De prijs van UNI-FLEX is echter dermate hoog dat een hobbyist zich dat niet kan permitteren. Ook voor kleine firma's vormt die prijs al een struikelblok. UNI-FLEX schijnt echter nog krachtiger te zijn dan UNIX. Maar nu terug naar Flex. Flex-9 is waarschijnlijk het sterkste operating system dat verkrijgbaar is voor de 6809. Het is in vele opzichten beter dan CP/M.

Wie zich intensief wil bezighouden met Flex, kunnen we het Amerikaanse tijdschrift "68 Micro Journal" aanbevelen. Wie geen moeite heeft met de Engelse computer-vaktaal, vindt hierin veel interessante hard- en software-informatie.

Zoals al een paar keer is opgemerkt, is er veel software leverbaar voor Flex. En mocht men in ons kleine kikkerlandje voor een bepaalde toepassing geen programma vinden, dan kan nog altijd een programma in de Verenigde Staten besteld worden. Want dat is het grote pluspuntr van Flex: elke bezitter van een Flex-systeemschijf kan *alle* Flex-programma's op zijn systeem laten draaien.

Het operating system Flex wordt op de markt gebracht door de Amerikaanse firma Technical Systems Consultants (TSC). Daarnaast leveren diverse software-huizen gebruikers-software voor Flex. Hieronder volgt een globaal overzicht van het Flex-operating-system, onderverdeeld in funktionele groepen. Dit overzicht is zeker niet kompleet, maar geeft een goede indruk van de mogelijkheden van het systeem.

De systeem-software

De diverse files, die gebruikt worden bij het "booten" (dat is een mooi Engels woord voor het opstarten van de computer), staan op de systeem-diskette.

Hier volgt een opsomming van een aantal files op deze schijf. We beginnen uiteraard met Flex zelf (op disk FLEX.SYS genaamd). Voor het gebruikersgemak is aan het DOS een systeem-file verbonden waarin de diverse foutmeldingen zijn opgenomen (filenaam : ERRORS.SYS).

Na het inlezen van het DOS zal Flex zoeken naar een command-file (herkenbaar aan het achtervoegsel .CMD) die EXEC (van execute) heet, EXEC.CMD. Flex voert namelijk de kommandoregel uit die staat in een tekst-file met de naam STARTUP.TXT. Meestal is dat: EXEC 0.START.TXT. Er moet dus ook een file START.TXT op disk staan. Deze tekst-file wordt door de gebruiker zelf geschreven met behulp van een tekstverwerker en bevat een aantal kommandoregels die achter elkaar worden uitgevoerd (EXEC 0.STARTUP.TXT betekent dus gewoon: voer een aantal kommando's uit in de volgorde zoals deze zijn opgesomd in de tekst-file STARTUP.TXT). Figuur 1 geeft een typisch voorbeeld van de mogelijke inhoud van zo'n STARTUP-file. Elk kommando neemt een regel in beslag, omdat Flex elke regel als één kommando beschouwt. Een uitleg van de funktie van de verschillende kommandoregels is wel

1

De door de gebruiker ingegeven tekst is onderstreept.

```
+++ LIST STARTUP.TXT
0.VERIFY IS ON
0.HECHO 0C
0.HECHO 1B 2E 33
0.HECHO 0D 0A
0.DATE
0.HECHO 0D 0A
0.ECHO Good day FLEX software at your command
0.ECHO ### E L E K T O R .6809 Flex computer ###
0.ECHO My system information is:
0.ECHO [1] Memory capacity is 64 KBytes
0.ECHO [2] Microprocessor is 6809
0.ECHO [3] Disc format according to FD 1771
0.ECHO [4] If drive is 40 tracks and we have double side
0.ECHO     and double density with 18 sectors per track.
0.ECHO     Then we have a disc capacity of 1404 sectors
0.ECHO     of 256 bytes each. Thus we have 376 KBytes of
0.ECHO     disc storage.
0.HECHO 0D 0A
0.HECHO 0D 0A
0.ASN W=0 S=0
0.TTYSET WD=80 DP=25 PS=Y
0.TTYSET TB=5C EJ=00 ES=20 BE=08
0.LIST 0.START1.TXT
+++
```

Figuur 1. Zo kan de inhoud van een STARTUP.TXT-file er bijvoorbeeld uit zien.

fpinit	formatteringsbuffer openen	fpinit <ea-formule>, ak
fpreal	gebroken getal naar formatteringsbuffer sturen	fpreal <ea-REAL>, ak
fphex	hex-getal naar formatteringsbuffer sturen	fphex <ea-HEX>, ak
fpint	geheel getal naar formatteringsbuffer sturen	fpint <ea-INTEGER>, ak
fpalfa	string naar formatteringsbuffer sturen	
fpend	formatteringsbuffer sluiten	fpend ak
fpchar	een teken naar formatteringsbuffer sturen	fpchar <ea-CHAR>, ak
chkfmt	floating-point-formaat testen	chkfmt <ea1>, <ea2>
floppy	naar floppy-subsysteem een kommando sturen	floppy <ea-buffer>
inchr.x	lees een teken van het toetsenbord	inchr.x <ea>
inhex.x	lees hexadecimaal van het toetsenbord	inhex.x <ea>
input	lees een string vanaf het toetsenbord in een buffer	input <ea>
list	stuur een assembler-instruktie naar het beeldscherm	
msg	zet een foutkode om in karakters	
nocode	genereer geen object-code	
nolist	stuur geen listing naar het beeldscherm	
offset	definieer een offset voor de object-code	
org	definieer een begin van een programma	
outchr.x	stuur een karakter naar het beeldscherm	outchr.x <ea>
outhex.x	hex-ASCII-konversie naar het beeldscherm	outhex.x <ea>
page	nieuwe pagina	
print	stuur een string naar de printer	print <ea>
printer	stuur een listing naar de printer	
run	start een programma	
setdate.x	stel datum in voor real-time-klok	setdate.x <ea>
settime.x	stel de tijd in voor de real-time-klik	settime.x <ea>
time.x	lees tijd uit real-time-klok	time.x <ea>

tekenverklaring bij tabel 4

.x	.b, .w, .l
ak	een adresregister als format-pointer
<ea1>	effektief bron-adres
<ea2>	effektief bestemmingsadres
formule	zie hieronder

voorbeeld van een formule:

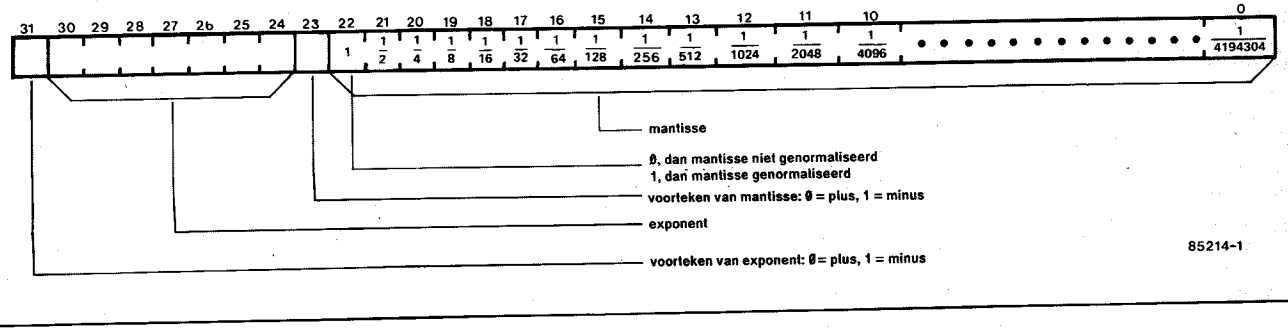
fm: format cl'decimaal ='xs4d.2dx,6a'..'cl

fm:	label
c	carriage return
l	line feed
a	plaats voor alfanumerieke tekens
d	plaats voor numerieke tekens (digits)
s	plaats voor plus of min (sign)
h	plaats voor hexadecimaal teken
x	plaats voor spaties
z	nulonderdrukking (leading zero suppression)
e	plaats voor exponent
'...'	plaats voor tekst
/	scheidingsteken voor volgende operanden
()	prioriteitshaken
0...9	herhalingsteller (repeat counter)

moment van dit schrijven op de EC-68K geïmplementeerd zijn. In het al eerder gememoreerde artikel "de 68000 kort en bondig" maken we ons verder vertrouwd met deze instrukties. Een aantal praktische programmavoorbeelden laten zien hoe gemakkelijk deze assembler werkt.
De meeste floating-point-instrukties zijn net zoals de meeste andere 68000-instrukties opgebouwd. Daarbij staat "ea" voor effektief adres:

mnemonic <ea1>,<ea2>
Een voorbeeld: geeft u de instruktie
fadd d0,d1
dan wordt het floating-point-getal van dataregister 0 opgeteld bij het floating-point-getal in register d1. Het resultaat van deze optelling staat in d1.
Een ander voorbeeld toont hoe groot de overeenkomst is tussen de 68000-pseudo-instrukties en de normale 68000-instrukties:
divs d0,d1 en
fdiv d2,d3
De eerste instruktie, een normale 68000-instruktie, levert een deling op tussen twee gehele getallen (integers). Het getal in d1 wordt gedeeld door het getal in d0. Het resultaat van de deling staat in d1 op bit 15...0. Bit 31...16 bevatten de rest van de deling. Op soortgelijke wijze verloopt de deling tussen twee floating-point-getallen. Het getal in d3 wordt gedeeld door het getal in d2, waarbij het resultaat, ook weer een floating-point-getal,in d3 terecht komt. Hoe de opbouw van een floating-point-getal is, laat figuur 2 zien.
Ook de goniometrische funkties hebben de gebruikelijke schrijfwijze in het effektieve adresveld:
fsin d0,d1.
De EC-68K berekent in dit voorbeeld de sinus van een bepaalde hoek. De hoek, in graden, bevindt zich als een floating-point-getal in dataregister d0. De berekende sinus komt als floating-point-getal te staan in d1. Het verhaal voor de exponentiële funkties is eigenlijk analoog:
fexp d4,d5.
De exponent van de te berekenen emacht vindt de processor in d4. De EC-68K berekent voor u ed 4 (waarbij e het grondtal is van de natuurlijke logaritme) en zet het resultaat in d5. Verdere details over de pseudo-instrukties zijn te vinden in tabel 5 en 6. Zoals gezegd is de opbouw van een floating-point-getal te vinden in figuur 2 en hoe u daarmee moet werken, wordt getoond in de demoprogramma's in het artikel "de 68000 kort en bondig" (deel 2)

1



op zijn plaats. Als eerste staat genoemd: VERIFY ON. Daarna volgt:

VERIFY.CMD. Dit is een command-file die verifieert of een file goed op schijf is weggeschreven, door de file na het wegschrijven weer terug te lezen en te vergelijken met de originele file. Het is prettig te weten dat deze kontrole wordt uitgevoerd, maar het kost wel een beetje tijd. Om die reden laten sommige gebruikers de lijn 0.VERIFY ON weg uit de startup-file.

HECHO.CMD is een file die hex-strings naar de terminal toe stuurt. Zo betekent bijvoorbeeld HECHO ØC: stuur het ASCII-teken ØC naar de terminal. Dit heeft tot gevolg dat het beeldscherm gewist wordt (0C betekent immers: Clear Screen). HECHO 0D 0A betekent: Carriage Return + Line Feed.

DATE.CMD drukt de bij het opbooten ingetypte datum op het beeldscherm af. DATE.CMD kan ook worden gebruikt om de eerder ingetikte datum te veranderen. Dit kommando is echter sterk afhankelijk van de hardware van de gebruikte computer In veel gevallen moet de file DATA.TXT dan ook gemodificeerd en nieuw geassembleerd worden.

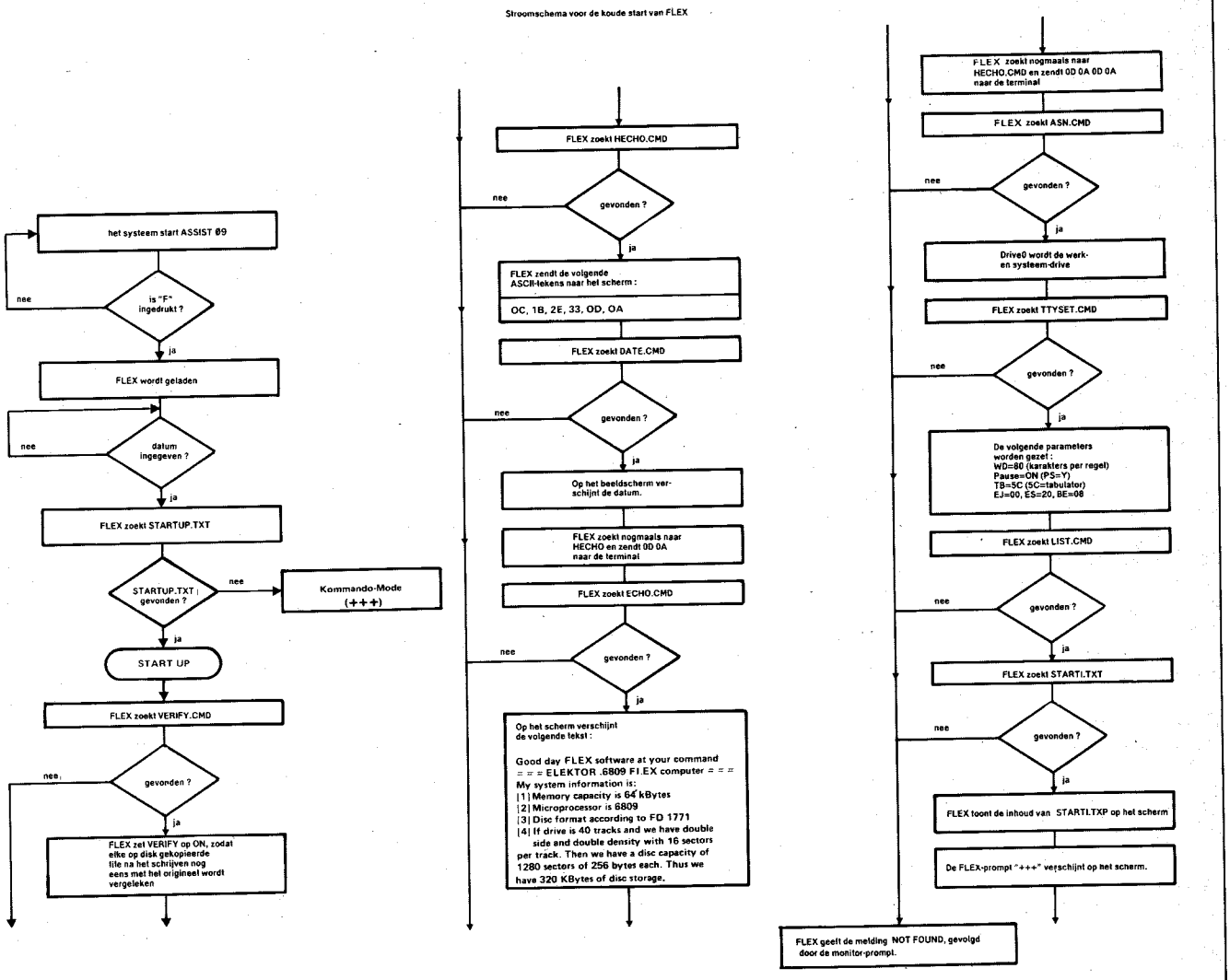
ECHO.CMD is een variant van HECHO.CMD. ECHO echoot ASCII-tekens naar het beeldscherm. Een handige manier om diverse meldingen of systeem-informatie mee te delen. Men kan zoveel ECHO-regels opnemen als men wil, als aan het begin van elke regel maar ØECHO gezet wordt.

ASN.CMD (afkorting van assign) is de volgende command-file die we tegenkomen. Assign wordt gebruikt door het DOS voor het toekennen van work-drive en system-drive. "Arme" computergebruikers met een enkele disk-drive zullen in deze regel schrijven: ØASN W=Ø S=Ø, wat tot gevolg heeft dat drive Ø zowel work- als system-drive is.

TTYSET.CMD is een heel belangrijke file, omdat deze dient te worden gebruikt voor het initialiseren van een aantal terminal-parameters. Met terminal-parameters wordt onder andere bedoeld: het aantal karakters per regel (WD = 80: 80 tekens per regel), het aantal regels per beeldscherm (DP = 25: 25 regels per beeldscherm), welk ASCII-teken gebruikt wordt voor de TAB (tabulator); in ons voorbeeld 5C.

LIST.CMD sluit de rij. De naam spreekt voor zich. List geeft de inhoud van de daarna genoemde file op het beeldscherm. ØLIST ØSTART1.TXT betekent dus: druk de inhoud van de tekst-file genaamd START1.TXT op het beeldscherm af. Omdat het mogelijk is vanuit de START.TXT-file naar bijvoorbeeld een tekstverwerker te gaan (de regel ØSCR DEMO.TXT is daarvoor voldoende) komt de START1.TXT goed van pas om de

2



3

```
+++ newdisk

ARE YOU SHURE? Y

SCRATCH DISK IN DRIVE 0? Y

80 TRACKS (N=40 TRACKS)? Y

DOUBLE SIDED DISK? Y

DOUBLE DENSITY DISK? Y

18 SECTORS (N=16 SECTORS)? Y

VOLUME NAME? ELEKTOR

VOLUME NUMBER? 1


FORMATTING COMPLETE

TOTAL SECTORS = 2844

+++
```

gebruiker enige huishoudelijke informatie te geven over het programma waarmee Flex opstart (bijvoorbeeld enkele beschrijvingen van de belangrijkste kommando's). Op een andere disk, die gebruikt wordt voor het testen en ontwikkelen van BASIC-programma's, staat de file START2.TXT waarin de gebruiker op de hoogte wordt gebracht van het feit dat er na het inladen van bijvoorbeeld Extended BASIC (0XBASIC) onmiddellijk kan worden begonnen met het intikken van BASIC-programmaregels.

Het spreekt voor zich dat de diverse in START.TXT genoemde kommando-files ook op disk aanwezig moeten zijn (anders reageert Flex met "NOT FOUND"). Een file die even belangrijk is als FLEX.SYS is de file LINK.CMD. Hij is nodig voor het verbinden (linken) van Flex met de bootstrap-loader (syntax: LINK FLEX <CR>). Eindigen we het stuk systeemsoftware met het bespreken van een aantal files die voor het fatsoenlijk werken met Flex eigenlijk onontbeerlijk zijn.

P.CMD en PRINT.SYS worden gebruikt wanneer de output niet naar het beeldscherm, maar naar een parallel-printer moet worden gestuurd. In plaats van LIST file-naam.extension wordt het nu: P LIST file-naam.extension.

CAT.CMD en DIR.CMD drukken een catalog of directory van een disk af op het beeldscherm. P DIR 1 zet dus de directory van de schijf in drive nummer 1 op papier. DIR geeft uitgebreidere informatie dan CAT.

APPEND.CMD wordt gebruikt voor het aan elkaar knopen van files (voorbeeld: APPEND,file-naam1.ext,file-naam2.ext, etc...).

BUILD.CMD is een heel eenvoudig programma voor het maken van tekst zonder enige tekstverwerkingsfaciliteiten (syntax: BUILD,file-naam). Het is niet nodig een extension mee te geven, omdat de aangemaakte file automatisch de extension TXT krijgt.

COPY.CMD wordt gebruikt voor het dupliceren van files naar een andere schijf.

DELETE.CMD wist files op de schijf. Voor de zekerheid vraagt DELETE altijd een bevestiging van het voornemen om een file te verwijderen (ARE YOU SURE?). Het intikken van Y (yes) volstaat dan voor het verwijderen van de track- en sektorinformatie uit de directory (of catalog). De informatie zelf wordt dus niet verwijderd, maar de gebruikte sektoren worden vrijgegeven voor beschrijving met nieuwe files.

NEWDISC.CMD is waarschijnlijk een van de meest gebruikte files, omdat deze nodig is voor het formatteren van nieuwe

4

```
+++ DIR

DIRECTORY OF DRIVE NUMBER 0

DISK: ELEKTOR #1    CREATED: 28-MAY-85

FILE#   NAME     TYPE      BEGIN    END    SIZE      DATE       PRT

    1   FLEX     .SYS      01-01    01-17    23    28-MAY-85
    2   ERRORS   .SYS      01-18    01-1B     4    28-MAY-85
    3   EXEC     .CMD      01-1C    01-1C     1    28-MAY-85
    4   STARTUP  .TXT      01-1D    01-1D     1    28-MAY-85
    5   VERIFY   .CMD      01-1E    01-1E     1    28-MAY-85
    6   ECHO     .CMD      01-1F    01-1F     1    28-MAY-85
    7   HECHO    .CMD      01-20    01-20     1    28-MAY-85
    8   DATE     .CMD      01-21    01-22     2    28-MAY-85
    9   ASN      .CMD      01-23    01-23     1    28-MAY-85
   10   TTYSET   .CMD      01-24    02-01     2    28-MAY-85
   11   LIST     .CMD      02-02    02-04     3    28-MAY-85
   12   LINK     .CMD      02-05    02-05     1    28-MAY-85
   13   P        .CMD      02-06    02-06     1    28-MAY-85
   14   PRINT    .SYS      02-07    02-07     1    28-MAY-85
   15   CAT      .CMD      02-08    02-0E     7    28-MAY-85
   16   DIR      .CMD      02-0F    02-13     5    28-MAY-85
   17   APPEND   .CMD      02-14    02-16     3    28-MAY-85
   18   BUILD    .CMD      02-17    02-17     1    28-MAY-85
   19   COPY     .CMD      02-18    02-1C     5    28-MAY-85
   20   DELETE   .CMD      02-1D    02-1E     2    28-MAY-85
   21   NEWDISC  .CMD      02-1F    03-01     7    28-MAY-85
   22   0        .CMD      03-02    03-03     2    28-MAY-85
   23   RENAME   .CMD      03-04    03-04     1    28-MAY-85
 FILES=23, SECTORS=76, LARGEST=23, FREE=1328
+++
```

Figuur 2. Dit stroomschema toont het verloop van de startup-procedure. Als Flex is geladen, wordt gezocht naar de file STARTUP.TXT. In die file staan alle kommando's die de computer nodig heeft om de door de gebruiker ingegeven konfiguratie aan te nemen. De kommando's worden door de computer ingelezen alsof ze van het toetsenbord komen.

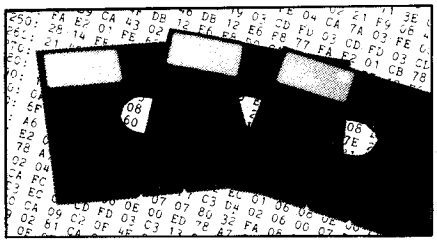
Figuur 3. Zo gaat het formatteren van een 80-track-ds-dd-diskette. Voor het begin van de procedure moet de te formatteren diskette in drive 0 zijn gestopt. Een reeds geformatteerde diskette wordt gewoon opnieuw geformatteerd. Daarbij gaat alle reeds aanwezige informatie op de schijf verloren!

Figuur 4. Een typisch voorbeeld van een directory van een diskette, dat verschijnt als een DIR-kommando wordt gegeven.

schijven. NEWDISC stelt een aantal vragen aan de gebruiker om zo aantal tracks, single of double density, single of double side, volume-naam, volume-nummer vast te kunnen stellen. Na het formatteren kontroleert NEWDISC tevens of alle sektoren bruikbaar zijn en meldt eventueel slechte sektoren (BAD SECTORS).

O.CMD is het output-kommando, waarmee het resultaat van een andere kommandoregel (bijvoorbeeld het opvragen van een directory) niet naar het beeldscherm gaat, maar naar de schijf. Syntaxvoorbeeld : O.DIR. De directory verschijnt niet op het beeldscherm, maar gaat als DIR.OUT naar de disk. Het resultaat van O.... is dus altijd een output-file.

RENAME.CMD sluit de rij. Met RENAME kunnen namen en extensions van files worden veranderd. DEMO.TXT kan bijvoorbeeld een BASIC-demonstratieprogramma zijn dat met de tekstverwerker is gemaakt. XBASIC kan echter geen .TXT-file laden of runnen. Het kommando RENAME DEMO.TXT DEMO.BAS is dan voldoende om de .TXT-extension in een .BAS(BASIC source code)-extension te veranderen.
De zojuist vermelde files (en nog enkele niet genoemde) maken deel uit van het software-pakket dat standaard wordt geleverd bij FLEX. Een uitvoerige beschrijving van deze files vindt men in het manual: The Flex Disk Operating System dat (evenals de Advanced Programmers Guide) meegeleverd wordt met de software. Tenslotte willen we als voorbeeld een directory geven van een disk waarop de tot nu toe besproken files staan opgeslagen (figuur 4). De door de gebruiker ingegeven tekst is hierbij onderstreept.
En dan nog een allerlaatste opmerking. Met Flex is het mogelijk met meerdere disk-drives tegelijk te werken. Bij gebruik van één drive is drive 0 automatisch zowel system- als workdrive (zie de bespreking van de ASN-file). Bij meerdere drives typt men het drive-nummer voor de naam van de kommando-file. 2 DIR geeft bijvoorbeeld de directory van de disk in drive nummer 2. Kommando's kunnen bij Flex overigens altijd worden ingetikt na het verschijnen van de system-prompt "+++".



Flex-utilities

Technical Systems Consultants biedt voor de uitbreiding en ondersteuning van het disk-operating-system een breed programma "utilities" aan. Daartoe behoren enkele kleine programma's die voor de Flex-gebruiker eigenlijk onmisbaar zijn. In dit artikel bespreken we de interessantste programma's uit de zesdelige utility-set. In de dokumentatie die bij de diskette geleverd wordt, staan alle op deze schijf aanwezige utilities natuurlijk uitgebreid beschreven.
In dit artikel introduceren we enkele nieuwe termen. We zullen bij elke file aangeven: de syntax, de default-waarden, een voorbeeld, het resultaat van de bewerking en eventueel hoe de output er uit zal zien. Met syntax wordt bedoeld: de exakte formulering van de kommandoregel (de grammatica van de computer dus); default staat voor een waarde of een tekst die automatisch door de computer wordt aangenomen indien de gebruiker zelf niets heeft ingetypt. Een tekstverwerker zal bijvoorbeeld automatisch naar een file met de extension .TXT (tekst) zoeken indien de gebruiker in de kommandoregel geen extension heeft aangegeven. Indien er een tekstfile in een voorbeeld wordt gebruikt, dan is dat een willekeurige file met de naam "PARAGR1.TXT". Maar nu terzake.

FIND

FIND.CMD zoekt strings in tekstfiles en toont de regels waarin de aangegeven strings staan op het beeldscherm (of op papier). Eigenlijk is FIND typisch een kommando uit een tekstverwerker (het komt daarin ook voor!). Hierin ligt ook de grote kracht van deze command-file. Het is namelijk niet nodig een tekstverwerker te gebruiken om in een tekstfile (of enige andere file) een bepaalde letter, een woord, een getal of zelfs een komplete zin te zoeken. Het laden van een tekstverwerker van gemiddelde lengte kost meer tijd dan het laden van FIND.CMD!
* syntax: FIND,file-naam,te zoeken string

* defaults: file-extension is TXT

* voorbeeld:
+++ FIND,PARAGR1,disk

* resultaat: Op het scherm verschijnen alle tekstregels uit de file "PARAGR1" waarin de tekst "disk" staat. Voorts worden de regelnummers gegeven.

* output:
+++ FIND,PARAGR1,Disk
(regelnummer)(eerste regel waarin "disk" staat)
(regelnummer)(tweede regel waarin "disk" staat)
...
...
TOTAL STRING OCCURENCE IS XX

Zoals we in het voorbeeld kunnen zien, wordt na alle uitgegeven regels ook nog het aantal gevonden plaatsen getoond.

WORDS

WORDS.CMD is nog een programma dat goed van pas kan komen bij het gebruik van teksten. Met WORDS wordt het aantal woorden en regels in een tekst-file geteld. Een handig hulpje bij het maken van teksten die gebonden zijn aan een maximum of minimum aantal woorden en/of regels (verslagen bijvoorbeeld).

* syntax: WORDS,file-naam

* defaults: extension is TXT

* voorbeeld:
+++ WORDS,PARAGR1

* Resultaat: Op het scherm verschijnen het aantal woorden en regels die voorkomen in de tekst-file PARAGR1. Als markering tussen de woorden worden de spatie en het teken nieuwe regel/cursor-naar-rechts herkend.

* output:
+++ WORDS,PARAGR1
TOTAL WORD COUNT IS 1662
TOTAL LINE COUNT IS 388

TYPOS

TYPOS.CMD is een luxe versie van WORDS. TYPOS telt niet alleen de woorden, maar vermeldt er ook nog bij hoeveel keer ze in de tekst voor komen. Dit programma is uitermate praktisch als men wil kontroleren of een bepaald woord niet al te vaak voorkomt. Omdat sommige woorden wel erg vaak worden gebruikt (zoals de, het en een) kan men ook een maximaal aantal opgeven. Doet men dit niet, dan kiest het systeem automatisch het aantal drie.

* syntax: TYPOS,file-naam,max. aantal

* defaults: extension is TXT, aantal is 3 (als er geen aantal wordt ingetypt, worden alle woorden die drie maal en minder voorkomen, afgedrukt).

voorbeeld:
+++ TYPOS,PARAGR1,5

* resultaat: een lijst van alle woorden die in de tekst-file PARAGR1 worden alle woorden opgesomd die 1, 2, 3, 4 en 5 maal voorkomen (eerst het aantal en dan het woord zelf), te beginnen met het grootste aantal (hier dus 5).

* output: spreekt voor zich en zou alleen maar bladvulling zijn (wordt derhalve achterwege gelaten).

1

```
+++ DUMP.SCRHELP.SYS

03 09
03 0A 00 01 43 6F 6D 6D 61 6E 64 73 20 69 6E 20 ____Commands in
74 68 65 20 63 6F 6D 6D 61 6E 64 20 6D 6F 64 65 the command mode
3A 0D 0D 50 41 47 45 3D 4E 09 0B 53 65 74 20 6E :__PAGE=N__Set n
65 77 20 70 61 67 65 6C 65 6E 67 74 68 2E 0D 4C ew pagelenoth._L
49 4E 45 3D 4E 09 0B 53 65 74 20 6E 65 77 20 6C INE=N__Set new l
69 6E 65 6C 65 6E 67 74 68 2E 0D 53 54 41 54 55 inelenoth._STATU
53 3D 54 52 55 45 09 06 45 6E 61 62 6C 65 20 73 S=TRUE__Enable s
74 61 74 75 73 20 6C 69 6E 65 20 69 6E 20 74 65 tatus line in te
78 74 20 6D 6F 64 65 2E 0D 53 54 41 54 55 53 3D xt mode._STATUS=
46 41 4C 53 45 09 05 44 69 73 61 62 6C 65 20 73 FALSE__Disable s
74 61 74 75 73 20 6C 69 6E 65 2E 0D 49 4E 53 45 tatus line._INSE
52 54 3D 54 52 55 45 09 06 45 6E 61 62 6C 65 20 RT=TRUE__Enable
61 75 74 6F 20 66 69 65 6C 64 69 6E 67 2E 0D 49 auto fieldino._I
4E 53 45 52 54 3D 46 41 4C 53 45 09 05 44 69 73 NSERT=FALSE__Dis
61 62 6C 65 20 61 75 74 6F 20 66 69 65 6C 64 69 able auto fieldi
6E 67 2E 0D 46 49 45 4C 44 3D 4E 09 0A 44 65 66 no._FIELD=N__Def

03 0A
03 0B 00 02 69 6E 65 20 66 69 65 6C 64 20 63 68 ____ine field ch
61 72 61 63 74 65 72 20 61 73 20 64 65 63 69 6D aracter as decim
61 6C 20 6E 75 6D 62 65 72 2E 0D 46 49 45 4C 44 al number._FIELD
3D 27 43 09 09 44 65 66 69 6E 65 [illegible]
64 20 [illegible]
53 [illegible]
```

Figuur 1. Dit voorbeeld van een DUMP.CMD-uitdraai spreekt voor zich. De twee eerste bytes van een sektor bevatten steeds het adres van de volgende sektor. Bij files met een lengte van slechts één sektor staat daar 0000. De file SCRHELP.SYS is overigens een utility die hoort bij een tekstverwerkingsprogramma.

SPLIT

SPLIT.CMD komt goed van pas bij het werken met (te) grote teksten. Vaak blijken te grote teksten onoverzichtelijk te worden en is het wenselijk met een gedeelte van de originele tekst verder te werken. Met SPLIT kan een tekst worden gesplitst in twee nieuwe teksten. Het afbreekpunt is een regelnummer.

* syntax: SPLIT,input file-naam (naam van de bestaande file),output file-naam nummer 1 (naam van de file die loopt van regel 1 tot aan het afbreekpunt),output file-naam nummer 2 (naam van de file die de tekst bevat vanaf het breekpunt tot aan het einde van de originele file),nummer van de regel waar de scheiding ligt tussen de twee nieuwe files.

* defaults: extension is TXT

* voorbeeld:
+++ SPLIT,PARAGR1,DEEL1,DEEL2,100

* resultaat: de tekst-file paragr1 wordt gesplitst in een tekst-file deel 1, die regel 1 tot en met regel 99 bevat, en een tekst-file deel 2 die regel 100 tot het einde van de tekst in PARAGR1 bevat. Bij deze operatie blijft de originele tekst PARAGR1.TXT ongeschonden.

LOW-UP, UP-LOW

LOW-UP.CMD en UP-LOW.CMD zijn command-files die zeer sterk op elkaar lijken. Ze worden gebruikt voor het konverteren van kleine letters naar hoofdletters en omgekeerd. Een omzetting van kleine letters naar hoofdletters kan bijvoorbeeld van pas komen bij BASIC-programma's die geschreven zijn voor een BASIC-interpreter die beide soorten letters kan verwerken, en vervolgens "gerund" moeten worden met een BASIC-interpreter die uitsluitend hoofdletters verwerken kan.

syntax: LOW-UP,originele file-naam met grote en kleine letters,naam van de nieuwe file waarin alle kleine letters zijn vervangen door hoofdletters. Het is vrij duidelijk hoe UP-LOW er uit zal zien.

* defaults: extension is TXT

* voorbeeld:
+++ LOW-UP,KLEIN,GROOT

* resultaat: de tekst-file KLEIN (kleine en hoofdletters) wordt omgezet in de file HOOFD (die geen kleine letters meer bevat, maar alleen hoofdletters).

DUMP

DUMP.CMD wordt gebruikt voor het "dumpen" (op het beeldscherm of op papier) van de inhoud van een of meerdere disk-sektoren. De output geeft links boven het track- en sektor-adres van de sektor (of bij langere files van de diverse sektoren) en vervolgens op elke regel telkens 16 bytes in hex-vorm met helemaal rechts de daarbij behorende ASCII-waarden. Dit programma komt goed van pas als men bij beschadigde disks wil nagaan wat er nog op de schijf staat (na een crash).

* syntax: DUMP,file-naam.extension

* defaults: extension is BIN (binary)

* voorbeeld:
+++ DUMP,SCRHELP.SYS

* resultaat: op het beeldscherm verschijnt het start-adres van de file SCRHELP.SYS (track-nummer en sektor-nummer uitgedrukt in hex) met daaronder 16 regels met op elke regel 16 bytes en daarnaast de eventuele ASCII-kode. Zo worden alle sektoren (elke sektor bevat 256 bytes) achter elkaar afgebeeld.

* output: zie figuur 1

CHECK

CHECK.CMD wordt gebruikt om de inhoud van twee files met elkaar te verge-

lijken. Het resultaat van de vergelijking (dat kan zijn: THE FILES CHECKED DO NOT MATCH of THE FILES CHECKED ARE IDENTICAL) wordt op het beeldscherm afgedrukt. Dit is een handig hulpmiddel wanneer de verdenking bestaat dat men te maken heeft met twee files met verschillende naam, maar met dezelfde inhoud.

* syntax: CHECK,file-naam1,file-naam2

* defaults: extension is TXT

* voorbeeld:
+++
CHECK,PARAGR1.TXT,PARAGR1.BAK

* resultaat: de inhouden van de tekst-file PARAGR1 en de backup-file worden met elkaar vergeleken.

* output:
+++
CHECK,PARAGR1.TXT,PARAGR1.BAK
THE FILES CHECKED ARE IDENTICAL

MAP

MAP.CMD is een programma ter bepaling van de laad- en executie-adressen van command-files. Het geeft dus aan waar in het geheugen de "gemapte" command-file wordt weggeschreven en op welk adres executie van dat programma plaats zal vinden (dit adres wordt ook het transfer-adres genoemd). Dit programma komt goed van pas in kombinatie met het SAVE-kommando (waarbij de inhoud van aangegeven geheugenblokken onder een gekozen naam naar schijf wordt weggeschreven).

* syntax: MAP,file-naam

* defaults: extension is BIN

* voorbeeld:
+++ MAP,MAP.CMD

* resultaat: Op het beeldscherm worden start- en eind-adressen afgebeeld van de geheugenblokken waarin MAP.CMD wordt ingelezen, gevolgd door het transfer-adres.

* output:
+++ MAP,MAP.CMD
C100-C126
C100-C1E8
C100

FREE

FREE.CMD geeft aan hoeveel plaats er nog over is op de disk. Dit wordt uitgedrukt in het aantal sektoren en de hoeveelheid kilobytes.

* syntax: FREE X

* defaults: drive 0

* voorbeeld: FREE

* resultaat: op het beeldscherm verschijnt de tekst:
SECTORS REMAINING = X
APPROXIMATE KILOBYTES = YY

* output:
+++ FREE SECTORS REMAINING = 7
APPROXIMATE KILOBYTES = 17

TEST

TEST.CMD kan gebruikers van een slechtere kwaliteit floppies een hoop ellende besparen. Alle sektoren op de disk worden getest en de eventuele slechte sektoren (bad sectors) worden gerapporteerd (met hun hex-adres).

* syntax: TEST,drive-nummer

* defaults: work-drive (zoals gekozen met ASN)

* voorbeeld: TEST,0

* resultaat: de diskette in drive nummer 0 wordt getest op bad sectors (de foutmelding zegt: BAD SECTOR AT . . . .). Uiteraard zal er ook een foutmelding plaatsvinden indien er een drive-nummer wordt gekozen van een niet aanwezige drive (melding: ILLEGAL DRIVE NUMBER).

* output:
+++ TEST,0
BAD SECTOR AT NN-MM
BAD SECTOR AT XX-YY
enz.
. . .
. . .
TEST COMPLETED

of:

+++ TEST,0
TEST COMPLETED

De laatste melding vindt plaats indien er geen bad sectors aangetroffen werden.

RPT

RPT.CMD kan veel werk besparen bij het "runnen" van programma's die meerdere malen dienen te worden uitgevoerd ("gerund"). Normaliter zal de gebruiker geneigd zijn de bewuste kommandoregel telkens opnieuw in te tikken. Dat is mogelijk maar niet erg efficiënt. Bij gebruik van RPT (repeat) hoeft alleen maar te worden ingetikt hoeveel keer de aangegeven kommandoregel dient te worden herhaald.

* syntax: RPT,aantal keer dat de kommandoregel dient te worden herhaald,de kommandoregel zelf.

* defaults: zijn niet van toepassing

* voorbeeld: RPT,3,P LIST PARAGR1.TXT

* resultaat: de inhoud van de tekst-file PARAGR1 wordt drie maal op papier afgedrukt.

PDEL

PDEL.CMD betekent: prompting delete. Bij de gewone delete geeft men de naam aan van de file of files die moeten worden gewist. PDEL vraagt niet om de namen van de files die gewist moeten worden, maar geeft de namen op het beeldscherm, waarna de gebruiker bij elke file aan kan geven of deze moet worden gewist of niet. De volgorde waarin men dat doet, komt overeen met de volgorde van de files in de directory. Bij elke naam op het beeldscherm moet achter het vraagteken (verschijnt achter de filenaam) een Y (delete ? antwoord: YES) of N (No) of een carriage-return-teken (dat betekent: breek uitvoering van PDFEL af) worden ingetikt. Na het afbreken komt het systeem terug in de command-mode (+++). Pas op: PDEL vraagt niet om verifikatie! Bij het gebruik van DELETE.CMD wordt er altijd nog gevraagd: ARE YOU SURE? PDEL doet dat niet. Y intikken betekent dus: weg file. In de kommandoregel kan nog een extra selektie worden gemaakt. Het opnemen van .TXT heeft zo tot gevolg dat alleen de files met de extension .TXT worden afgedrukt. Het is ook mogelijk alle files te nemen die beginnen met de letter A (of een kombinatie van letters), etc.

* syntax: PDEL,drive-nummer(,opties-lijst)

* defaults: zijn niet van toepassing.

* voorbeeld: PDEL,0,.TXT

* resultaat: Alle tekst-files die voorkomen op de diskette in drive 0, worden op het scherm getoond, waarbij telkens een Y, N of Carriage Return moet worden ingetoetst. Daarna verschijnt pas de volgende naam verschijnt. Dat gaat zo door totdat de laatste file is verschenen of een carriage return is ingetikt.

MEMDUMP

MEMDUMP.CMD maakt het mogelijk de inhoud van het geheugen op het beeldscherm weer te geven. Dat gebeurt in blokken van 256 bytes. Na het dumpen van het eerste blok kan de gebruiker kiezen uit een drietal kommando's: F (forward), B (backward) of carriage return. Na het intikken van F of f wordt het volgende geheugenblok van 256 bytes afgedrukt. Bij B of b wordt het vorige geheugenblok getoond). De carriage return heeft ook hier weer de betekenis van afbreken van het programma en teruggaan naar de command-mode.

* syntax: MEMDUMP,startadres in hex

* defaults: indien geen startadres wordt ingetikt, begint MEMDUMP automatisch op adres 0000.

* voorbeeld: MEMDUMP,F000

* resultaat: op het beeldscherm verschijnt de inhoud van geheugenblok F000 tot F0FF.

MEMTEST

MEMTEST.CMD spreekt voor zich. Dit programma test alle RAM-geheugenplaatsen.

* syntax: MEMTEST,startadres,eindadres

* defaults: zijn niet van toepassing.

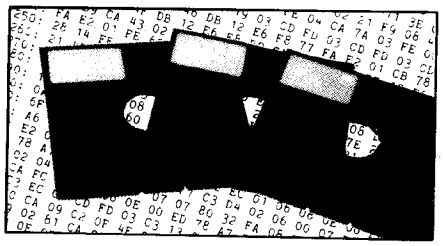
* voorbeeld: MEMTEST,0,01FF

* resultaat: de geheugenlokaties van 0000

tot ØlFF worden met toevalsnummers gevuld, waarna die nummers weer worden teruggelezen en vergeleken met de originele. Dit wordt een tijdje herhaald (het testen van een blok van 4 K duurt ongeveer 60 minuten). Er mag natuurlijk niet getest worden in het geheugengebied waar het programma MEMTEST zelf staat (de oplettende lezer weet al lang dat met behulp van het programma MAP, dat al eerder werd beschreven, makkelijk kan worden vastgesteld op welke geheugenlokaties MEMTEST.CMD werd weggeschreven). Het met sukses afwerken van een testcyclus wordt aangegeven door het afdrukken van een "1" op het beeldscherm. Het onderbreken van MEMTEST gaat op een vrij brute manier: door het indrukken van de reset-knop.

Bij de aanschaf van de Utility-Commandset van TSC zal blijken dat er nog veel meer utilities in die set voorkomen. Voor de duidelijkheid volgt ter afsluiting van dit artikel nog een lijstje (in alfabetische volgorde) van de besproken utilities:

CHECK
DUMP
FIND
FREE
LOW-UP
MAP
MEMDUMP
MEMTEST
PDEL
RPT
SPLIT
TEST
TYPOS
UP-LOW
WORDS



graphics-software

Zoals in onze tweede Elektuur Computing beloofd, gaan we hier verder met het tweede en derde deel over de stuursoftware voor de grafische kleurenkaart. De hardware van deze kaart hebben we, zoals u weet, reeds in diverse Elektuur-nummers besproken.
Hier dus nu deel 2 ("graphics") en deel 3 ("circle") van de stuur-software voor de 6502-processor. Bij de Octopus 65 ligt de stuursoftware in het bereik $C000...CFFF en kan heel eenvoudig op een systeemschijf worden "geïmplementeerd". Zo:
Op een 80-tracks-diskette is voor het programma nog plaats genoeg beschikbaar. Bij een 40-tracks-exemplaar is dat helaas niet het geval en moet er een utility-programma met een lengte van 2 tracks van de kopie van de systeemschijf worden verwijderd (bijvoorbeeld eentje die men toch niet vaak gebruikt). Met de DOS-bevelen "CAll" en "SAve" worden de twee tracks dan op de dienovereenkomstige sporen van de graphics-systeemschijf gezet. Daarna moet in het programma BEXEC* op een geschikte plaats een extra kommandoregel worden ingebouwd. Bijvoorbeeld als volgt:
1905 DISK!"CA C000=60,1":DISK!"CA C800=61,1":DISK!" GO C000"
Uiteraard is dit slechts een voorbeeld. Het regelnummer en de track-nummers hangen af van de BEXEC*-versie die men heeft en de voor de grafische software gebruikte tracks. Ga dus eerst na hoe het bij uw versie zit! De grafische software wordt, als alles goed is gedaan, automatisch bij het booten meegeladen. Men kan het grafische gedeelte dus direkt na het laden met BASIC-regels aanspreken (ook in BASIC-immediate-mode).
Tot slot nog even dit: op de binnenkant van de omslag ziet u enkele fraaie kleurenfoto's die tonen waartoe de diverse demoprogramma's in staat zijn. Bij het aanschouwen van de beelden zult u het met ons eens zijn dat de grafische kaart behoorlijk wat in zijn mars heeft. En dit zijn nog maar "stilstaande" plaatjes! In werkelijkheid verandert de afbeelding namelijk voortdurend van vorm, zodat met recht over "kunst uit de computer" kan worden gesproken. Maar dat zult u zelf wel zien wanneer u de diverse demo's op uw eigen Octopus laat draaien.
Een allerlaatste opmerking nog: Omdat de bij de kleurenfoto's behorende programma's deels te lang waren om ze ernaast af te drukken, hebben we de demoprogramma's opgenomen in het hoofdstukje "programma's", dat u verderop in deze Elektuur Computing kunt vinden.

```
.EX
07 TRACK(S)
A%LO SAAMG_ _ _ MM_ GRA

A%re_ _ RE AS


.A

  10          ;#################################
  20          ;#     V I D E O                 #
  30          ;#     ---------                 #
  40          ;#VIDEO-UTILITIES WITH GDP9366  #
  50          ;# WRITTEN BY P.LAVIGNE  NOV'83 #
  60          ;#                               #
  70          ;#  PART II : GRAPHICS           #
  80          ;#                               #
  90          ;#################################
 100          ;
 110 E200          X = $E200   ;GDP REGS
 120 E200=      STATUS = X
 130 E200=      CMD    = X
 140 E201=      CTRL1  = X+1
 150 E202=      CTRL2  = X+2
 160 E203=      CSIZE  = X+3
 170 E205=      DELTAX = X+5
 180 E207=      DELTAY = X+7
 190 E208=      MSBX   = X+8
 200 E209=      LSBX   = X+9
 210 E20A=      MSBY   = X+10
 220 E20B=      LSBY   = X+11
 230          ;
 240 E218          X = $E218   ;HARDWARE REGS
 250 E218=      COLOR  = X
 260 E219=      SCROLL = X+1
 270 E21A=      PAGE   = X+2
 280 E21B=      HSCROL = X+3
 290          ;
 300 CF80          X = $CF80   ;SOFTWARE REGS
 310 CF85=      MIRXH  = X+5
 320 CF86=      MIRXL  = X+6
 330 CF87=      MIRYH  = X+7
 340 CF88=      MIRYL  = X+8
 350 CF89=      CHAR   = X+9
 360 CF8A=      PIXBUF = X+10
 370 CF8D=      GRCNT  = X+13
 380 CF91=      COLRPT = X+17
```

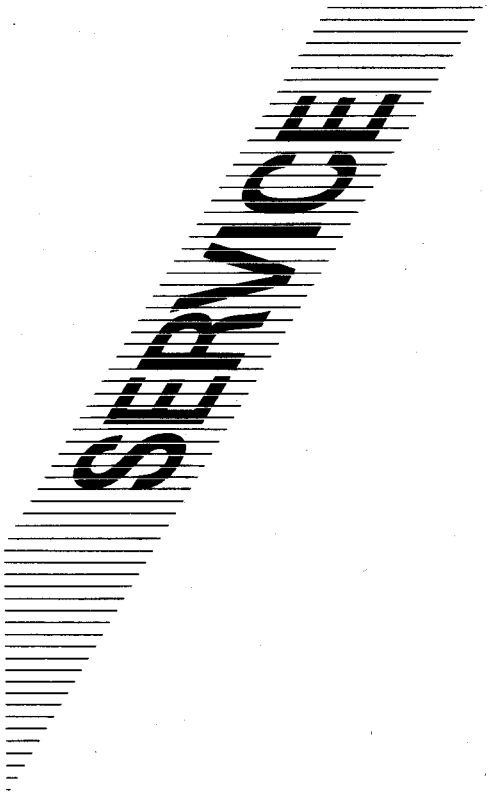
```
390 CF93=       PAGEPT = %+19
400 CF94=       MODE   = %+20
410 CF95=       TXTMOD = %+21
420 CF97=       BACKGR = %+23
430 CF9C=       INDJMP = %+28
440 CF9E=       DATA2  = %+30
450 CFA0=       DATA1  = %+32
460 CFA2=       DATA0  = %+34
470             ;
480 0100=       STACK  = $100
490             ;
500 D000            % = $D000
510             ;
520 D08A=       READY  = %+$8A
530             ;
540 D508              % = %+$508
550             ;
560 D508 2018D5 GRAPHI JSR GRAPH
570 D50B AD95CF        LDA TXTMOD
580 D50E F007          BEQ RETGRA   ;IT WAS NO GRAPH IN TEXT
590 D510 AD89CF        LDA CHAR     ;TEST FOR <CR>
600 D513 C90D          CMP #$D
610 D515 F01F          BEQ GOTEXT
620 D517 60     RETGRA RTS          ;NORMAL EXIT
630             ;
640 D518 8D89CF GRAPH  STA CHAR   ;SAVE (A)
650 D51B C050          CPY #'P    ;PRINTING IN GRAPHIC ?
660 D51D D00F          BNE TESTA  ;IF NOT CONT TESTING
670 D51F 2041D6        JSR PRTEST ;SEE FOR PRINTABLE CHAR
680 D522 9015          BCC PRINT
690 D524 C90D          CMP #$D    ;IS IT äCRü?
700 D526 D005          BNE RETA
710 D528 A901          LDA #1
720 D52A 8D94CF        STA MODE
730 D52D 60     RETA   RTS
740 D52E C911   TESTA  CMP #$11   ;CHR$(17)? IF SO --)TEXTMODE
750 D530 F004          BEQ GOTEXT
760 D532 C941          CMP #'A    ;OR 'A ? IF SO--)TXTMOD TOO
770 D534 D009          BNE TESTB
780 D536 4C6CD6 GOTEXT JMP BCKTXT
790 D539 8D00E2 PRINT  STA CMD
800 D53C 4C8AD0        JMP READY
810 D53F C920   TESTB  CMP #'     ;äSPü?
820 D541 F0EA          BEQ RETA
830 D543 203AD6        JSR AZTEST
840 D546 B059          BCS TESTC
850 D548 2048D6        JSR ZEROBF ;DATA0,1,2=0
860 D54B AD89CF        LDA CHAR
870 D54E 297F          AND #$7F
880 D550 8D94CF        STA MODE
890 D553 38            SEC
900 D554 E942          SBC #'B
910 D556 0A            ASL A
920 D557 AA            TAX
930 D558 BD6FD5        LDA GRAPAD,X ;GET INDIRECT JUMPADDRESS
940 D55B 8D9CCF        STA INDJMP
950 D55E BC70D5        LDY GRAPAD+1,X
960 D561 8C9DCF        STY INDJMP+1
970 D564 C0FF          CPY #$FF  ;SEE FOR NOT-USED ADDRESSES
980 D566 D004          BNE JUMPGR
990 D568 C9FF          CMP #$FF
1000 D56A F0C1         BEQ RETA
1010 D56C 6C9CCF JUMPGR JMP (INDJMP)
1020 D56F ADD6   GRAPAD .WORD B,C,D,$FFFF,$FFFF,G
1020 D571 F3D6
1020 D573 8FD7
1020 D575 FFFF
1020 D577 FFFF
1020 D579 D9D9
1030 D57B 92D6          .WORD H,I,J,$FFFF,L,M,$FFFF,O,$FFFF,Q
1030 D57D 9FD6
1030 D57F CCD7
1030 D581 FFFF
1030 D583 18D7
1030 D585 71D8
1030 D587 FFFF
1030 D589 AED9
1030 D58B FFFF
1030 D58D 29D7
1040 D58F 84D8          .WORD RR,SS,T,U,V,W,XX
1040 D591 CDD8
1040 D593 41D7
1040 D595 EFD8
1040 D597 16D9
1040 D599 5ED7
1040 D59B 42D9
1050 D59D FFFF          .WORD $FFFF,Z
1050 D59F 71D7
1060 D5A1 C001   TESTC  CPY #1
1070 D5A3 F020          BEQ RETB
1080 D5A5 A8            TAY
1090 D5A6 AD94CF        LDA MODE  ;SAVE (A) IN (Y) AND GET MODE
1100 D5A9 101B          BPL TSTCOM
1110 D5AB 297F          AND #$7F
1120 D5AD 8D94CF        STA MODE
1130 D5B0 C02B          CPY #'+
1140 D5B2 F011          BEQ RETB
1150 D5B4 C02D          CPY #'- ;TEST + OR -
1160 D5B6 D00E          BNE TSTCOM
1170 D5B8 A902          LDA #2
1180 D5BA 38            SEC
1190 D5BB ED8DCF        SBC GRCNT
1200 D5BE 0A            ASL A   ;GET APPROPRIATE BUFFER
1210 D5BF AA            TAX
1220 D5C0 A980          LDA #$80  ;SET MSB FOR NEG.
1230 D5C2 9D9ECF        STA DATA2,X
1240 D5C5 60    RETB    RTS
1250 D5C6 C02C   TSTCOM CPY #',  ;COMMA?
1260 D5C8 D00B          BNE TSTCR ;IF NOT SEE FOR äCRü
1270 D5CA 0980          ORA #$80  ;SET MSB IN MODE
1280 D5CC 8D94CF        STA MODE
1290 D5CF CE8DCF        DEC GRCNT
1300 D5D2 3098          BMI JUMPGR
1310 D5D4 60            RTS
1320 D5D5 C00D   TSTCR  CPY #$D    ;äCRü?
1330 D5D7 D007          BNE TST09
1340 D5D9 0980          ORA #$80    ;SET MSb IN MODE
1350 D5DB 8D94CF        STA MODE
1360 D5DE D08C          BNE JUMPGR
1370 D5E0 98    TST09   TYA
1380 D5E1 2031D6        JSR DATEST ;0-9?
1390 D5E4 B04A          BCS RETC
1400 D5E6 290F   UPDATE AND #$F      ;STRIP ASCII
1410 D5E8 48            PHA          ;SAVE
1420 D5E9 A902          LDA #2        ;GET POINTER IN DATAFIELD
1430 D5EB 38            SEC
1440 D5EC ED8DCF        SBC GRCNT
1450 D5EF 0A            ASL A
1460 D5F0 AA            TAX
1470 D5F1 BD9ECF        LDA DATA2,X ;10 TIMES DATA & ADD
1480 D5F4 48            PHA         ;NEW NUMBER
1490 D5F5 BD9FCF        LDA DATA2+1,X ;DONT LOSE SIGN
1500 D5F8 48            PHA
1510 D5F9 1E9FCF        ASL DATA2+1,X ;DO X 4
1520 D5FC 3E9ECF        ROL DATA2,X
1530 D5FF 1E9FCF        ASL DATA2+1,X ;SIGN IS LOST
1540 D602 3E9ECF        ROL DATA2,X
1550 D605 18            CLC           ;ADD FOR X 5
```

```
1560 D686 68            PLA
1570 D687 7D9FCF        ADC DATA2+1,X
1580 D68A 9D9FCF        STA DATA2+1,X
1590 D68D 68            PLA
1600 D68E 7D9ECF        ADC DATA2,X
1610 D611 9D9ECF        STA DATA2,X   ;SIGN IS IN MSb
1620 D614 1E9FCF        ASL DATA2+1,X
1630 D617 3E9ECF        ROL DATA2,X   ;SIGN IN (C)
1640 D61A 68            PLA           ;NOW ADD NEW NUMBER
1650 D61B 08            PHP           ;SAVE SIGN FIRST
1660 D61C 18            CLC
1670 D61D 7D9FCF        ADC DATA2+1,X
1680 D620 9D9FCF        STA DATA2+1,X
1690 D623 A900          LDA #0
1700 D625 7D9ECF        ADC DATA2,X
1710 D628 28            PLP           ;GET SIGN
1720 D629 9002          BCC PLUS
1730 D62B 0980          ORA #$80      ;IF NEGAT.
1740 D62D 9D9ECF  PLUS  STA DATA2,X
1750 D630 60      RETC  RTS
1760              ;
1770              ;GRAPH TEST SUBROUTINES ETC.
1780              ;
1790 D631 C930    DATEST CMP #'0
1800 D633 9003           BCC NODATA
1810 D635 C93A           CMP #$3A
1820 D637 60             RTS
1830 D638 38      NODATA SEC
1840 D639 60             RTS
1850 D63A C941    AZTEST CMP #'A
1860 D63C 90FA           BCC NODATA
1870 D63E C95B           CMP #$5B
1880 D640 60             RTS
1890 D641 C920    PRTEST CMP #'     ;TEST FOR PRINTABLE CHAR
1900 D643 90F3           BCC NODATA
1910 D645 C980           CMP #$80
1920 D647 60             RTS
1930              ;
1940 D648 A900    ZEROBF LDA #0
1950 D64A A005           LDY #5
1960 D64C 999ECF  ZERO   STA DATA2,Y
1970 D64F 88             DEY
1980 D650 10FA           BPL ZERO
1990 D652 60             RTS
2000              ;
2010 D653 AD94CF  ENDMOD LDA MODE   ;COMMAND EXECUTED
2020 D656 297F           AND #$7F ;RESET MSB
2030 D658 8D94CF         STA MODE
2040 D65B 2048D6         JSR ZEROBF ;IF MORE COMMAND SETS
2050 D65E AD89CF         LDA CHAR
2060 D661 C92C           CMP #',
2070 D663 D003           BNE SETMOD
2080 D665 6C9CCF         JMP (INDJMP) ;GO FOR NEXT SET
2090 D668 A901    SETMOD LDA #1
2100 D66A D022           BNE STOMOD
2110 D66C A900    BCKTXT LDA #0
2120 D66E 8D95CF         STA TXTMOD
2130 D671 8D94CF  STMOD  STA MODE
2140 D674 A90B           LDA #$B
2150 D676 8D01E2         STA CTRL1
2160 D679 A907    ADJTST LDA #7
2170 D67B 2D0BE2         AND LSBY
2180 D67E F005           BEQ ENDAD
2190 D680 EE0BE2         INC LSBY
2200 D683 D0F4           BNE ADJTST
2210 D685 60      ENDAD  RTS
2220              ;
2230              ;DIRECT COMMANDS (H & I)
2240              ;
2250 D686 8D8DCF  SETCNT STA GRCNT     ;TAIL OF 1-ST ENTRY IN ROUTIN
2260 D689 AD94CF         LDA MODE      ;SET MSB
2270 D68C 0980           ORA #$80
2280 D68E 8D94CF  STOMOD STA MODE
2290 D691 60             RTS
2300              ;
2310 D692 A203    H      LDX #3    ;HOME PEN TO CURR. ORIG.
2320 D694 BD85CF  HOME   LDA MIRXH,X ;WITHOUT DRAWING
2330 D697 9D08E2         STA MSBX,X
2340 D69A CA             DEX
2350 D69B 10F7           BPL HOME
2360 D69D 300B           BMI DONI
2370              ;
2380 D69F A203    I      LDX #3    ;SET CURR LOC AS ORIGIN
2390 D6A1 BD08E2  NEWOR  LDA MSBX,X
2400 D6A4 9D85CF         STA MIRXH,X
2410 D6A7 CA             DEX
2420 D6A8 10F7           BPL NEWOR
2430 D6AA 4C68D6  DONI   JMP SETMOD
2440              ;
2450              ;ONE DATA COMMANDS
2460              ;
2470 D6AD AD94CF  B      LDA MODE    ;SET BACKGROUND COLOR
2480 D6B0 3005           BMI SETPAP
2490 D6B2 A900           LDA #0
2500 D6B4 4C86D6         JMP SETCNT
2510 D6B7 ADA3CF  SETPAP LDA DATA0+1   ;GET REGISTER LOW BYTE
2520 D6BA 290F           AND #$F    ;STRIP MSN
2530 D6BC 48             PHA
2540 D6BD 2CA2CF         BIT DATA0
2550 D6C0 100A           BPL CLEAR  ;SEE FOR ERASE TOO
2560 D6C2 2D97CF         AND BACKGR
2570 D6C5 0A             ASL A
2580 D6C6 0A             ASL A
2590 D6C7 0A             ASL A
2600 D6C8 0A             ASL A
2610 D6C9 0DA3CF         ORA DATA0+1
2620 D6CC 8D91CF  CLEAR  STA COLRPT
2630 D6CF 8D18E2         STA COLOR
2640 D6D2 A902           LDA #2
2650 D6D4 2C00E2  WAIT0  BIT STATUS
2660 D6D7 D0FB           BNE WAIT0
2670 D6D9 2C00E2  WAIT1  BIT STATUS
2680 D6DC F0FB           BEQ WAIT1
2690 D6DE A90C           LDA #$C
2700 D6E0 8D00E2         STA CMD
2710 D6E3 20BAD0         JSR READY
2720 D6E6 68             PLA
2730 D6E7 8D97CF         STA BACKGR ;NEW COLOR--->BACKGR.
2740 D6EA AD91CF         LDA COLRPT
2750 D6ED 8D18E2         STA COLOR  ;RESET ORIGINAL COLOR
2760 D6F0 4C68D6         JMP SETMOD ;RET. VIA CLOSING COMMAND
2770              ;
2780 D6F3 AD94CF  C      LDA MODE    ;SET PEN COLOR
2790 D6F6 1036           BPL STCNT0
2800 D6F8 ADA3CF  SETPEN LDA DATA0+1 ;GET DATA
2810 D6FB 290F           AND #$F
2820 D6FD 8DA3CF         STA DATA0+1
2830 D700 2CA2CF         BIT DATA0  ;SEE FOR POS.
2840 D703 100A           BPL CLRPEN
2850 D705 2D97CF         AND BACKGR ;USE BACKGR TO LOOK
2860 D708 0A             ASL A      ;FOR BITS NOT USED BY
2870 D709 0A             ASL A      ;THE 'WRITING' COLOR
2880 D70A 0A             ASL A
2890 D70B 0A             ASL A
2900 D70C 0DA3CF         ORA DATA0+1
2910 D70F 8D91CF  CLRPEN STA COLRPT
2920 D712 8D18E2         STA COLOR
2930 D715 4C68D6         JMP SETMOD
```

```
2940            ;
2950 D718 AD94CF  L       LDA MODE    ;GET LINE TYPE
2960 D71B 1011            BPL STCNT0
2970 D71D ADA3CF  LINTYP  LDA DATA0+1
2980 D720 2903            AND #3
2990 D722 8DA3CF          STA DATA0+1
3000 D725 A90C            LDA #$C
3010 D727 D029            BNE MRGCTL  ;MERGE IN CTRL2
3020            ;
3030 D729 AD94CF  Q       LDA MODE
3040 D72C 3005            BMI PRDIR
3050 D72E A900    STCNT0  LDA #0
3060 D730 4C86D6          JMP SETCNT
3070 D733 ADA3CF  PRDIR   LDA DATA0+1  ;0=HOR PRINT, 3=VERT PRINT
3080 D736 2902            AND #2
3090 D738 0A              ASL A
3100 D739 0A              ASL A
3110 D73A 8DA3CF          STA DATA0+1
3120 D73D A907            LDA #7
3130 D73F D011            BNE MRGCTL  ;MERGE IN CTRL2
3140            ;
3150 D741 AD94CF  T       LDA MODE
3160 D744 10E8            BPL STCNT0  ;SET CHAR TYPE
3170 D746 ADA3CF  TYPE    LDA DATA0+1
3180 D749 2901            AND #1
3190 D74B 0A              ASL A
3200 D74C 0A              ASL A
3210 D74D 8DA3CF          STA DATA0+1
3220 D750 A90B            LDA #$B
3230 D752 2D02E2  MRGCTL  AND CTRL2
3240 D755 0DA3CF          ORA DATA0+1
3250 D758 8D02E2          STA CTRL2
3260 D75B 4C68D6          JMP SETMOD
3270            ;
3280 D75E AD94CF  W       LDA MODE    ;SEE FOR R/M/W-MODE
3290 D761 10CB            BPL STCNT0
3300 D763 ADA3CF  RMODW   LDA DATA0+1
3310 D766 6A              ROR A
3320 D767 6A              ROR A
3330 D768 2980            AND #$80
3340 D76A 8DA3CF          STA DATA0+1
3350 D76D A97F            LDA #$7F
3360 D76F D00F            BNE MRGPAG
3370            ;
3380 D771 AD94CF  Z       LDA MODE    ;Zp SET NEW PAGE
3390 D774 10B8            BPL STCNT0
3400 D776 ADA3CF  NPAGE   LDA DATA0+1
3410 D779 2903            AND #3
3420 D77B 8DA3CF          STA DATA0+1
3430 D77E A980            LDA #$80
3440 D780 2D93CF  MRGPAG  AND PAGEPT
3450 D783 0DA3CF          ORA DATA0+1
3460 D786 8D1AE2          STA PAGE
3470 D789 8D93CF          STA PAGEPT
3480 D78C 4C68D6          JMP SETMOD
3490            ;
3500            ;TWO DATA COMMANDS
3510            ;
3520 D78F AD94CF  D       LDA MODE
3530 D792 103D            BPL STCNT1  ;DRAW TO ABSOL.COORDS
3540 D794 A200            LDX #0      ;GET ABCIS (X)
3550 D796 20A1D7          JSR CALCX
3560 D799 A202            LDX #2      ;GET ORDINATE (Y)
3570 D79B 20A1D7          JSR CALCX
3580 D79E 4CD6D7          JMP JDRAW   ;DO ACTUAL DRAWING
3590 D7A1 BDA1CF  CALCX   LDA DATA1+1,X ;REPL.ABS.COORDS BY RELAT.ONE
3600 D7A4 38              SEC
3610 D7A5 FD09E2          SBC MSBX+1,X
3620 D7A8 9DA1CF          STA DATA1+1,X
3630 D7AB BDA0CF          LDA DATA1,X
3640 D7AE 297F            AND #$7F
3650 D7B0 FD08E2          SBC MSBX,X
3660 D7B3 9DA0CF          STA DATA1,X
3670 D7B6 B013            BCS POSRET
3680 D7B8 A900            LDA #0
3690 D7BA 38              SEC
3700 D7BB FDA1CF          SBC DATA1+1,X
3710 D7BE 9DA1CF          STA DATA1+1,X
3720 D7C1 A900            LDA #0
3730 D7C3 FDA0CF          SBC DATA1,X
3740 D7C6 0980            ORA #$80
3750 D7C8 9DA0CF          STA DATA1,X
3760 D7CB 60      POSRET  RTS
3770            ;
3780 D7CC AD94CF  J       LDA MODE    ;DO RELATIVE DRAWING
3790 D7CF 3005            BMI JDRAW
3800 D7D1 A901    STCNT1  LDA #1
3810 D7D3 4C86D6          JMP SETCNT
3820 D7D6 A203    JDRAW   LDX #3      ;GET COORDINATES ON STACK
3830 D7D8 BDA0CF  JDRW    LDA DATA1,X
3840 D7DB 48              PHA
3850 D7DC CA              DEX
3860 D7DD 10F9            BPL JDRW
3870 D7DF 20E5D7          JSR DRAWON  ;DO ACTUAL PLOTTING
3880 D7E2 4C53D6          JMP ENDMOD  ;AND PREP. FOR NEXT COMD
3890            ;
3900 D7E5 BA      DRAWON  TSX         ;GET STACKPOINTER
3910 D7E6 A940            LDA #$40    ;GET X-&Y-BITS ON RIGHT PLACE
3920 D7E8 1E0501          ASL STACK+5,X ;GET DIRECTION FOR PLOTTING
3930 D7EB 2A              ROL A
3940 D7EC 1E0301          ASL STACK+3,X
3950 D7EF 2A              ROL A
3960 D7F0 2A              ROL A
3970 D7F1 48              PHA           ;AND SAVE
3980 D7F2 5E0301          LSR STACK+3,X ;MSB BACK, LOSE SIGN
3990 D7F5 5E0501          LSR STACK+5,X
4000 D7F8 BD0401          LDA STACK+4,X ;LSB IN GDP
4010 D7FB 8D05E2          STA DELTAX
4020 D7FE BD0601          LDA STACK+6,X
4030 D801 8D07E2          STA DELTAY
4040 D804 A901            LDA #1
4050 D806 9D0401          STA STACK+4,X
4060 D809 9D0601          STA STACK+6,X
4070 D80C BC0301  TSTDIV  LDY STACK+3,X ;DIV VECTORS UNTIL
4080 D80F D005            BNE CONDIV    ;MSB'S ARE ZERO
4090 D811 BC0501          LDY STACK+5,X
4100 D814 F015            BEQ DONDIV
4110 D816 5E0301  CONDIV  LSR STACK+3,X ;DIVIDE
4120 D819 6E05E2          ROR DELTAX
4130 D81C 7E0401          ROR STACK+4,X
4140 D81F 5E0501          LSR STACK+5,X
4150 D822 6E07E2          ROR DELTAY
4160 D825 7E0601          ROR STACK+6,X ;SAVE FRACTION FOR ADJUST
4170 D828 0A              ASL A         ;(A) HOLDS TIMES TO DRAW
4180 D829 D0E1            BNE TSTDIV    ;LOOP BACK
4190 D82B A8      DONDIV  TAY           ; Y IS COUNTER
4200 D82C BD0301  NXDRAW  LDA STACK+3,X ;SUM FOR X FRACTION, IF A CAR
4210 D82F 18              CLC           ;OCCURES, AN ADJUSTMENT MUST
4220 D830 7D0401          ADC STACK+4,X ;PERFORMED IN THAT DIRECTION
4230 D833 9D0301          STA STACK+3,X
4240 D836 08              PHP           ;SAVE CARRY IF ONE
4250 D837 BD0501          LDA STACK+5,X ;SAME FOR Y FRACTION
4260 D83A 18              CLC
4270 D83B 7D0601          ADC STACK+6,X
4280 D83E 9D0501          STA STACK+5,X
4290 D841 BD0001          LDA STACK,X   ;GET DIR.BYTE
4300 D844 9002            BCC XADJ      ;NO ADJUSTMENT IN Y-DIR.
4310 D846 0988            ORA #$88      ;ONE DOT IN Y-DIR.
```

```
4320 D848 28      XADJ   PLP
4330 D849 9002           BCC TSTADJ
4340 D84B 09A0           ORA #$A0     ;ONE DOT IN X-DIR.
4350 D84D 48      TSTADJ PHA
4360 D84E 68             PLA
4370 D84F 1006           BPL NOADJ
4380 D851 8D00E2         STA CMD
4390 D854 20BAD0         JSR READY
4400 D857 BD0001  NOADJ  LDA STACK,X   ;FOR NORMAL VECTOR-PLOT
4410 D85A 0910           ORA #$10
4420 D85C 8D00E2         STA CMD
4430 D85F 20BAD0         JSR READY
4440 D862 88             DEY           ;MORE PLOTTING?
4450 D863 D0C7           BNE NXDRAW
4460 D865 68             PLA           ;ADJUST STACK,GET RETURN-
4470 D866 68             PLA           ;ADDRESS ON RIGHT PLACE,
4480 D867 9D0501         STA STACK+5,X
4490 D86A 68             PLA           ;AND RETURN
4500 D86B 9D0601         STA STACK+6,X
4510 D86E 68             PLA
4520 D86F 68             PLA
4530 D870 60             RTS
4540              ;
4550 D871 AD94CF  M      LDA MODE     ;MOVE ABS. NO DRAWING
4560 D874 1013           BPL ST1CNT
4570 D876 A203    MOVAB  LDX #3
4580 D878 BDA0CF  MOVE   LDA DATA1,X
4590 D87B 9D00E2         STA MSBX,X   ;ENTER COORDINATES
4600 D87E CA             DEX
4610 D87F 10F7           BPL MOVE
4620 D881 4C68D6         JMP SETMOD
4630              ;
4640 D884 AD94CF  RR     LDA MODE     ;MOVE REL. NO DRAWING
4650 D887 3005           BMI RELMOV
4660 D889 A901    ST1CNT LDA #1
4670 D88B 4C86D6         JMP SETCNT
4680 D88E A200    RELMOV LDX #0
4690 D890 20BDD8         JSR ADSUB
4700 D893 A202           LDX #2
4710 D895 20BDD8         JSR ADSUB
4720 D898 4C68D6         JMP SETMOD
4730 D89B BDA0CF  ADSUB  LDA DATA1,X
4740 D89E 3014           BMI SUB
4750 D8A0 BD00E2         LDA MSBX+1,X
4760 D8A3 18             CLC
4770 D8A4 7DA1CF         ADC DATA1+1,X
4780 D8A7 9D00E2         STA MSBX+1,X
4790 D8AA BD00E2         LDA MSBX,X
4800 D8AD 7DA0CF         ADC DATA1,X
4810 D8B0 9D00E2         STA MSBX,X
4820 D8B3 60             RTS
4830 D8B4 297F    SUB    AND #$7F
4840 D8B6 9DA0CF         STA DATA1,X ;MASK SIGN
4850 D8B9 BD00E2         LDA MSBX+1,X
4860 D8BC 38             SEC
4870 D8BD FDA1CF         SBC DATA1+1,X
4880 D8C0 9D00E2         STA MSBX+1,X
4890 D8C3 BD00E2         LDA MSBX,X
4900 D8C6 FDA0CF         SBC DATA1,X
4910 D8C9 9D00E2         STA MSBX,X
4920 D8CC 60             RTS
4930              ;
4940 D8CD AD94CF  SS     LDA MODE  ;Sx,y SIZE IN X- AND Y-DIRECTION
4950 D8D0 3005           BMI SIZE
4960 D8D2 A901    SET1   LDA #1
4970 D8D4 4C86D6         JMP SETCNT
4980 D8D7 ADA3CF  SIZE   LDA DATA0+1 ;PREPARE Y-SIZE
4990 D8DA 290F           AND #$0F
5000 D8DC 8DA3CF         STA DATA0+1
5010 D8DF ADA1CF         LDA DATA1+1 ;SHIFT FOR X SIZE
5020 D8E2 0A             ASL A
5030 D8E3 0A             ASL A
5040 D8E4 0A             ASL A
5050 D8E5 0A             ASL A
5060 D8E6 0DA3CF         ORA DATA0+1 ;GET MIX
5070 D8E9 8D03E2         STA CSIZE
5080 D8EC 4C68D6         JMP SETMOD
5090
5100 D8EF AD94CF  U      LDA MODE     ;Up,u USE PEN/ERASER UP/DOWN
5110 D8F2 10DE           BPL SET1
5120 D8F4 ADA3CF         LDA DATA0+1 ;p=1:PEN,p=0:ERASER
5130 D8F7 2901           AND #1
5140 D8F9 8DA3CF         STA DATA0+1
5150 D8FC ADA1CF         LDA DATA1+1 ;u=1:DOWN, u=0:UP
5160 D8FF 2901           AND #1
5170 D901 0A             ASL A
5180 D902 0DA3CF         ORA DATA0+1 ;MERGE AND SAVE
5190 D905 8DA3CF         STA DATA0+1
5200 D908 AD01E2         LDA CTRL1   ;GET THE NEW DATA IN CONTROL 1
5210 D90B 29FC           AND #$FC
5220 D90D 0DA3CF         ORA DATA0+1
5230 D910 8D01E2         STA CTRL1
5240 D913 4C68D6         JMP SETMOD
5250              ;
5260 D916 AD94CF  V      LDA MODE   ;Vx,y GET PIXEL WITH ADDRESS
5270 D919 10B7           BPL SET1   ;x,y IN LS-NIBBLE OF PIXBUF
5280 D91B A203           LDX #3
5290 D91D BD00E2  VIEW0  LDA MSBX,X
5300 D920 48             PHA        ;SAVE COORDS
5310 D921 CA             DEX
5320 D922 10F9           BPL VIEW0
5330 D924 A203           LDX #3     ;AND GET NEW ONES
5340 D926 BDA0CF  VIEW   LDA DATA1,X
5350 D929 9D00E2         STA MSBX,X
5360 D92C CA             DEX
5370 D92D 10F7           BPL VIEW
5380 D92F A90F           LDA #$F    ;ENTER MFREE MODE
5390 D931 8D00E2         STA CMD
5400 D934 20BAD0         JSR READY
5410 D937 AD18E2         LDA COLOR  ;PIXELS ARE IN b0...b3
5420 D93A 290F           AND #$F
5430 D93C 8D8ACF         STA PIXBUF
5440 D93F 4C2BDA         JMP ENDRW ;RESTORE COORDS
5450              ;
5460              ;THREE DATA COMMANDS
5470              ;
5480 D942 AD94CF  XX     LDA MODE     ;DRAW AN AXIS
5490 D945 3009           BMI AXIS
5500 D947 A902    STCNT2 LDA #2
5510 D949 4C86D6         JMP SETCNT
5520 D94C 18      DIRTAB .BYTE $18,$1E,$1A,$1C
5520 D94D 1E
5520 D94E 1A
5520 D94F 1C
5530 D950 ADA1CF  AXIS   LDA DATA1+1  ;UNIT OF LENGHT
5540 D953 8D05E2         STA DELTAX   ;IN VECTORS
5550 D956 8D07E2         STA DELTAY
5560 D959 AD9FCF         LDA DATA2+1 ;X OR Y AXIS?
5570 D95C 2901           AND #1       ;A(---AXIS
5580 D95E 8D9FCF         STA DATA2+1
5590 D961 0EA0CF         ASL DATA1    ;SIGN ---) (C)
5600 D964 2A             ROL A      ;(A)(---(C)
5610 D965 AA             TAX          ;THIS COMBINATION IS OFFSET IN
5620 D966 BC4CD9         LDY DIRTAB,X ;GET COMMAND IN Y
5630 D969 AEA3CF         LDX DATA0+1  ;GET TIMES IN X
5640 D96C 8C00E2  LOOP   STY CMD      ;PLOT ONE SEGMENT
5650 D96F 20BAD0         JSR READY
5660 D972 AD9FCF         LDA DATA2+1  ;GET DIRECTION FOR MARK
```

vervolg op bladzijde 63


SERVICE

SERVICE

printen zelf maken

■ U hebt hiervoor nodig: een spuitbus transparant-spray, een layout-pagina, een UV-lamp, natronloog en positief fotogevoelig printmateriaal (evt. zelf maken met positieve fotokopieerlak en printmateriaal).

■ De fotogevoelige koperzijde van het printmateriaal wordt met de transparant-spray goed nat gespoten.

■ De uit de layout-pagina geknipte koper-layout (in spiegelbeeld) legt u met de gedrukte zijde op het natte printmateriaal. Druk het papier licht aan en verwijder eventuele opgesloten luchtbelletjes door voorzichtig met een prop papier over de layout te strijken.

■ Het geheel kan nu met een UV-lamp belicht worden. De belichtingstijd is afhankelijk van de gebruikte UV-lamp, de afstand hiervan tot het printmateriaal en het fotogevoelige materiaal.

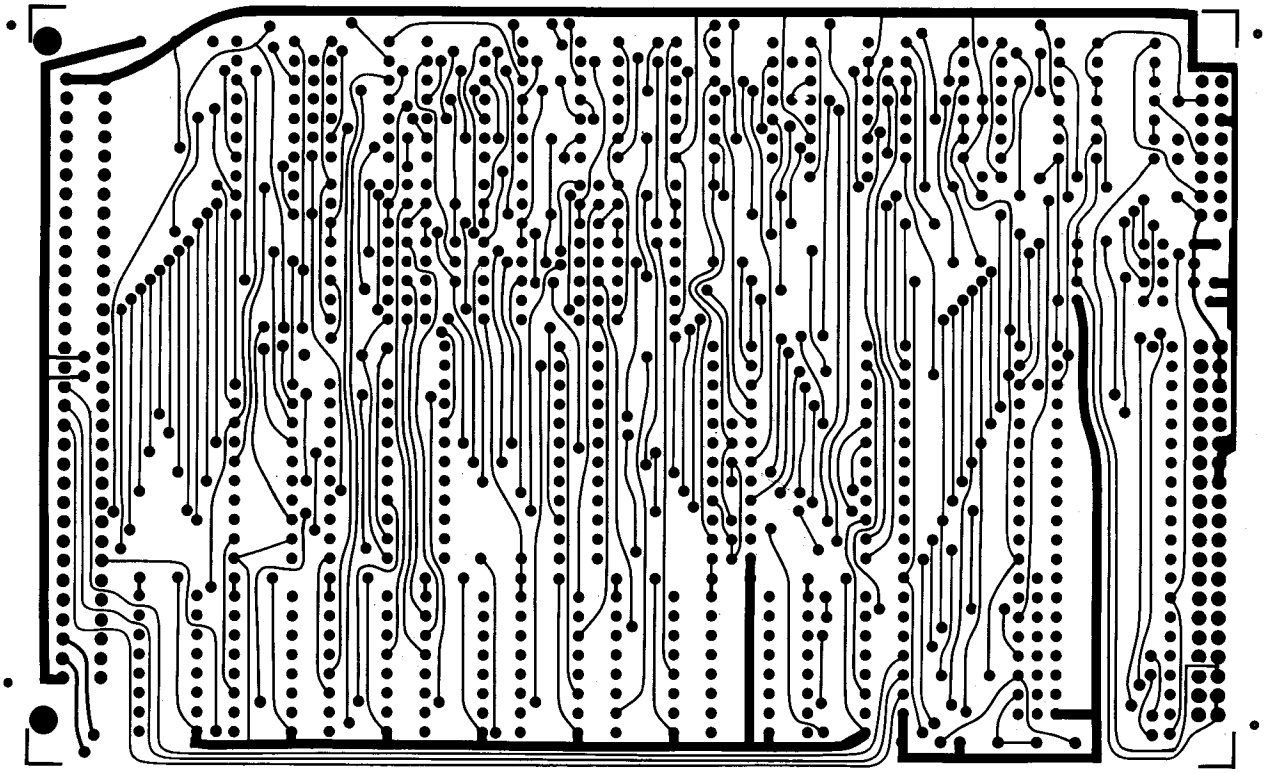
■ Na het belichten verwijdert u het layoutvel (nog meerdere malen bruikbaar) en spoelt u het printmateriaal onder stromend water schoon.

■ Na het ontwikkelen van de fotogevoelige laag in natronloog (ongeveer 9 gram in 1 liter water oplossen) kan de print in ijzer-3-chloride (500 gram $FeCl_3$ in 1 liter water) geëtst worden. Spoel daarna de print grondig schoon (en ook uw handen!), verwijder met wat staalwol het fotogevoelige laagje van de kopersporen en boor de gaatjes.

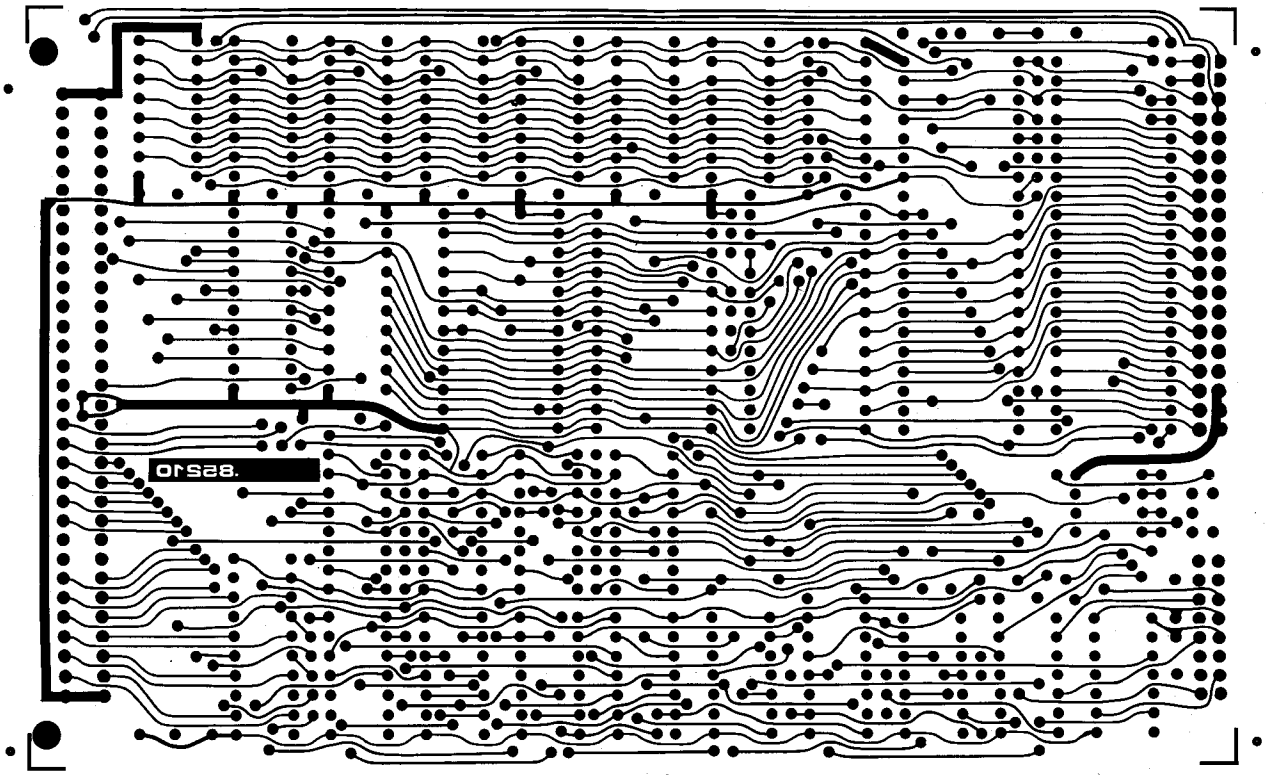
■ Spaar ons milieu en gooi geen uitgewerkte chemicaliën of resten ervan achteloos in de gootsteen, maar informeer in uw gemeente naar een hiervoor bestemd depot!

EC68: CPU/DRAM-kaart

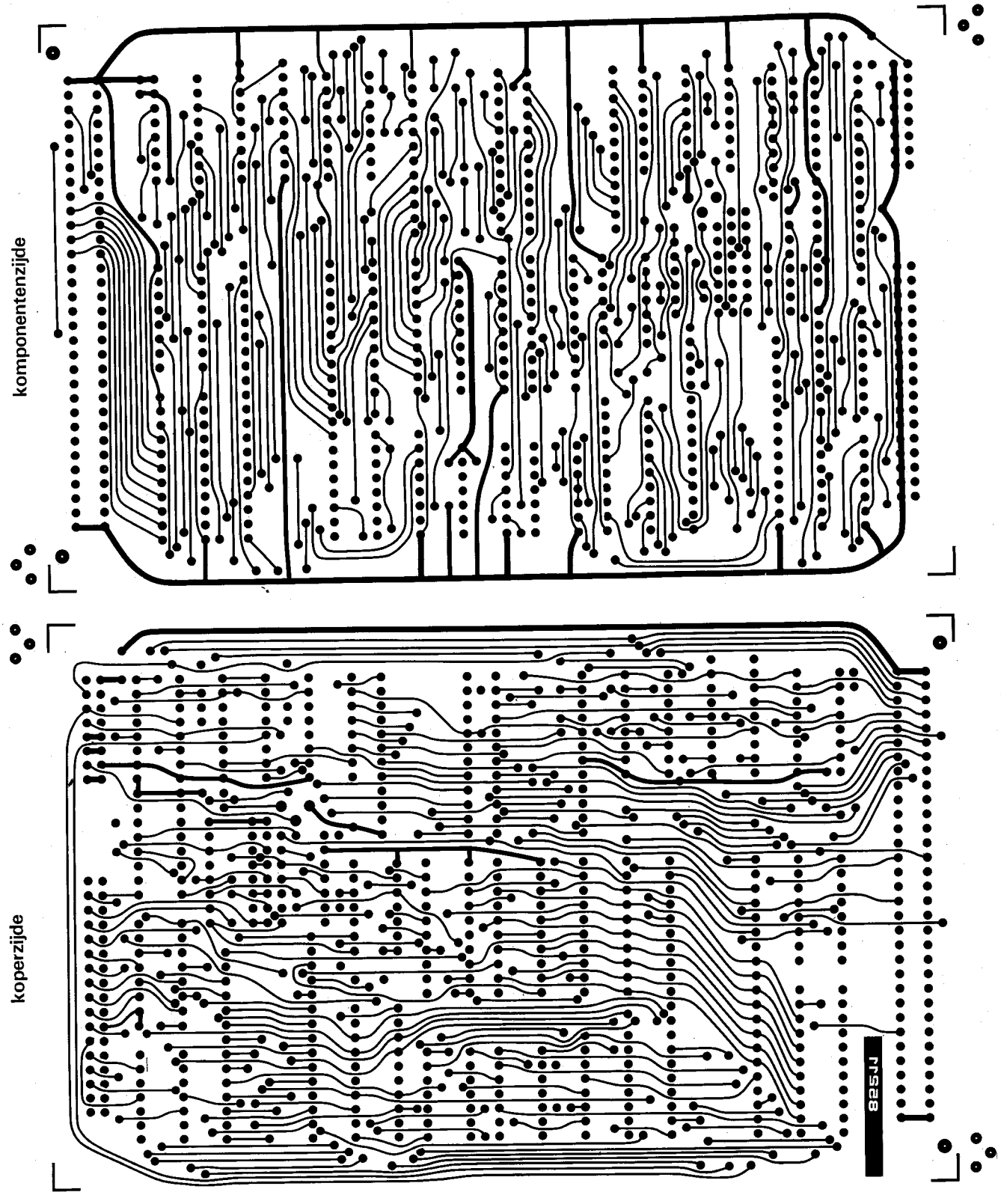
komponentenzijde

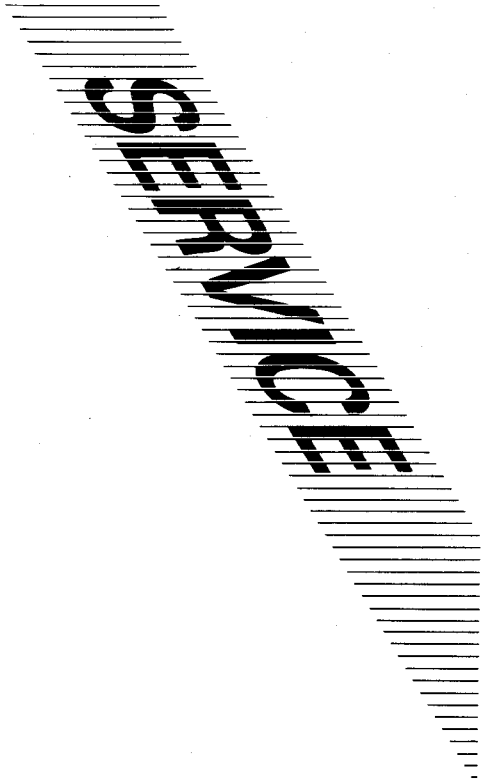


koperzijde



EC68: CRT/FDC-kaart




SERVICE

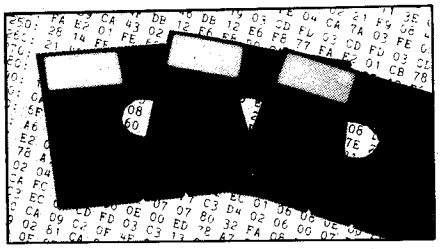
```
5670 D975 F006          BEQ STRIPY   ;X-AXIS HAS MARKS IN Y DIRECTI
5680 D977 2086D9        JSR XSTRIP
5690 D97A 18            CLC
5700 D97B 9003          BCC TSTCNT   ;ALLWAYS
5710 D97D 209AD9  STRIPY JSR YSTRIP
5720 D980 CA      TSTCNT DEX
5730 D981 D0E9          BNE LOOP
5740 D983 4C53D6        JMP ENDMOD   ;MULT. CMD IS POSSIBLE
5750 D986 2091D9  XSTRIP JSR INCX
5760 D989 A9D6          LDA #$D6
5770 D98B 8D00E2        STA CMD
5780 D98E 20BAD0        JSR READY
5790 D991 EE09E2  INCX  INC LSBX
5800 D994 D003          BNE DONX
5810 D996 EE08E2        INC MSBX
5820 D999 60      DONX  RTS
5830 D99A 20A5D9  YSTRIP JSR INCY
5840 D99D A9D4          LDA #$D4
5850 D99F 8D00E2        STA CMD
5860 D9A2 20BAD0        JSR READY
5870 D9A5 EE0BE2  INCY  INC LSBY
5880 D9A8 D003          BNE DONY
5890 D9AA EE0AE2        INC MSBY
5900 D9AD 60      DONY  RTS
5910              ;
5920              ;SPECIAL COMMANDS
5930              ;
5940 D9AE AD94CF  O     LDA MODE    ;Os,r,t
5950 D9B1 1094    ST2CNT BPL ST2CNT2 ;ENTER THREE DATA
5960 D9B3 AD9FCF        LDA DATA2+1 ;SECTORS?
5970 D9B6 D015          BNE SECTOR
5980 D9B8 A9FF          LDA #$FF
5990 D9BA 8D9FCF        STA DATA2+1 ;CIRCLE OR RING
6000 D9BD ADA3CF        LDA DATA0+1 ;THICKNESS?
6010 D9C0 F002          BEQ PLOTB   ;IF CIRCLE
6020 D9C2 A901    PLOTA LDA #1
6030 D9C4 8DA0CF  PLOTB STA DATA1   ;SET UP SHAPE
6040 D9C7 20B7DA        JSR CIRC    ;AND DO PLOTTING
6050 D9CA 4C68D6        JMP SETMOD  ;RETURN
6060 D9CD ADA3CF  SECTOR LDA DATA0+1
6070 D9D0 CDA1CF        CMP DATA1+1 ;COMPLETELY FILLED?
6080 D9D3 D0ED          BNE PLOTA   ;IF SECTORED RING
6090 D9D5 A902          LDA #2      ;IF FULL SECTORED
6100 D9D7 D0EB          BNE PLOTB
6110              ;
6120 D9D9 AD94CF  G     LDA MODE    ;Gn,x,y,...
6130 D9DC 10D3          BPL ST2CNT
6140 D9DE A203          LDX #3      ;SAVE CUR. ORIG ON STACK
6150 D9E0 BD08E2  SAVSTK LDA MSBX,X
6160 D9E3 48            PHA
6170 D9E4 CA            DEX
6180 D9E5 10F9          BPL SAVSTK
6190 D9E7 2C9ECF        BIT DATA2   ;FILLED?
6200 D9EA 08            PHP         ;SAVE RESULT
6210 D9EB AD9FCF        LDA DATA2+1 ;TRIANGLE OR RECTANGLE?
6220 D9EE 2901          AND #1
6230 D9F0 F047          BEQ TRIANG
6240 D9F2 2088DA        JSR XDRW    ;DRAW RECTANGLE
6250 D9F5 20A4DA        JSR YDRW
6260 D9F8 2088DA        JSR XDRW
6270 D9FB 20A4DA        JSR YDRW
6280 D9FE 28            PLP         ;DONE?
6290 D9FF 102A          BPL ENDRW   ;YES
6300 DA01 A200    FILOOP LDX #0
6310 DA03 205FDA        JSR DECVEC
6320 DA06 F023          BEQ ENDRW
6330 DA08 2CA0CF  INCDEC BIT DATA1  ;MOVE ONE STEP AWAY
6340 DA0B 300B          BMI DECAX   ;FROM CURR.ORIGIN
6350 DA0D EE09E2        INC LSBX
6360 DA10 D013          BNE DONMAT
6370 DA12 EE08E2        INC MSBX
6380 DA15 4C25DA        JMP DONMAT
6390 DA18 CE09E2  DECAX DEC LSBX
6400 DA1B AD09E2        LDA LSBX
6410 DA1E C9FF          CMP #$FF
6420 DA20 D003          BNE DONMAT
6430 DA22 CE08E2        DEC MSBX
6440 DA25 20A4DA  DONMAT JSR YDRW
6450 DA28 4C01DA        JMP FILOOP  :LOOP BACK
6460 DA2B A200    ENDRW LDX #0
6470 DA2D 68      RESTK PLA         ;RESTORE ORIGIN
6480 DA2E 9D08E2        STA MSBX,X
6490 DA31 E8            INX
6500 DA32 E004          CPX #4
6510 DA34 D0F7          BNE RESTK
6520 DA36 4C53D6        JMP ENDMOD
6530              ;
6540 DA39 2088DA  TRIANG JSR XDRW   ;DRAW A TRIANGLE
6550 DA3C 2086DA        JSR XYDRW
6560 DA3F 20A4DA        JSR YDRW
6570 DA42 28            PLP         ;DONE?
6580 DA43 10E6          BPL ENDRW
6590 DA45 2088DA        JSR XDRW    ;FILL TRIANGLE
6600 DA48 A202    NXFILL LDX #2     ;OFFSET FOR Y
6610 DA4A 205FDA        JSR DECVEC ;TEST FOR 0,DECR.VECTR & TEST
6620 DA4D F0DC          BEQ ENDRW
6630 DA4F 2086DA        JSR XYDRW
6640 DA52 A200          LDX #0      ;OFFSET FOR X
6650 DA54 205FDA        JSR DECVEC ;TEST
6660 DA57 F0D2          BEQ ENDRW
6670 DA59 2086DA        JSR XYDRW
6680 DA5C 4C48DA        JMP NXFILL ;LOOP BACK
6690              ;
6700 DA5F 2075DA  DECVEC JSR TSTVEC  ;TEST FOR ZERO
6710 DA62 F01B          BEQ RETST   ;RETURN IF ZERO
6720 DA64 BDA1CF        LDA DATA1+1,X ;öVECö=öVEC-1ö
6730 DA67 38            SEC
6740 DA68 E901          SBC #1
6750 DA6A 9DA1CF        STA DATA1+1,X
6760 DA6D BDA0CF        LDA DATA1,X
6770 DA70 E900          SBC #0
6780 DA72 9DA0CF        STA DATA1,X
6790 DA75 BDA0CF  TSTVEC LDA DATA1,X
6800 DA78 297F          AND #$7F
6810 DA7A D003          BNE RETST
6820 DA7C BDA1CF        LDA DATA1+1,X
6830 DA7F 60      RETST RTS           ;TEST AGAIN & RETURN
6840              ;
6850 DA80 A900    XDRW  LDA #0        ;DRAW IN X-DIRECTION
6860 DA82 48            PHA
6870 DA83 48            PHA
6880 DA84 F00D          BEQ XDRW0
6890              ;
6900 DA86 ADA3CF  XYDRW LDA DATA0+1 ;DRAW A VECTOR
6910 DA89 48            PHA         ;AND REVERSE DIRECTION
6920 DA8A ADA2CF        LDA DATA0   ;FOR NEXT DRAW
6930 DA8D 48            PHA
6940 DA8E 4980          EOR #$80
6950 DA90 8DA2CF        STA DATA0
6960 DA93 ADA1CF  XDRW0 LDA DATA1+1
6970 DA96 48            PHA
6980 DA97 ADA0CF        LDA DATA1
6990 DA9A 48            PHA
7000 DA9B 4980          EOR #$80
7010 DA9D 8DA0CF        STA DATA1
7020 DAA0 20E5D7  YDRW0 JSR DRAWON
7030 DAA3 60            RTS
7040
```

```
7050 DAA4 ADA3CF  YDRW   LDA DATA0+1 ;DRAW IN Y-DIRECTION
7060 DAA7 48             PHA
7070 DAA8 ADA2CF         LDA DATA0
7080 DAAB 48             PHA
7090 DAAC 4980           EOR #$80
7100 DAAE 8DA2CF         STA DATA0
7110 DAB1 A900           LDA #0
7120 DAB3 48             PHA
7130 DAB4 48             PHA
7140 DAB5 F0E9           BEQ YDRW0
7150              ;
7160 DAB7=        CIRC   =*
```

```
.EX
'07 TRACK(S)
A*LO SAMMCI_ _ _ CIR

A*RE AS


.AA_

   10             ;################################
   20             ;#      V I D E O               #
   30             ;#      ---------               #
   40             ;#VIDEO-UTILITIES WITH GDP9366  #
   50             ;# WRITTEN BY P.LAVIGNE  NOV'83 #
   60             ;#                              #
   70             ;#  PART III :  CIRCLE          #
   80             ;#                              #
   90             ;################################
  100             ;
  110 E200             * = $E200      ;GDP REGS
  120 E200=        STATUS = *
  130 E200=        CMD    = *
  140 E201=        CTRL1  = *+1
  150 E202=        CTRL2  = *+2
  160 E203=        CSIZE  = *+3
  170 E205=        DELTAX = *+5
  180 E207=        DELTAY = *+7
  190 E208=        MSBX   = *+8
  200 E209=        LSBX   = *+9
  210 E20A=        MSBY   = *+10
  220 E20B=        LSBY   = *+11
  230             ;
  240 E218             * = $E218      ;HARDWARE REGS
  250 E218=        COLOR  = *
  260 E219=        SCROLL = *+1
  270             ;
  280 CF80             * = $CF80      ;SOFTWARE REGS
  290 CF84=        SAVX   = *+4
  300 CF85=        MIRXH  = *+5
  310 CF86=        MIRXL  = *+6
  320 CF87=        MIRYH  = *+7
  330 CF88=        MIRYL  = *+8
  340 CF92=        SCRLPT = *+18
  350 CF9E=        DATA2  = *+30
  360 CFA0=        DATA1  = *+32
  370 CFA2=        DATA0  = *+34
  380             ;
  390             ;PARAMETERS IN 'PCIRCL'
  400             ;
  410 CFA4=        XL     = *+36
  420 CFA5=        YL     = *+37
  430 CFA6=        SL     = *+38
  440 CFA7=        SH     = *+39
  450             ;
  460 CF86=        X0L    =MIRXL
  470 CF85=        X0H    =MIRXH
  480 CF88=        Y0L    =MIRYL
  490 CF87=        Y0H    =MIRYH      ;COORDS OF CENTRE
  500 CFA1=        RAD    =DATA1+1    ;RADIUS
  510 CFA0=        SHP    =DATA1      ;SHAPE:0=CIRC,1=RING,2=OCTANTS
  520 CFA3=        THCK   =DATA0+1    ;THICKNESS OF RING
  530 CF9F=        OCT    =DATA2+1    ;EVRY OCTANT IS REPRES BY BIT
  540             ;
  550 0100=        STACK  = $100
  560             ;
  570 D000             * = $D000
  580             ;
  590 D08A=        READY  = *+$8A
  600             ;
  610 DAB7             * = *+$AB7
  620             ;
  630 DAB7 4C03DC  PCIRCL JMP PLCIRC
  640             ;
  650             ;GENERAL ADD/SUSTR.ROUTINE :ADSB16
  660             ;(C) ON ENTRY GIVES ADD(C=0) OR SUB(C=1)
  670             ;STACK MANAGEMENT:
  680             ;ON ENTRY
  690             ; SP--->S   -)  (A)
  700             ;       S+1 -) RETADD
  710             ;       S+2 -) RETADD
  720             ;       S+3 -) P1-H
  730             ;       S+4 -) P1-L
  740             ;       S+5 -) P2-H
  750             ;       S+6 -) P2-L
  760             ;BEFORE RETURN
  770             ; SP--->S+4 -)  ( )
  780             ;       S+5 -) RETADD
  790             ;       S+6 -) RETADD
  800             ;
  810             ;"P1"+/-"P2" IS PERFORMED
  820             ;
  830 DABA 8E84CF  ADSB16 STX SAVX       ;SAVE (X)
  840 DABD BA             TSX            ;SP--)(X)
  850 DABE BD0401         LDA STACK+4,X  ;GET LSB OF "P1"
  860 DAC1 B00D           BCS SUB16      ;IF (C)=1:SUBSTRACT
  870 DAC3 7D0601         ADC STACK+6,X
  880 DAC6 A8             TAY            ;LSB OF RESULT IN (Y)
  890 DAC7 BD0301         LDA STACK+3,X  ;GET MSB
  900 DACA 7D0501         ADC STACK+5,X  ;ADD "P2"-MSB
  910 DACD E8             INX            ;FOR BRANCH
  920 DACE D00B           BNE ADJST      ;ALLWAYS
  930 DAD0 FD0601  SUB16  SBC STACK+6,X  ;SUBSTRACT "P2"-LSB
  940 DAD3 A8             TAY
  950 DAD4 BD0301         LDA STACK+3,X
  960 DAD7 FD0501         SBC STACK+5,X  ;MSB OF RESULT IN (A)
  970 DADA E8             INX
  980 DADB 48      ADJST  PHA            ;SAVE (A) ON STACK
  990 DADC BD0001         LDA STACK,X    ;REPLACE RETURN ADDRESS
 1000 DADF 9D0401         STA STACK+4,X
 1010 DAE2 BD0101         LDA STACK+1,X
 1020 DAE5 9D0501         STA STACK+5,X
 1030 DAE8 68             PLA
 1040 DAE9 E8             INX            ;ADJUST STACK
 1050 DAEA E8             INX
 1060 DAEB E8             INX
 1070 DAEC 9A             TXS            ;S=(S+1)+3
 1080 DAED AE84CF         LDX SAVX       ;RESTORE (X)
 1090 DAF0 60             RTS
 1100            ;
 1110            ;AD(SB)X(Y)CO IS A SET OF 4 SUBROUTINES
```

```
1120             ;ON ENTRY (A) HOLDS XL OR YL
1130             ;
1140 DAF1 18     ADXCO  CLC          ;RESET (C) FOR ADDITION
1150 DAF2 24            .BYTE $24    ;FOR JUMP OVER 'SEC'
1160 DAF3 38     SBXCO  SEC          ;SET (C) FOR SUBSTRACTION
1170 DAF4 A200          LDX #0       ;POINTER IN "TABLE"
1180 DAF6 F005          BEQ COMON
1190 DAF8 18     ADYCO  CLC
1200 DAF9 24            .BYTE $24
1210 DAFA 38     SBYCO  SEC
1220 DAFB A202          LDX #2
1230 DAFD 48     COMON  PHA          ;XL OR YL ON STACK
1240 DAFE A900          LDA #0
1250 DB00 48            PHA          ;MSB IS ZERO
1260 DB01 BD86CF        LDA X0L,X    ;LSB OF X0 (OR Y0)
1270 DB04 48            PHA          ;ON STACK
1280 DB05 BD85CF        LDA X0H,X    ;MSB
1290 DB08 48            PHA
1300 DB09 20BADA        JSR ADSB16   ;ADD OR SUB DEPS ON (C)
1310 DB0C 9D08E2        STA MSBX,X   ;STORE IN PROPER GDP-REGISTER
1320 DB0F 98            TYA
1330 DB10 9D09E2        STA LSBX,X
1340 DB13 60            RTS
1350             ;
1360 DB14 EEA6CF ADDIT  INC SL       ;DO S=S+1 FOR HORNS ALGORITHM
1370 DB17 D003          BNE STOP
1380 DB19 EEA7CF        INC SH
1390 DB1C 60     STOP   RTS
1400             ;
1410             ;LIGHT LIGHTS A DOT IF IN CIRCLE OR RING MODES
1420             ;      OR WRITES A SEGMENT IF IN DISK MODE
1430             ;ON INPUT:- (Y) HOLDS VECTORCODE FOR DISKSEGMENT
1440             ;         - DELTAX&Y AND X,Y MUST HAVE BEEN
1450             ;           SETUP IN GDP BEFORE CALLING 'LIGHT'
1460             ;
1470 DB1D 8C00E2 LIGHT  STY CMD
1480 DB20 4C8AD0        JMP READY
1490             ;
1500             ;PLOT66 IS USED IN ALL PLOTMODES
1510             ;       8 DOTS ARE PLOTTED ACCORDING TO
1520             ;       THE 8 BITS IN 'OCT'
1530             ;REMARK: Y-DIR. IS DIV BY 2 FOR 256 X 512
1540             ;      SCREEN
1550             ;
1560             ;
1570 DB23 A900   PLOT66 LDA #0
1580 DB25 ACA0CF        LDY SHP
1590 DB28 C002          CPY #2
1600 DB2A D007          BNE DOTS
1610 DB2C ADA4CF        LDA XL       ;DO X-Y
1620 DB2F 38            SEC
1630 DB30 EDA5CF        SBC YL
1640 DB33 8D05E2 DOTS   STA DELTAX
1650 DB36 4A            LSR A        ;DIV 2
1660 DB37 8D07E2        STA DELTAY
1670 DB3A A901          LDA #1       ;MASK b0
1680 DB3C 2C9FCF        BIT OCT
1690 DB3F F012          BEQ OCT2     ;IF ZERO NO LIGHT
1700 DB41 ADA4CF        LDA XL
1710 DB44 20F1DA        JSR ADXCO    ;X=X0+X
1720 DB47 ADA5CF        LDA YL
1730 DB4A 4A            LSR A
1740 DB4B 20F8DA        JSR ADYCO    ;Y=Y0+Y
1750 DB4E A016          LDY #$16     ;SET PROPER COMMAND
1760 DB50 201DDB        JSR LIGHT
1770 DB53 A902   OCT2   LDA #2       ;MASK b1
1780 DB55 2C9FCF        BIT OCT
1790 DB58 F012          BEQ OCT3
1800 DB5A ADA5CF        LDA YL
1810 DB5D 20F1DA        JSR ADXCO    ;X=X0+Y
1820 DB60 ADA4CF        LDA XL
1830 DB63 4A            LSR A
1840 DB64 20F8DA        JSR ADYCO    ;Y=Y0+X
1850 DB67 A014          LDY #$14
1860 DB69 201DDB        JSR LIGHT
1870 DB6C A904   OCT3   LDA #4
1880 DB6E 2C9FCF        BIT OCT
1890 DB71 F012          BEQ OCT4
1900 DB73 ADA5CF        LDA YL
1910 DB76 20F3DA        JSR SBXCO    ;X=X0-Y
1920 DB79 ADA4CF        LDA XL
1930 DB7C 4A            LSR A
1940 DB7D 20F8DA        JSR ADYCO    ;Y=Y0+X
1950 DB80 A014          LDY #$14
1960 DB82 201DDB        JSR LIGHT
1970 DB85 A908   OCT4   LDA #8
1980 DB87 2C9FCF        BIT OCT
1990 DB8A F012          BEQ OCT5
2000 DB8C ADA4CF        LDA XL
2010 DB8F 20F3DA        JSR SBXCO    ;X=X0-X
2020 DB92 ADA5CF        LDA YL
2030 DB95 4A            LSR A
2040 DB96 20F8DA        JSR ADYCO    ;Y=Y0+Y
2050 DB99 A010          LDY #$10
2060 DB9B 201DDB        JSR LIGHT
2070 DB9E A910   OCT5   LDA #$10
2080 DBA0 2C9FCF        BIT OCT
2090 DBA3 F012          BEQ OCT6
2100 DBA5 ADA4CF        LDA XL
2110 DBA8 20F3DA        JSR SBXCO    ;X=X0-X
2120 DBAB ADA5CF        LDA YL
2130 DBAE 4A            LSR A
2140 DBAF 20FADA        JSR SBYCO    ;Y=Y0-Y
2150 DBB2 A010          LDY #$10
2160 DBB4 201DDB        JSR LIGHT
2170 DBB7 A920   OCT6   LDA #$20     ;MASK b5
2180 DBB9 2C9FCF        BIT OCT
2190 DBBC F012          BEQ OCT7
2200 DBBE ADA5CF        LDA YL
2210 DBC1 20F3DA        JSR SBXCO    ;X=X0-Y
2220 DBC4 ADA4CF        LDA XL
2230 DBC7 4A            LSR A
2240 DBC8 20FADA        JSR SBYCO    ;Y=Y0-X
2250 DBCB A012          LDY #$12
2260 DBCD 201DDB        JSR LIGHT
2270 DBD0 A940   OCT7   LDA #$40     ;MASK b6
2280 DBD2 2C9FCF        BIT OCT
2290 DBD5 F012          BEQ OCT8
2300 DBD7 ADA5CF        LDA YL
2310 DBDA 20F1DA        JSR ADXCO    ;X=X0-Y
2320 DBDD ADA4CF        LDA XL
2330 DBE0 4A            LSR A
2340 DBE1 20FADA        JSR SBYCO    ;Y=Y0-X
2350 DBE4 A012          LDY #$12
2360 DBE6 201DDB        JSR LIGHT
2370 DBE9 A980   OCT8   LDA #$80     ;MASK b7
2380 DBEB 2C9FCF        BIT OCT
2390 DBEE D001          BNE SETIT
2400 DBF0 60            RTS
2410 DBF1 ADA4CF SETIT  LDA XL
2420 DBF4 20F1DA        JSR ADXCO    ;X=X0+X
2430 DBF7 ADA5CF        LDA YL
2440 DBFA 4A            LSR A
2450 DBFB 20FADA        JSR SBYCO    ;Y=Y0-Y
2460 DBFE A016          LDY #$16
2470 DC00 4C1DDB        JMP LIGHT
2480             ;
2490 DC03 ADA1CF PLCIRC LDA RAD      ;PLOT CIRCLE,RING OR DISK
```

```
2500 DC06 8DA4CF          STA XL        ;X=R
2510 DC09 8DA6CF          STA SL        ;S=R
2520 DC0C A900            LDA #0        ;ZERO MSB'S & Y
2530 DC0E 8DA5CF          STA YL
2540 DC11 8DA7CF          STA SH
2550 DC14 8D19E2          STA SCROLL  ;RESET SCREEN
2560 DC17 8D92CF          STA SCRLPT  ;SAVE IN RAM
2570 DC1A ACA0CF          LDY SHP       ;SEE SHAPE: 00=CIRCLE, 01=RIN
2580 DC1D F014            BEQ CIRCLE
2590 DC1F C001            CPY #1
2600 DC21 F00D            BEQ RING
2610 DC23 38              SEC
2620 DC24 EDA6CF          SBC SL
2630 DC27 8DA6CF          STA SL
2640 DC2A CEA7CF          DEC SH        ;SH=$FF, SL=-SL
2650 DC2D 4C0CDD          JMP OUTDEF    ;PLOT & TESTXY
2660 DC30 2023DB  RING    JSR PLOT66    ;PLOT(X,Y) WITH X,Y=R,0
2670 DC33 ADA4CF  CIRCLE  LDA XL        ;S=S-2*X
2680 DC36 0A              ASL A
2690 DC37 48              PHA
2700 DC38 A900            LDA #0
2710 DC3A 2A              ROL A
2720 DC3B 48              PHA
2730 DC3C ADA6CF          LDA SL
2740 DC3F 48              PHA
2750 DC40 ADA7CF          LDA SH
2760 DC43 48              PHA
2770 DC44 38              SEC           ;SET (C) FOR SUB
2780 DC45 20BADA          JSR ADSB16
2790 DC48 8DA7CF          STA SH
2800 DC4B 8CA6CF          STY SL        ;S=S-2*X
2810 DC4E ACA5CF  TESTXY  LDY YL
2820 DC51 A902            LDA #2
2830 DC53 CDA0CF          CMP SHP
2840 DC56 D005            BNE DOTEST    ;IF NOT DISK: (Y)=Y
2850 DC58 18              CLC           ;ELSE (Y)=Y+2
2860 DC59 6DA5CF          ADC YL
2870 DC5C A8              TAY
2880 DC5D CCA4CF  DOTEST  CPY XL
2890 DC60 9003            BCC AFTER     ;IF LESS
2900 DC62 4C12DD          JMP ENDP      ;IF GREATER
2910 DC65 ADA0CF  AFTER   LDA SHP
2920 DC68 D003            BNE NOCIRC
2930 DC6A 2023DB          JSR PLOT66  ;IF CIRCLE PLOT DOTS
2940 DC6D ADA6CF  NOCIRC  LDA SL      ;S=(2*Y+S)+1
2950 DC70 48              PHA
2960 DC71 ADA7CF          LDA SH
2970 DC74 48              PHA
2980 DC75 ADA5CF          LDA YL
2990 DC78 0A              ASL A       ;(A)=2*Y
3000 DC79 48              PHA
3010 DC7A A900            LDA #0
3020 DC7C 2A              ROL A       ;IF CARY, (A)=01
3030 DC7D 48              PHA
3040 DC7E 18              CLC         ;PREP FOR ADDITION
3050 DC7F 20BADA          JSR ADSB16
3060 DC82 8DA7CF          STA SH      ;RESULT IN S
3070 DC85 8CA6CF          STY SL
3080 DC88 2014DB          JSR ADDIT   ;DO S=S+1
3090 DC8B EEA5CF          INC YL      ;DO Y=Y+1
3100 DC8E ADA7CF          LDA SH      ;SEE FOR S<0
3110 DC91 302C            BMI TSFORM
3120 DC93 ADA4CF          LDA XL      ;NOW DO S=S-2*X+2
3130 DC96 0A              ASL A       ;2*X
3140 DC97 48              PHA
3150 DC98 A900            LDA #0
3160 DC9A 2A              ROL A       ;IF CARY?
3170 DC9B 48              PHA
3180 DC9C ADA6CF          LDA SL
3190 DC9F 48              PHA
3200 DCA0 ADA7CF          LDA SH
3210 DCA3 48              PHA
3220 DCA4 38              SEC         ;PREPARE FOR SUBSTRACTION
3230 DCA5 20BADA          JSR ADSB16
3240 DCA8 8DA7CF          STA SH
3250 DCAB 8CA6CF          STY SL
3260 DCAE 2014DB          JSR ADDIT   ;DO S=S+2
3270 DCB1 2014DB          JSR ADDIT
3280 DCB4 ADA4CF          LDA XL      ;DO X=X-1
3290 DCB7 38              SEC
3300 DCB8 E901            SBC #1
3310 DCBA 8DA4CF          STA XL
3320 DCBD 9053            BCC ENDP    ;IF X<0
3330 DCBF ADA0CF  TSFORM  LDA SHP
3340 DCC2 F04B            BEQ JUMP    ;IF CIRCLE
3350 DCC4 C901            CMP #1
3360 DCC6 D044            BNE OUTDEF  ;IF DISK
3370 DCC8 ADA4CF          LDA XL      ;COMPUTE S-2*X+2*R+1
3380 DCCB 0A              ASL A
3390 DCCC 48              PHA
3400 DCCD A900            LDA #0
3410 DCCF 2A              ROL A
3420 DCD0 48              PHA
3430 DCD1 ADA6CF          LDA SL
3440 DCD4 48              PHA
3450 DCD5 ADA7CF          LDA SH
3460 DCD8 48              PHA
3470 DCD9 38              SEC
3480 DCDA 20BADA          JSR ADSB16    ;S-2*X
3490 DCDD AA              TAX           ;SAVE (A) IN (X)
3500 DCDE 98              TYA           ;(Y) TO (A) :LSB IN (A)
3510 DCDF 48              PHA
3520 DCE0 8A              TXA           ;MSB IN (A)
3530 DCE1 48              PHA
3540 DCE2 ADA1CF          LDA RAD
3550 DCE5 0A              ASL A         ;2*R
3560 DCE6 48              PHA
3570 DCE7 A900            LDA #0
3580 DCE9 2A              ROL A
3590 DCEA 48              PHA
3600 DCEB 18              CLC
3610 DCEC 20BADA          JSR ADSB16    ;NOW DO 2*R+(S-2*X)
3620 DCEF C8              INY           ;ADD 1 TI RESULT
3630 DCF0 D003            BNE NOC
3640 DCF2 18              CLC
3650 DCF3 6901            ADC #1        ;INCR. MSB IF Y WAS $80
3660 DCF5 2980    NOC     AND #$80    ;SEE MSb OF MSB
3670 DCF7 C980            CMP #$80      ;IF MSBIT=1 THEN
                                        SKIP PLOT(X
3680 DCF9 D011            BNE OUTDEF
3690 DCFB 38              SEC
3700 DCFC ADA4CF          LDA XL
3710 DCFF E901            SBC #1
3720 DD01 8DA4CF          STA XL
3730 DD04 B003            BCS NEG       ;IF X<0 THEN SKIP TOO
3740 DD06 2023DB          JSR PLOT66
3750 DD09 EEA4CF  NEG     INC XL        ;RESTORE X
3760 DD0C 2023DB  OUTDEF  JSR PLOT66
3770 DD0F 4C4EDC  JUMP    JMP TESTXY
3780 DD12 ADA0CF  ENDP    LDA SHP
3790 DD15 C901            CMP #1
3800 DD17 D005            BNE END
3810 DD19 ADA3CF          LDA THCK      ;RING PLOTTED?
3820 DD1C D001            BNE MORE
3830 DD1E 60      END     RTS
3840 DD1F CEA3CF  MORE    DEC THCK      ;NEXT CIRCLE IN RING
3850 DD22 CEA1CF          DEC RAD
3860 DD25 4C03DC          JMP PLCIRC
3870              ;
3880 DD28=        ELECTR = *
```



BASIC-plus

BASIC-plus, super-DOS en supermonitor voor de Octopus 65

We hebben het geweten. Er kwamen nogal wat telefoontjes bij ons binnen nadat de eerste Elektuur Computing met de Octopus 65 op de markt was verschenen. Bijna iedereen wilde weten wat de voordelen zijn van een zelfbouwcomputer ten opzichte van een kant-en-klaar exemplaar. Anders gezegd, loont het vandaag de dag nog wel de moeite om zelf een computer in elkaar te "steken"? Ons antwoord daarop: ja! Zonder meer. Want anders als het bij een fabrieksmachine het geval is, kan een zelfgebouwde computer helemaal aangepast worden aan de persoonlijke behoeftes en eisen. Zowel voor wat de betreft de software als de harware.

In het nu volgende stukje laten we zien hoe professionele programmeurs de BASIC-interpreter, het disk operating system (DOS) en de monitor hebben "getuned".

Een aparte beschrijving van de oorspronkelijke DOS is hierbij niet noodzakelijk. Weliswaar is er links en rechts in het "inwendige" ervan nogal wat gewijzigd, maar de basiskommando's zijn vrijwel identiek gebleven met DOS 3.3. We bespreken hier dan ook alleen maar de diverse wijzigingen. Maar eerst even dit: voor de Octopus 65 bestaat nu ook een nieuwe krachtige assembler, de "ASS114". Deze werkt interaktief met de diskette en wordt met een uitvoerige handleiding geleverd. We zullen deze assembler in één van de komende EC's uitvoerig bespreken. Terug nu naar BASIC-plus.

"Tuning": BASIC-plus-interpreter

Labels in plaats van regelnummers

De Microsoft-BASIC-interpreter van de Octopus 65 tast een BASIC-programma regel voor regel af. Die regels hebben, zoals u weet, elk een eigen nummer. Normaliter gebruikt de interpreter een regelnummer als sprongmarkering. Bij het maken van langere programma's zijn echter veel hulpprogramma's noodzakelijk, wat het nadeel met zich mee brengt dat veel regelnummers moeten worden "onthouden". Vooral wanneer een BASIC-programma opnieuw genummerd wordt (renumber), is er van een goed overzicht echt geen sprake meer.

Gelukkig biedt BASIC de mogelijkheid om in plaats van de regelnummers zogenaamde markers (labels) te gebruiken. Figuur 1 toont hier hoe gemakkelijk, overzichtelijk en komfortabel een BASIC-programma dan kan worden gemaakt. De "gereserveerde" kommando's "GOTO", "GOSUB" en "IF...THEN GOTO", brengen de interpreter naar de regels die door "LAB" gekenmerkt zijn. De naam van de markering mag maximaal een regel lang zijn. De label-search-routine werkt hierbij zo snel, dat alleen bij meer dan 50 labels een teruggang van de verwerkingssnelheid merkbaar is. BASIC-plus staat op de schijf met de nieuwe DOS 4.0. Waar wat op de schijf en in het geheugen van de Octopus 65 staat, toont tabel 1.

CLEARGOSUB

Het kommando CLEARGOSUB wist de terugsprongparameter van een GOSUB-instruktie op de stack. Daardoor is het mogelijk om vanuit een hulpprogramma naar een willekeurig regelnummer te springen.

INKY X$

INKY X$ wacht op een ingave van het toetsenbord. De waarde van de ingedrukte toets wordt dan in de variabele X$ opgeslagen.

RESTORE regelnummer

Achter het kommando RESTORE kan een regelnummer staan. Daardoor kan RESTORE ook voor een bepaald datablok worden gebruikt.

Nieuwe residente BASIC-plus-kommando's

De zojuist genoemde BASIC-kommando's vormen het nieuwe bestanddeel van de BASIC-interpreter. In tegenstelling tot DOS 3.3. bevindt de inhoudsopgave zich op "track 7" en beslaat drie "pages". Er kunnen maximaal 48 files worden ingegeven, waarbij een file-naam maximaal 14 karakters lang mag zijn.

Er zullen slechts weinige "computeristen" zijn die niet al eens met een rekenmachientje of "met de hand" een hexadecimaal getal omgezet hebben in een decimaal getal (of andersom natuurlijk). Immers, bij PEEK- en POKE-kommando's zijn dergelijke omzettingen onvermijdelijk. En hoe vaak gebeurt het hierbij niet dat de computer alleen maar met behulp van de resettoets terug in de "werkelijkheid" kan worden gehaald, omdat het omgezette getal weer eens een keertje verkeerd was berekend. Daar komt nu verandering in. $NNNN zet namelijk een hexadecimaal getal om in zijn decimale en binaire "broertjes". #NNNNN berekent het hexadecimale adres en het binaire bitpatroon.

<BY> BASIC-plus uitschakelen

Het hulpprogramma voor BASIC-plus bestaat uit een toolkit, die alleen "aktief" is wanneer er iets ingegeven wordt, dus als er een programma wordt gemaakt. Het

tabel 1. Zo staat de software van BASIC-plus, DOS 4.0 en EDMO op de schijf. Ook de laadvektoren zijn overzichtelijk weergegeven.

Track	Sektor	Pages	Vektor	Opmerking
0	1	8	$2200	DOS (deel 1)
1	1	8	$2A00	DOS (deel 2)
2	1	8	$0200	BASIC (deel 1)
3	1	8	$0A00	BASIC (deel 2)
4	1	8	$1200	BASIC (deel 3)
5	1	8	$1A00	BASIC (deel 4)
6	1	1	$2200	BASIC (deel 5)
6	2	1	$20C4	BASIC (deel 6)
6	3	6	$3200	Input/Output-programma
7	1	1	$2E79	Directory (deel 1)
7	2	1	$2E79	Directory (deel 2)
7	3	1	$2E79	Directory (deel 3)
7	4	1	$2E79	Get/Put-Overlays
7	5	1	$0000	Page 0
7	6	2	$3800	BASIC (deel 7)
8	1	8	BASIC	BEXEC*
9	1	8	$D000	BASICplus (deel 1)
10	1	8	$D800	BASICplus (deel 2)
11	1	8	$BC00	EDMO (deel 1)
12	1	8	$C400	EDMO (deel 2)
13	1	4	$CC00	EDMO (deel 3)
14	1	8	BASIC	Track-Zero-Utility
15	1	8	BASIC	Copy-Utility

hulpprogramma staat in het geheugen van $D000 tot $DFFF. Normaliter kan de Octopus 65 kwa lengte vrijwel elk programma in zijn geheugen onderbrengen. Betreft het echter een zeer lang programma, dan kan het noodzakelijk zijn dat het zojuist genoemde bereik in gebruik moet worden genomen. Dat gaat als volgt in zijn werk: de kommando's van de toolkit worden door het omleiden van de interpreter-error-routine aangesproken. Met BY kan deze omleiding uitgeschakeld worden, zodat de error-routine weer normaal funktioneert. Het geheugenbereik $D000...$DFFF kan nu door BASIC gebruikt worden. Om de Octopus er op te attenderen dat dit geheugenbereik weer beschikbaar is, hoeft men alleen maar POKE 133,215 in te geven.

<CH> tekst in BASIC-programma's veranderen
Na het ingeven van <CH> geeft BASIC "Search:" op het scherm, waarna een tekst kan worden ingegeven. Antwoordt men met <cr>, dan wordt het kommando afgebroken. Na het ingeven van een karakterreeks komt BASIC terug met de melding "Replace:" (ook nu kan nog met <cr> onderbroken worden). Na een input wordt het BASIC-programma in het geheugen doorzocht, waarna alle regelnummers op het scherm worden weergegeven waarin een karakterreeks door een andere is vervangen.
Het kommando CH is heel handig wanneer bijvoorbeeld "GOSUB 500" vervangen moet worden door "GOTO 500". Bij lange BASIC-regels kan het echter voorkomen dat door dit kommando een nieuwe regel wordt gekreëerd die meer dan 71 karakters bevat. BASIC geeft in dat geval een waarschuwing en zet een tekentje achter de desbetreffende regel. Het is ook mogelijk om achter het CH-kommando zogenaamde "schakelaars" te zetten. Deze kunnen dan door het ingeven van een letter geaktiveerd worden. De CH-routine vraagt nu vooraf of een karakterreeks in een regel daadwerkelijk moet worden verwisseld door een andere.

De schakelaar "V"
Wil men de variabele "i" omzetten in bijvoorbeeld de variabele "a", dan zal de CH-routine, wanneer de schakelaar niet is gezet, de karakters "di" omzetten in "da". De schakelaar "V" zorgt er voor dat vóór het verwisselen eerst gekeken wordt of de gevonden "i" ook werkelijk een echte variabele is.

De schakelaar "T"
Deze schakelaar werkt in principe hetzelfde als de "V"-schakelaar, maar dan met het verschil dat tekst nu alleen maar binnen een karakterreeks of in een REM-regel wordt veranderd.

De schakelaar "A"
Schrijft men bijvoorbeeld <CH A> met daarachter eventueel nog V (voor variabele) of T (voor tekst), dan gebeurt het volgende:
De input na "Search:" en "Replace:" blijft hetzelfde. BASIC zoekt ook nu net zo lang in het in het geheugen opgeslagen programma totdat de gewenste karakterreeks is gevonden. Is de desbetreffende regel gevonden, dan wordt deze op het scherm weergegeven. Geeft men nu cr in, dan vindt er geen uitwisseling plaats en wordt de CR-routine onderbroken. Wordt er echter een spatie ingegeven, dan vindt er eveneens geen uitwisseling plaats, maar zoekt BASIC verder naar een volgende karakterreeks. Alleen als er met "A" wordt geantwoord, zal de karakterreeks door een andere worden vervangen, waarna de veranderde BASIC-regel op het scherm verschijnt.

<DE> DELETE-kommando
Het wissen van komplete programmablokken is, en daar zult u het ongetwijfeld met ons eens zijn, een vrij omslachtig karweitje wanneer men dat regel voor regel moet doen. Bij BASIC-plus kan dat gelukkig ook anders. Door het ingeven van het kommando <DE/eerste regelnummer/laatste regelnummer> kan men namelijk meerdere regels gelijktijdig wissen. Het kommando <DE/120/300> bijvoorbeeld, zal alle regels tussen 120 en 300 wissen. Een spatie op een bepaalde plaats in het kommando heeft de volgende uitwerkingen: <DE//300> wist alle programmaregels tot en met regel 300. <DE 300//> wist daarentegen alles wat na programmaregel 300 staat. Het schuine streepje "/" vormt hierbij een deel van de syntax. Een verkeerde ingave zal dan ook een SN-error-melding tot gevolg zal hebben.

<DI> DIRECTORY-kommando
Het kommando DI geeft een inhoudsoverzicht van de schijf op het scherm. In één oogopslag kunt u dan zien welk loopwerk in gebruik is, de namen van alle opgeslagen bestanden, de lengte van de bestanden en zowel het eerste als het laatste track-nummer. Met P kan deze informatie naar de printer worden gestuurd.

<DR> DRIVE-SELECT-kommando
Het ingeven van <DR A> heeft precies hetzelfde resultaat als wanneer men het

Figuur 1. Bij de "getunede" interpreter van de Octopus 65 is het mogelijk om regelnummers te vervangen door sprongmarkeringen (labels), waardoor gestruktueerd in BASIC kan worden geprogrammeerd. Een supersnelle label-search-routine zorgt er voor dat de verwerkingssnelheid ook bij lange labels hoog blijft.

Figuur 2. Bij het booten van het systeem wordt ook een komfortabele debugger — EDMO — in het geheugen geladen. Vooral beginnende computerenthousiasten kunnen met behulp van EDMO snel in de "geheimen" van het programmeren in machinetaal doordringen.

1

```
LIST

 100 GOSUB "EC3"
 110 GOSUB "ELEKTUUR": PRINT
 120 X=0
 130 LAB "LOOP": GOSUB "LUS"
 140 IF X>5 THEN GOSUB "EINDE": END
 150 GOTO "LUS"
 160 :
 170 LAB "EC3": PRINT "De nieuwe Elektuur Computing": RETURN
 180 :
 190 LAB "ELEKTUUR": PRINT "Uitgeversmij. Elektuur B.V.": RETURN
 200 :
 210 LAB "LUS"
 220 PRINT "Lus", X; "Keer doorlopen"
 230 X=X+1
 240 RETURN
 250 :
 260 LAB "EINDE": PRINT: PRINT "Einde van het programma"
 270 RETURN

Ok
RUN
De nieuwe Elektuur Computing
Uitgeversmij. Elektuur B.V.
Lus      0 keer doorlopen
Lus      1 keer doorlopen
Lus      2 keer doorlopen
Lus      3 keer doorlopen
Lus      4 keer doorlopen
Lus      5 keer doorlopen
Einde van het programma

Ok
```

langere kommando <DISK!"SE A"> ingeeft. Met het eerstgenoemde kommando kan dus met veel minder typewerk een loopwerk worden geselekteerd. Bij het schrijven van een programma kan men daardoor snel en komfortabel nagaan op welk loopwerk de diverse bestanden zijn opgeslagen.

<GE> GET-kommando

DOS 3.3 beschikt over de mogelijkheid om bestanden zowel in het geheugen te schrijven, als deze te lezen (memory-input/output). In het normale geval betekent dit dat een BASIC-bestand in ASCII-formaat slechts van $3A79...7FFD lang kan zijn. Met <LIST#5> kan een BASIC-programma echter als ASCII-file in het geheugen van de Octopus 65 worden gezet. Het beginadres van deze ASCII-file ligt dan bij $8000. Nadat het LIST#5-kommando is afgewerkt, moet altijd "$" ingegeven worden (dit stuurt een EOT-teken voor de verdere verwerking). Het kommando <LIST#5,500-600> zet alleen maar een gedeelte van het BASIC-programma in het geheugen.
Geeft men nu het kommando <NEW> in, dan is voor BASIC het programmageheugen helemaal leeg. Met het tekentje "X" wordt het ASCII-bestand (vanaf $8000) weer in het programmageheugen van de BASIC-interpreter gelezen. De interpreter leest nu net zolang het ASCII-bestand in RAM totdat deze een " " tegenkomt. Goed, tot zover het principe.
Wat te doen echter wanneer een BASIC-programma boven $8000 uitkomt? Natuurlijk kan men binnen bepaalde grenzen nog wat met het MEM-kommando in de Kernel doen, maar helemaal probleemloos gaat dat ook niet. Want hoe snel is er wel niet een fout gemaakt bij het zetten van de parameters? Met BASIC-plus kunnen al deze problemen elegant opgelost worden. Het manipuleren gebeurt bij BASIC-plus namelijk niet in het geheugen, maar rechtstreeks op de schijf. Wie dus een komfortabele softwarebibliotheek wil opbouwen, kan dat heel eenvoudig doen.
Veel programmeurs hebben een aantal hulpprogramma's op schijf opgeslagen, waarmee ze volgens het "bouwdoosprincipe" hun programma's vervaardigen. Hoe dat in de praktijk in zijn werk gaat, kunnen we het beste aan de hand van een voorbeeld uit de doeken doen: Om het overzichtelijk te houden noemen we de voorbeeldprogramma's gemakshalve PROG1 tot en met PROG5. Om te beginnen moeten we de genoemde bestanden in de directory van een schijf zetten. Dat doen we met behulp van het kommando <MA> (we komen straks nog even op dit kommando terug). Om de eerste bestandsnaam in de inhoudsopgave van de schijf te zetten, moet <MA "PROG1,1> worden ingegeven. Het BASIC-programma dat de gewenste hulproutine bevat zal nu in het geheugen worden gezet. In het voorbeeld vinden we deze hulproutine overigens vanaf regel 800 tot en met regel 900 van het BASIC-programma. Vervolgens geeft men <PU "PROG1"800—900> in, waarna de opgegeven programmaregels in ASCII-formaat op de schijf worden gezet. Dezelfde aanpak geldt voor de overige 4 hulpprogramma's (PROG2...PROG5). Heeft men een BASIC-programma in het geheugen staan waarin men PROG1 wil toevoegen, dan hoeft men alleen maar het kommando <GE "PROG1"> in te geven. In ons voorbeeld worden de regels 800...900 aan het programma toegevoegd. Maar wat als deze regels al door het BASIC-programma zelf gebruikt worden? Niet zo'n probleem. Met het kommando RE (dat we later nog zullen bespreken) kan men namelijk het programma dat in het geheugen is opgeslagen van andere regelnummers voorzien. Het maakt hierbij niets uit of het een BASIC-programma is of een hulproutine. In principe gaat dat als volgt in zijn werk: Met het kommando <LO "programmanaam"> wordt een BASIC-programma in het geheugen geladen, waarbij met <GE> "programmanaam" meerdere hulpprogramma's in een BASIC-programma kunnen worden toegevoegd. Hierbij hoeft men overigens

2 Korte beschrijving Echo (Vers. DOS 4.8)

Oproepen vanuit BASIC met 'EM'
Oproepen vanuit Kernel met 'GO AC00' of 'GO AC03'
(bij AC03 kan vanuit Echo met 'E' naar de opgeroepen plaats worden teruggesprongen.)

Cursor-sturing :
Control A = Scroll down
E = Move right
H = Cursor left
I = Cursor right
J = Cursor down
K = Cursor up
L = Clear screen
Q = Scroll up
T = Home
W = Move left
X = Erase rest of screen
Y = Erase rest of line

Transfer :
T AAAA EEEE AAAA

Relocate :
N AAAA EEEE AAAA

Search :
H AAAA EEEE NN NN ..

Branch-berekening :
O AAAA AAAA

Checksum :
& AAAA EEEE

Rekenen :
+ NNNN NNNN
- NNNN NNNN

Memory-dump :
M AAAA EEEE of M AAAA-

Disassembling :
D AAAA EEEE of D AAAA-

Stoppen van uitgave met spatietoets, verder met willekeurige toets.

Assembler :
A NNNN LDA #$00
A NNNN ...

Compare :
C AAAA EEEE AAAA

Fill RAM :
F AAAA EEEE NN

Jump to :
G AAAA

Omrekening :
$ NNNN = Hex Asc Dec Bin
# NNNN = Dec Asc Hex Bin
% 1010 = Hex Asc Dec Bin

Veranderingen in geheugen:

Memorydump / Disassembling
Cursor naar de desbetreffende geheugenplaats sturen, nieuwe waarde ingeven en op <RETURN> drukken.

Assembler
Cursor naar de desbetreffende mnemonics-plaats sturen, nieuwe opcode ingeven en op <RETURN> drukken.

Let op bij editing !
Altijd worden de nieuw berekende adressen gegeven.

geen disk-buffer aan te leggen, omdat BASIC-plus dat automatisch berekent.

GO-kommando
Dit kommando is vrijwel identiek aan het GO-kommando in versie 3.3. Het enige verschil is dat "Page 0" en "Page 1" niet meer worden gesaved. Met het GO-kommando kan men zowel decimaal als hexadecimaal naar een bepaald adres springen: <GO $NNNN> en <GO NNNNN>. Het "$"-teken geeft hierbij aan dat het om een hexadecimaal adres gaat.

<KI> KILL-kommando
Het KILL-kommando wist bestanden die in de directory van een schijf zijn opgenomen. Mocht een bestand niet gevonden worden, dan meldt de computer dat met de mededeling "File not found".

<LO> LOAD-kommando
Bij het ingeven van <LO "programmanaam"> wordt het BASIC-programma met die naam in het geheugen van de computer gezet. Staat het programma niet op de schijf, dan wordt dat ook hier weer gemeld met "File not found".

<MA> MAKE-kommando
Bij BASIC-plus is het mogelijk om een file in het inhoudsoverzicht van de disk op te nemen, zonder dat een BASIC-programma op de schijf wordt gekopieerd. Daartoe moet op de computer het volgende worden ingegeven: <MA "bestandsnaam",N>. De N staat hierbij voor het aantal sporen dat gereserveerd moet worden. Terugmeldingen zoals "File in use" of "No room on disk' zijn gelijk aan die bij het SA-kommando.

<NN> NEWNAME-kommando
Een andere mogelijkheid om de directory op een schijf vanuit BASIC te bewerken, biedt het NN-kommando. Met dit kommando is het namelijk mogelijk om de diverse bestanden van een nieuwe naam te voorzien. Daartoe een voorbeeld: <NN "Name A" TO "Name B"> doopt het met "Name A" opgegeven bestand om in "Name B". Mocht een bepaald bestand niet op de schijf aanwezig zijn, dan zal er een foutmelding worden gegeven. Bevat de schijf al een bestand met die naam, dan zal dat worden gemeld met de mededeling "File in use".

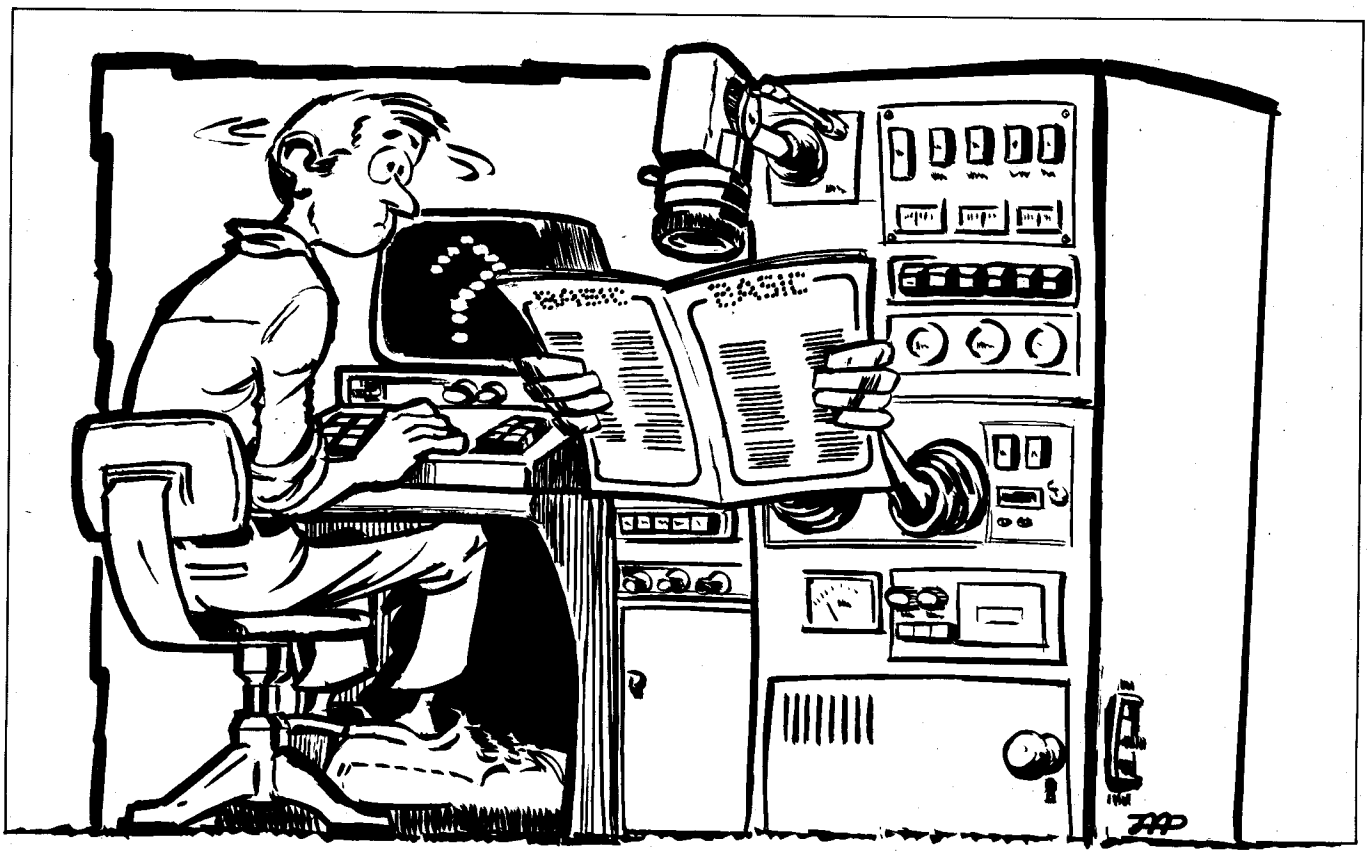
<PA> PACK-kommando
Wie vaak langere BASIC-programma's schrijft, kan in de problemen komen door een te klein werkgeheugen. Het PA-kommando reduceert daarom de omvang van het BASIC-programma, door alle spaties en alle kommentaren in REM-regels te wissen. REM-kommando's aan het begin van een regel staan worden zekerheidshalve ongemoeid gelaten.

<RE> RENUMBER-kommando
Bij de meeste "RENUMBER"-programma's is het niet mogelijk om binnen een BASIC-programma gericht regelnummers te veranderen. Het in BASIC-plus ingebouwde RE-kommando kent dat nadeel niet. Het ingeven van <RE/eerste regel/laatste regel/nieuwe eerste regel/regelafstand> bewerkstelligt een hernummering binnen de gestelde parameters.

<RE/500/999/700/2/>
In dit voorbeeld worden alle regels die zich binnen het bereik 500...999 bevinden, vanaf regel 700 met een regelafstand van 2 genummerd. Zet men voor "eerste regel" geen nummer (dus: "//"), dan begint de nummering bij de eerste programmaregel. Hetzelfde geldt wanneer voor de "laatste regel" geen nummer wordt gegeven ("//"). De programmanummering eindigt dan bij de laatste programmaregel.
<RE /////>
Dit kommando hernummert het hele programma in stappen van 10 en begint bij regel 100.

<SA> SAVE-kommando
Het SAVE-kommando zet een zich in het werkgeheugen bevindend programma op schijf. Bevat de schijf reeds een bestand met die naam, dan zal het nieuwe bestand over het oude worden geschreven, waarna de melding "File updated" op het scherm verschijnt.
Staat het te saven programma nog niet op schijf, dan wordt eerst gekeken of er nog genoeg plaats op de disk beschikbaar is. Is dat het geval, dan zet de computer de nieuwe bestandsnaam in de directory (inklusief het aantal benodigde tracks) waarna de melding "Saved: bestandsnaam" op het scherm verschijnt. Mocht er niet meer genoeg plaats op de schijf beschikbaar zijn, dan wordt de melding "No room on disk' gegeven. In dat geval kan men wat ruimte kreëren door met behulp van het KI-kommando een oud en niet meer benodigd bestand te wissen.

<SE> SEARCH-kommando
Door het ingeven van <SE> kan men in het hele programmabereik statements opzoeken, echter niet in REM-regels en binnen karakterreeksen. Aan het SE-kommando kunnen schakelaars worden gekoppeld.
Schakelaar "T": Er wordt alleen maar in het tekstgedeelte gezocht. Hiertoe behoren ook REM-regels en ASCII-karakterketens.
Schakelaar "V": Zoals ook bij het CH-kommando het geval is, wordt hier alleen



maar naar variabelen gezocht.
De uitgave gebeurt normaliter door het weergeven van de regelnummers waarin de gezochte statements staan. Aktiveert men echter de schakelaar "Z", dan geeft de computer de hele regel weer.
Maakt men schakelaar "S" aktief, dan verschijnen op het scherm de regelnummers waarin zich de gezochte BASIC-statements bevinden, inklusief de statements zelf. Voor de duidelijkheid geven we hier nog een tweetal voorbeelden.
Wil men bijvoorbeeld weten waar overal in het programma het kommando "GOSUB 1000" staat, dan moet eerst <SEZ> worden ingegeven. BASIC-plus antwoordt dan met "Search:", waarna "GOSUB 1000" moet worden ingetypt. Op het scherm verschijnen nu alle programmaregels waarin het kommando "GOSUB 1000" staat.
Door het ingeven van <SETS> (zoek in tekst en geef statements) kan men nagaan in welke programmaregels bepaalde karakterreeksen staan. BASIC-plus meldt zich dan met "Search", waarna bijvoorbeeld de karakterreeks "file" kan worden ingegeven. Op het scherm zijn nu alle regelnummers en statements te zien die de karakterreeks "file" bevatten, of - als het een REM-regel betreft — de ASCII-reeks "file".

<TC> TRACE-kommando
Wil men een BASIC-programma stap voor stap op fouten kontroleren, dan hoeft men alleen maar het kommando TC in te geven. Eventuele fouten in het programma kunnen nu snel en eenvoudig gelokaliseerd worden. Het uitschakelen van de tracer gebeurt simpelweg door het nogmaals ingeven van het TC-kommando.

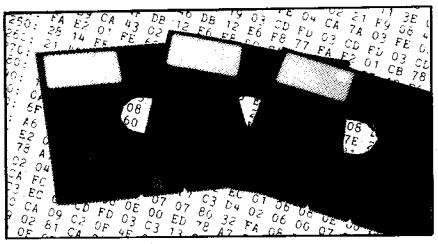
<TR> TRACKS
Dit kommando geeft het aantal tracks dat een bepaald BASIC-programma beslaat. Het is identiek aan de kommando's "EXIT" en "REENTER BASIC".

EDMO, de nieuwe monitor

Naast BASIC-plus bevat de systeemschijf nog het super-monitorprogramma EDMO. Het in het geheugen laden van EDMO gebeurt automatisch tijdens het booten. Hoe eenvoudig met deze monitor kan worden gewerkt, blijkt wel uit figuur 2, waarin we alle kommando's op een rijtje hebben gezet.

Enkele afsluitende opmerkingen

BASIC-plus en EDMO vormen beide op hun gebied een goedkoop maar krachtig hulpmiddel bij het vervaardigen van BASIC- en machinetaalprogramma's. De prijs van de diskette (inklusief dokumentatie) ligt om en nabij de 100 gulden. Beide softwarepakketten hebben hun eigen programma's voor het sturen van het beeldscherm, terwijl de input/output-routines van de Octopus 65 in de EPROM niet meer worden opgeroepen.
BASIC-plus werd geschreven door Werner Bascik. EDMO kwam voor rekening van Fred Aubert. Beide behoren tot een groep Berlijnse "computerfreaks", die zich naast 6502-processoren ook nog met CP/M- en 68000-machines bezig houden.



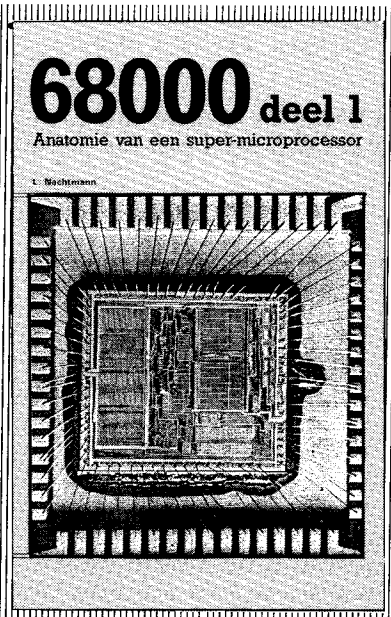
de tracer van de Octopus 65

In het nu volgende stukje willen we nog wat verder ingaan op de tracer van de Octopus 65. Immers, wie al veel in machinetaal heeft geschreven, zal zo'n hulpje bij het opsporen van softwarefouten gegarandeerd niet meer willen missen.
De tracer van de Octopus 65 bestaat uit twee gedeeltes: een stukje hardware en een stukje software. Het hardwaregedeelte met het bijbehorende printje hebben we reeds in onze eerste Elektuur Computing gepubliceerd. In dit verhaal gaan we dan ook alleen maar in op de software van de tracer en hoe men er mee om moet springen.
Om te beginnen moet de tracer-print met behulp van het daarop aanwezige toetsje op "scherp" worden gezet, hetgeen door een oplichtende LED aangegeven wordt. Is dat gebeurd, dan moet nog het startadres ingegeven worden (gevolgd door een spatie) van waaruit de computer een machinetaalprogramma stap voor stap moet gaan bewerken. Met het indrukken van de T-toets start men het tracer-programma. Op het scherm verschijnt dan de melding: "↑L,↑P,↑S" (↑L=List, ↑P=Page, ↑S=Step). Geeft men het List-kommando, dan verschijnen op het scherm alle instrukties die de computer uitvoert. Ook de inhoud van alle CPU-registers en de bovenste data op de stack kunnen op het scherm worden afgelezen. De computer blijft zolang in de trace-mode totdat een "breakpoint" wordt bereikt, of totdat het kommando "↑C" wordt ingegeven. Hetzelfde geldt voor de kommando's "↑P" en "↑S". Bij een "↑P"-kommando blijft de Octopus 65 in de trace-mode totdat het beeldscherm helemaal vol is. Door nog eens "↑P" in te geven, kan het tracen worden voortgezet. Het "↑S"-kommando heeft in principe dezelfde uitwerking, alleen wordt nu na elke instruktie die de computer uitvoert aan de "noodrem" getrokken.
Zoals we reeds in ons eerste nummer vertelden, geeft de monitor van de Octopus 65 twee verschillende terugmeldingen: ":" en "-Octopus 65-". Bij de eerstgenoemde terugmelding is de stackpointer niet gezet, hetgeen bij de tweede melding wel het geval is (op $01FF). Verder is het vaak belangrijk dat men de tracer kan verlaten om bijvoorbeeld een geheugenplaats of een register van de CPU te modificeren. Daartoe hoeft men alleen maar het kommando ".!" in te geven en de tracer gaat terug naar het monitorprogramma. Op het scherm wordt dit aangegeven met de melding ":". Nu kan alles naar wens gemodificeerd worden. Moet met de gemodificeerde data verder gewerkt worden, dan hoeft men alleen maar de toetsen "P" (nieuwe programmateller) en "T" (tracer weer ingeschakeld) in te drukken. Ook de drie tracer-kommando's "↑L, ↑P, ↑S" kunnen nu weer worden gebruikt. Maar wat gebeurt er eigenlijk allemaal in de computer nadat het "T"-kommando is ingegeven? Wel, allereerst wordt door het tracer-programma de NMI-vektor als het ware "omgebogen". Het wijst naar adres $FE38, waar het tracer-kommando in de monitor-EPROM door de software geëmuleerd wordt. Leest de processor een opcode, dan wordt de SYNC-lijn aktief, waardoor via de hardware van de tracer een "NMI" (Non Maskable Interrupt) wordt veroorzaakt. Door de NMI-vektor gaat de processor naar het EPROM-adres $FE38, waar het statusregister van de processor en de programmateller van de stack worden gehaald en gesaved. Ook worden de interne CPU-registers door de processor gekopieerd en in een gereserveerd bereik op "page 0" gezet. Vervolgens worden de register-inhouden op het scherm weergegeven, waarna de Octopus de volgende instruktie gaat bewerken. Hierdoor kan men gemakkelijk zien onder welke startposities de processor met een nieuwe instruktie begint.
Zijn alle beschreven werkzaamheden door de computer uitgevoerd, dan wacht de tracer totdat weer een van de drie kommando's wordt ingegeven. Geeft men "↑L", "↑P" of "↑S" in, dan brengt een spronginstruktie de processor naar het adres $F8AA (label: RUN + 4), waar de nieuwe programmateller en het statusregister op de stack worden gezet. Ook de CPU-registers worden weer op hun momentele waarde gezet. Via een "RTI"-instruktie (Return from Interrupt) begint de processor dan aan de volgende instruktie die uitgevoerd moet worden. Door het ingeven van de monitor-kommando's ".P" of ".S" kan de uitvoer van de tracer naar een parallelle of seriële printer-interface worden gestuurd.

de 68.000

en zijn smallere broertje, de 68008 spelen de hoofdrol in:

Anatomie van een super-microprocessor



o.a. nu al in:

Sinclair QL
Atari 520 ST
en 1040 STF

EC-68K
Commodore Amiga

Weer een gegarandeerd succes van de elektuur-makers!

Het is gedáán met de achtbitters onder de microprocessoren! Bijna alle moderne personal computers en homecomputers — en ook vele zelfbouwcomputers — zijn uitgerust met een zestienbits microprocessor. De 68000 van de firma Motorola (èn zijn wat smallere broertje, de 68008) neemt daarbij een onaantastbare eerste plaats in.

Deze wereldkampioen onder de zestienbitters wordt uitgebreid besproken in dit boek, en in het daarop aansluitende tweede deel. Na een oriënterend hoofdstuk over de aansluitingen en het programmeermodel volgt een hoofdstuk, waarin de hardware en het tijdsverloop van relevante signalen tot in de kleinste details worden behandeld.

Dè 68000 weet zich omringd door een groot aantal ondersteunende chips. Daarbij blijft het niet uitsluitend binnen de (68000-)familie: ook de bestaande achtbits geïntegreerde satelliethardware voelt zich er thuis. Dit is een apart hoofdstuk waard.

En wát te zeggen van de unieke adresseringsmogelijkheden die de 68000 kent! Het vierde hoofdstuk gaat daar zeer uitgebreid op in.

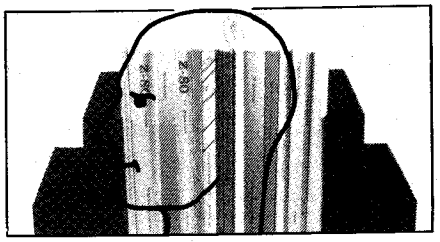
Er is een uitgebreid aanhangsel gewijd aan de 68008-versie van de 68000. Dit omdat de 68008 de overbrugging vormt van achtbitters naar zestienbitters: èn achtbits hardware, èn zestienbits software! In een aantal populaire homecomputers klopt dit 8/16 bits hart, hetgeen resulteert in een uiterst gunstige prestatie/prijsverhouding.
Ca. 220 blz. *f* 42,50/Bfrs. 850,—
ISBN 90-70160-41-2

Het boek **68000 deel 2** is in zijn geheel gewijd aan de kompakte, maar zeer krachtige instruktieset, met een totnutoe onbekende hoeveelheid details en tips.
Deel 2 verschijnt in juni.

De handboeken **68000 deel 1** en **68000 deel 2** vormen "verplichte nummers" voor iedereen die op de een of andere wijze te maken heeft of krijgt met de 68000 of de 68008: de ontwerpers van hardware of software; de bezitters (kompleetkopers of bouwhetzelvers) van personal of homecomputers; of diegenen die "zomaar" een interessant computerstudieonderwerp zoeken.

Bestel deel 1 nu en vul de bestelkaart elders in het blad in!
Blijf de "konkurrentie" voor en lees Elektuur-boeken!!

KNOW-HOW



de 68000 kort en bondig — deel 2

— Alles in één oogopslag —

In deel 1 beschreven we de hardware van de 68000. Nu leren we de software van deze moderne processor kennen. We houden dit artikel kort, zodat óók beginners de 68000 zo snel mogelijk onder de knie kunnen krijgen. De vele tabellen en figuren laten zien, hoe eenvoudig het is om deze moderne processor te doorgronden. Tot slot tonen we aan de hand van een aantal assembler-demonstratieprogramma's aan, hoe gemakkelijk het is om met behulp van de EC-68K-editor/assembler machinetaalprogramma's te maken en te testen. De kompakte presentatie van alle 68000-instrukties en adresseringsmogelijkheden in de vorm van dit artikel is uniek!

De 68000-mnemonics

Voordat men in staat is om een computer via een assembler in machinetaal te programmeren, moet men de mnemonics van de microprocessor kennen. Zoals tabel 1 laat zien kent de 68000 74 mnemonics (instruktie-typen). Lezers die al vertrouwd zijn met de 68XX- en 65XX-processoren hoeven slechts een paar mnemonics nieuw te leren, gezien de verbluffende overeenkomst met de mnemonics voor deze 8-bits-processoren. Voor Z80-fans ligt het wat dit betreft een beetje moeilijker, maar zij zullen snel in de gaten krijgen dat de sprong naar de 68000 eigenlijk maar klein is: de notatie van de bron- en bestemmingsadressen bij de 68000 heeft veel weg van die voor de Z80.

Een blik op de mnemonics in tabel 1 leert dat de 68000 geen load/store-instrukties kent. Ter vereenvoudiging zijn beide instruktie-typen gekombineerd in de MOVE-instruktie. Deze instruktie verzorgt het datatransport van een bron-adres naar een bestemmingsadres. Het bron-adres kan één van de interne registers van de

Tabel 1: De mnemonics van de 68000

Mnemonic	Engels	Nederlands
ABCD	Add decimal with Extend	decimaal optellen (inkl. X-bit)
ADD	Add Binary	binair optellen
ADDA	Add Address	tel op bij inhoud adresregister
ADDI	Add Immediate	immediate data optellen
ADDQ	Add Quick	verhoog registerinhoud met 1...8
ADDX	Add Extended	binair optellen (inkl. X-bit)
AND	AND Logical	logische EN
ANDI	AND Immediate	logische EN op immediate data
ASL	Arithmetic Shift Left	bitsverschuiving naar links
ASR	Arithmetic Shift Right	bitsverschuiving naar rechts
Bcc	Branch Conditionally	voorwaardelijke sprong naar ander programma-adres (verspringen)
BCHG	Test a Bit and Change	test bit en wijzig het
BCLR	Test a Bit and Clear	test bit en maak het Ø
BRA	Branch Always	verspring altijd (naar een ander programma-adres)
BSET	Test a Bit and Set	test bit en maak het 1
BSR	Branch to Subroutine	verspring naar een subroutine
BTST	Test a Bit	test een bit
CHK	Check Register Against Bounds	grensonderzoek van een register
CLR	Clear an Operand	maak alle operand-bits Ø
CMP	Compare	vergelijk
CMPA	Compare Address	vergelijk inhoud adresregister met operand
CMPI	Compare Immediate	vergelijk met immediate data
CMPM	Compare Memory	vergelijk twee geheugen-inhouden
DBcc	Test Condition, Decrement, and Branch	test konditie, verlaag en verspring
DIVS	Signed Divide	delen op basis van positieve of negatieve getallen
DIVU	Unsigned Divide	delen op basis van absolute getallen
EOR	Exclusive OR Logical	logische EXOR
EORI	Exclusive OR Immediate	logische EXOR op immediate data
EXG	Exchange Registers	verwissel twee registerinhouden
EXT	Sign Extend	verhoging aantal bits op basis van het teken
ILLEGAL	Illegal Instruktion	niet toegestane instruktie (verspring naar TRAP-vektor)
JMP	Jump	spring
JSR	Jump to Subroutine	spring naar een subroutine
LEA	Load Effective Address	laad het effektief adres
LINK	Link and Allocate	red stackpointer in adresregister en reserveer plaats op stack
LSL	Logical Shift Left	logische bitsverschuiving naar links
LSR	Logical Shift Right	logische bitsverschuiving naar rechts
MOVE	Move Data from Source to Destination	transporteer data van bron naar bestemming
MOVEA	Move Address	transporteer data naar een adresregister
MOVEM	Move Multiple Registers	transporteer meerdere register-inhouden
MOVEP	Move Peripheral Data	transporteer data naar 68000-I/O-komponenten
MOVEQ	Move Quick	transporteer een byte, dat tot 32 bits is uitgebreid op basis v/h teken
MULS	Signed Multiply	vermeningvuldigen op basis van positieve of negatieve getallen
MULU	Unsigned Multiply	vermenigvuldigen op basis van absolute getallen
NBCD	Negate Decimal with Extend	decimale tekenomkering (inkl. X-bit)
NEG	Negate	binaire tekenomkering
NEGX	Negate with Extend	binaire tekenomkering (inkl. X-bit)
NOP	No Operation	geen aktie (alles ongewijzigd)
NOT	Logical Complement	één-komplement
OR	Inclusive OR Logical	inklusieve OF-operatie
ORI	Inclusive OR Immediate	inklusieve OF-operatie op immediate data

processor zijn, of een willekeurig geheugenadres. Hetzelfde geldt voor het bestemmingsadres. De adresseringsbits in de opcode geven aan, van waar naar waar een operand getransporteerd moet worden.

De adresseringsmogelijkheden van de 68000

Tabel 2 toont de adresseringsmogelijkheden van de 68000. De gebruiker kan kiezen uit 12 krachtige adresseringsmogelijkheden. Deze mogelijkheden zijn in kategoriën ingedeeld: data, memory, control en alterable. Men kan bijvoorbeeld bij de kategorie "data" alle adresseringsmogelijkheden toepassen, met uitzondering van "adresregister direkt". Bij de kategorie "alterable" kunnen alle adresseringsmogelijkheden worden toegepast, met uitzondering van "programmateller relatief", idem met index en immediate. Zoals we later nog zullen zien, zijn bij veel instrukties mengvormen van adresseringsmogelijkheden mogelijk, zoals bijvoorbeeld "data-alterable".

Dataregister direkt

Bij deze adresseringsmogelijkheid bevindt zich de operand in één van de acht dataregisters. Bij de meeste instrukties die deze adresseringsmogelijkheid kennen, kan de operand een bit, een byte, een woord of een lang woord zijn. De assembler-notatie luidt: Dn, waarbij n de waarde 0...7 heeft.

Adresregister direkt

De operand bevindt zich in één van de acht adresregisters. Let erop dat A7 door de processor als stackpointer wordt

PEA	Push Effective Address	effektief adres op stack
RESET	Reset external Devices	maak RESET-lijn 0
ROL	Rotate Left without Extend	rotatie naar links (zonder X-bit)
ROR	Rotate Right without Extend	rotatie naar rechts (zonder X-bit)
ROXL	Rotate Left with Extend	rotatie naar links (met X-bit)
ROXR	Rotate Right with Extend	rotatie naar rechts (met X-bit)
RTE	Return from Exception	terugkeer van supervisor mode naar user mode
RTR	Return and Restore Condition Codes	terugkeer van subroutine met behoud van konditiekodes
RTS	Return from Subroutine	terugkeer van subroutine
SBCD	Subtract Decimal with Extend	decimaal aftrekken (inkl. X-bit)
Scc	Set According to Codition	maak de bits v.e. byte 1, afhankelijk van een bepaalde voorwaarde
STOP	Load Status Register and Stop	laad het statusregister en stop
SUB	Subtract Binary	binair aftrekken
SUBA	Subtract Address	trek af van inhoud adresregister
SUBI	Subtract Immediate	trek de immediate data af
SUBQ	Subtract Quick	verlaag registerinhoud met 1...8
SUBX	Subtract with Extend	trek af (inkl. X-bit)
SWAP	Swap Register Halfs	verwissel registerhelften
TAS	Test and Set an Operand	test byte-operand en maak bit 7 1
TRAP	Trap	schakel om op exception processing (software-interrupt)
TRAPV	Trap on Overflow	exception processing (supervisor mode) indien V = 1
TST	Test an Operand	test op operand nul
UNLK	Unlink	tegenovergestelde van LINK. Herstel de toestand van de stack-pointer; adresregister krijgt oorspronkelijke inhoud terug

Tabel 1. De mnemonics van de 68000, met Engelse en Nederlandstalige toelichting.

Tabel 2: De adresseringsmogelijkheden van de 68000

Adresserings-mogelijkheid	Assembler-syntax	Data	Memory	Control	Alterable
dataregister direkt	Dn	*			*
adresregister direkt	An				*
adresregister in-direkt	(An)	*	*	*	*
adresregister indi-rekt (verhoging achteraf)	(An)+	*	*		*
adresregister indi-rekt (verlaging vooraf)	-(An)	*	*		*
adresregister indi-rekt (16-bits offset)	d16(An)	*	*	*	*
adresregister indi-rekt (8-bits offset + index)	d8(An,Xi)	*	*	*	*
absoluut kort	xxxx	*	*	*	*
absoluut lang	xxxxxx	*	*	*	*
programmateller relatief (16-bits offset)	d16(PC)	*	*	*	
programmateller relatief (8-bits offset + index)	d8(PC,Xi)	*	*	*	
immediate (onmid-dellijk)	#xxxx	*	*		

Opmerkingen:

d8 = 8-bits data (byte)
d16 =16-bits data (woord)
An = een adresregister
Dn = een dataregister
Xi = indexregister. Het indexregister is een data- of adresregister. De inhoud van het indexregister wordt als een positief of een negatief getal opgevat (.W of .L).

Figuur 1. De adresseringsmogelijkheid: dataregister direkt.

Figuur 2. De adresseringsmogelijkheid: adresregister direkt.

Figuur 3. De adresseringsmogelijkheid: adresregister indirekt.

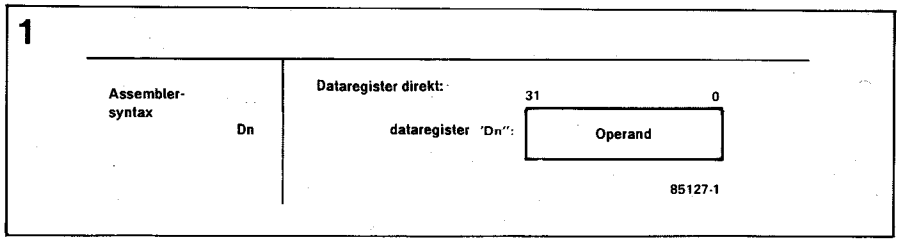
Figuur 4. De adresseringsmogelijkheid: adresregister indirekt met verhoging achteraf.

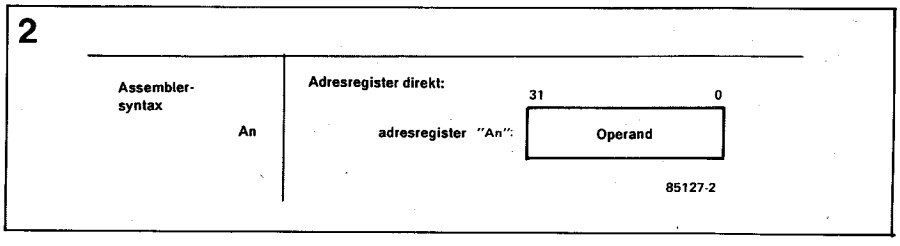
Figuur 5. De adresseringsmogelijkheid: adresregister indirekt met verlaging vooraf.

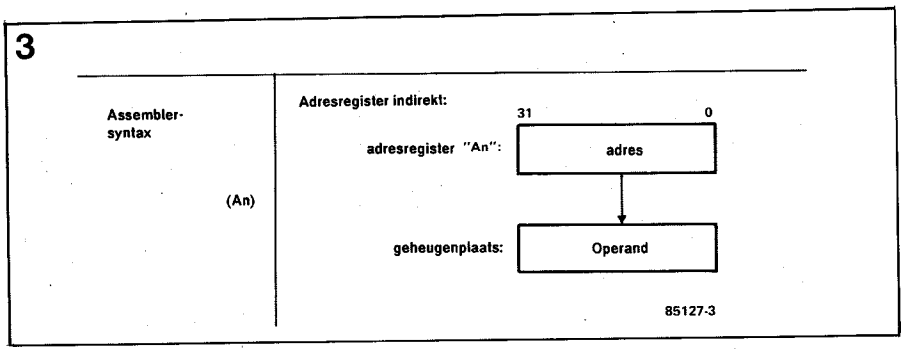
Figuur 6. De adresseringsmogelijkheid: adresregister indirekt met 16-bits offset.

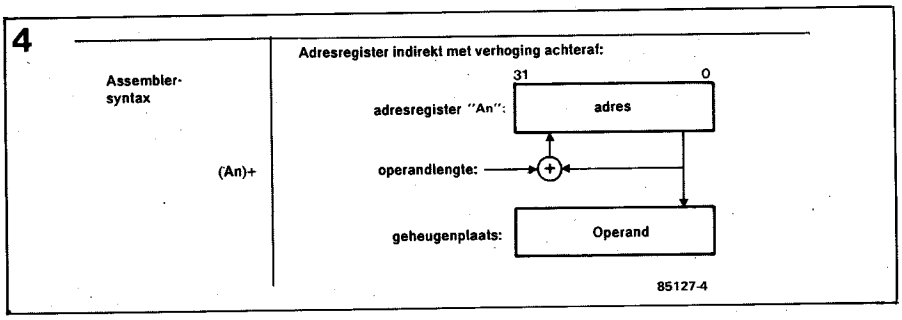
Figuur 7. De adresseringsmogelijkheid: adresregister indirekt met 8-bits offset en index. Als index kan een dataregister of een adresregister fungeren. De offset is positief of negatief.

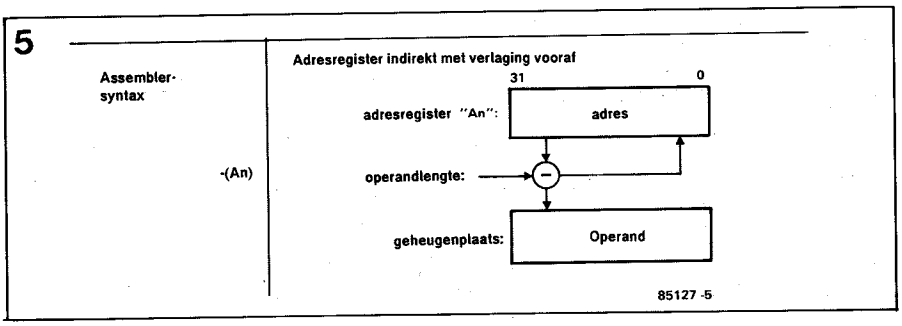
Figuur 8. De adresseringsmogelijkheid: programmateller relatief met 16-bits offset. De offset is positief of negatief. Met deze manier van adresseren is het mogelijk om verplaatsbare (relocatable) programma's te schrijven.

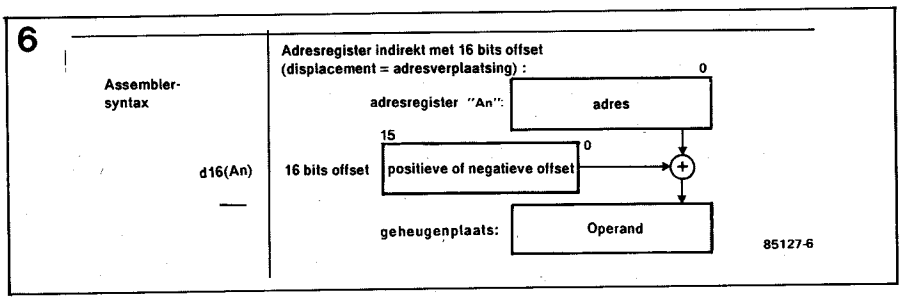


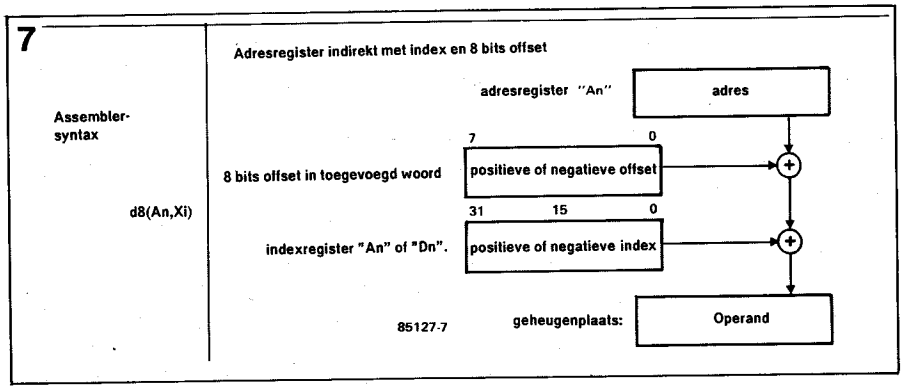


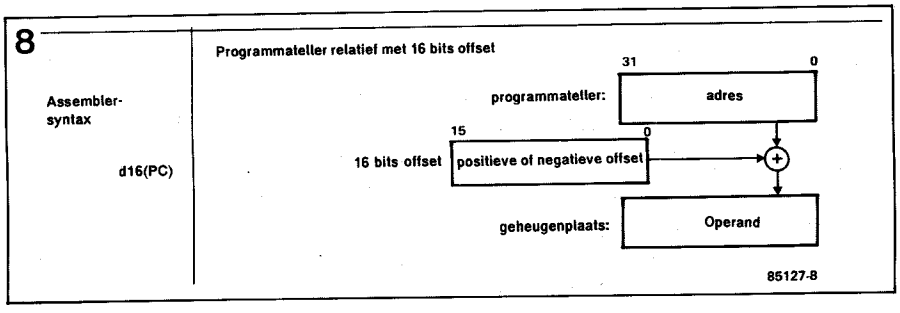












gebruikt. De notaties A7 en SP zijn derhalve identiek voor vele assemblers. Bij de meeste instrukties die deze adresseringsmogelijkheid kennen, kan de operand een woord of een lang woord zijn. De assembler-notatie luidt: An, waarbij n de waarde 0...7 heeft.

Adresregister indirekt

De operand bevindt zich in het computergeheugen. Het adres van de operand is in een adresregister gespecificeerd. Het adresregister wijst dus op de operand (adreswijzer = pointer). De operand kan een bit, een byte, een woord of een lang woord zijn. Deze adresseringsmogelijkheid behoort tot de kategorie "data", met uitzondering van JMP en JSR. De assembler-notatie luidt: (An), waarbij n = 0...7.

Adresregister indirekt met verhoging achteraf

De operand bevindt zich in het computergeheugen. Het operand-adres staat in een adresregister. Het adresregister wijst dus op de operand (pointer = adreswijzer). Na gebruik van het operand-adres wordt het adresregister kwa inhoud verhoogd met een bedrag dat van het operandformaat afhangt:
byte: (An)=(An)+1
woord: (An)=(An)+2
lang woord: (An)=(An)+4
De assembler-notatie luidt: (An)+. Ook hier is n = 0...7.
Deze adresseringsmogelijkheid behoort tot de kategorie "data".

Adresregister indirekt met verlaging vooraf

De operand bevindt zich in het computergeheugen. Het operand-adres staat in een adresregister. Het adresregister wijst dus op de operand (pointer = adreswijzer). Voordat de processor het operand-adres gebruikt, wordt de inhoud van het adresregister met een bedrag verlaagd, dat van het operand-formaat afhangt (byte: verlaging met 1; woord: verlaging met 2; lang woord: verlaging met 4). Deze adresseringsmogelijkheid behoort tot de kategorie "data". De assembler-notatie luidt —(An), waarbij n = 0...7.

Adresregister indirekt met 16-bits offset

De operand bevindt zich in het computergeheugen. Het operand-adres is de adresregisterinhoud plus een 16-bits offset met voorteken (d15). De assembler-notatie luidt: d16(An), waarbij n = 0...7. Deze adresseringsmogelijkheid behoort tot de kategorie "data", met uitzondering van JMP en JSR.

Adresregister indirekt met index en 8-bits offset

De operand bevindt zich in het computergeheugen. In tegenstelling tot de andere adresseringsmogelijkheden kan de opcode niet meer in één woord worden ondergebracht. Daarom wordt een woord toegevoegd. De bits 15...8 vormen de aanvulling op de opcode, terwijl de bits 7...0 een positive of negatieve 8-bits offset (d8) bevatten. Het operand-adres is:

inhoud An plus (of min) 8 bits offset plus (of min) de inhoud van het indexregister Xi. De index kan een woord of een lang woord zijn. Als index kan de inhoud van een adres- of dataregister dienen. De assemblernotatie luidt: d8(An,Xi), waarbij n = 0...7; X = A of D; i = 0...7). Deze adresseringsmogelijkheid behoort tot de kategorie "data", met uitzondering van JMP en JSR.

Programmateller relatief met 16-bits offset

De operand bevindt zich in het computergeheugen. Het geheugenadres is: inhoud programmateller plus (of min) een 16-bits offset. Uitgangspunt voor de inhoud van de programmateller is het programmaadres met de opcode, met daarbij het bedrag twee opgeteld. Het grote voordeel van deze adresseringsmogelijkheid is, dat het nu mogelijk is om positieonafhankelijk programma's te schrijven (relocatable = verplaatsbaar). Deze adresseringsmogelijkheid behoort tot de kategorie "program", een mengvorm van verschillende kategoriën van tabel 2.
De meeste 8-bits microprocessoren kennen deze vorm van adresseren niet. Vandaar dat we nu de 68000-beginners uitleggen wat dit in de praktijk betekent. Deze uitleg gaat het beste aan de hand van een vergelijking met de bekende adresseringsmethode "absoluut" of "extended". Normaal is het zo dat de processor het absolute adres van de operand in het geheugen moet kennen. Een assembler is in staat om zo'n absoluut adres (dat door een label is aangegeven) te berekenen. Wil men echter het machinetaalprogramma naar een ander geheugenbereik verplaatsen, dan moeten deze adressen opnieuw worden berekend.
Hoe anders gaat het toe bij de adresseringsvorm programmateller relatief. Hier berekent de assembler geen absoluut adres, maar een offset (adresverplaatsing) ten opzichte van de momentele inhoud van de programmateller. De inhoud van de programmateller plus (of min, afhankelijk van het teken) de offset, levert het effektief adres op. De assembler-notatie luidt: d16(PC).

Programmateller relatief met 8-bits offset en index

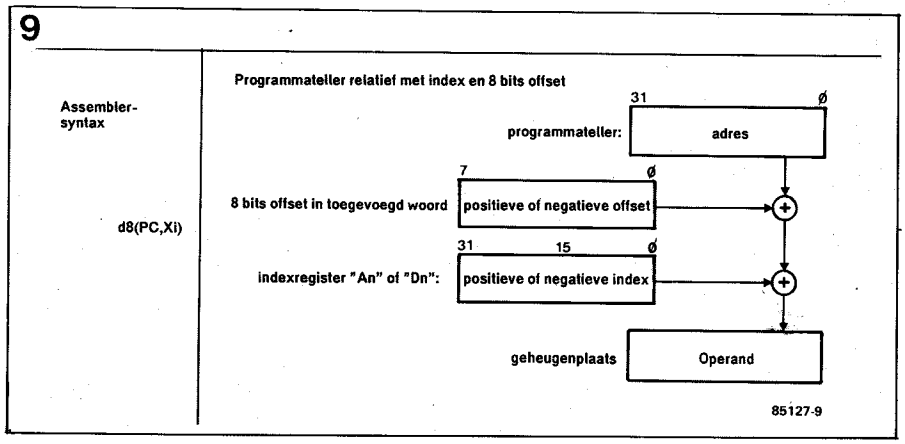
Bij deze adresseringsmogelijkheid bevindt zich de operand in het computergeheugen. De opcode kan niet meer in één woord worden ondergebracht. Daarom wordt een woord toegevoegd. De bits 15...8 van dit toegevoegde woord vormen de aanvulling op de opcode, terwijl de bits 7...0 een positieve of negatieve 8-bits offset bevatten. Het operand-adres is: inhoud programmateller plus (of min) de 8-bits offset plus (of min) de inhoud van het indexregister Xi. De index kan een woord of een lang woord zijn. Als index kan een adres- of dataregister worden genomen. De assembler-notatie luidt: d8(PC,Xi), met X = An of Dn, i = n = 0...7. Deze vorm van adresseren hoort tot de kategorie "program".

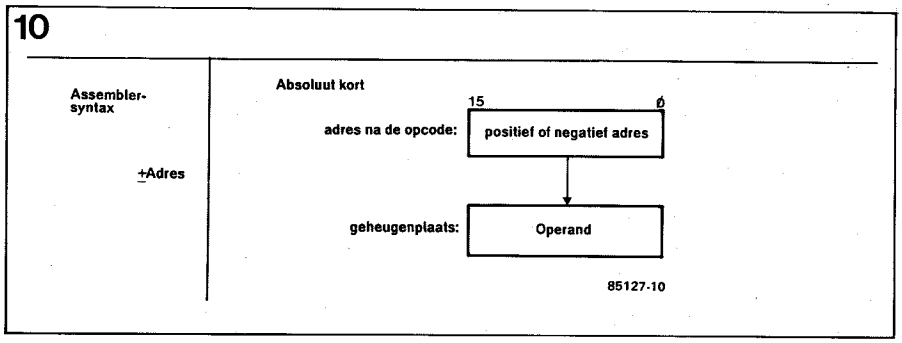
Absoluut kort

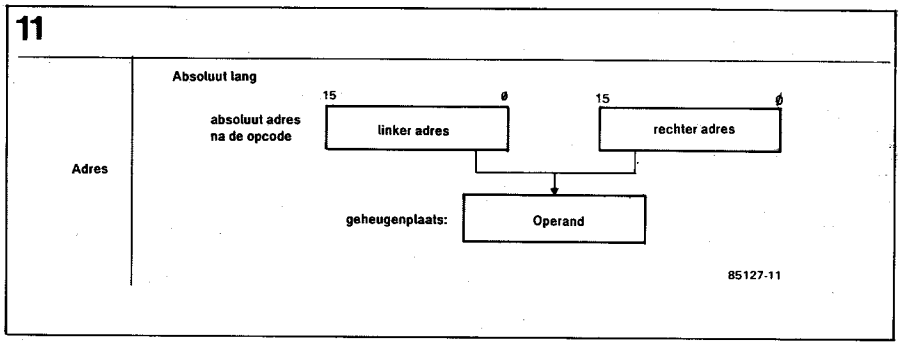
Bij deze adresseringsvorm bevindt zich de operand in het computergeheugen. Het absolute (positieve of negatieve) 16-bits adres is in een toegevoegd woord gespecificeerd. Omdat dit adres positief of negatief kan zijn, heeft de processor toegang tot operanden die zich maximaal 32Kbyte boven of onder het adres $000000 in het geheugen bevinden. Deze adresseringsvorm behoort tot de kategorie "data" (JMP en JSR uitgezonderd).

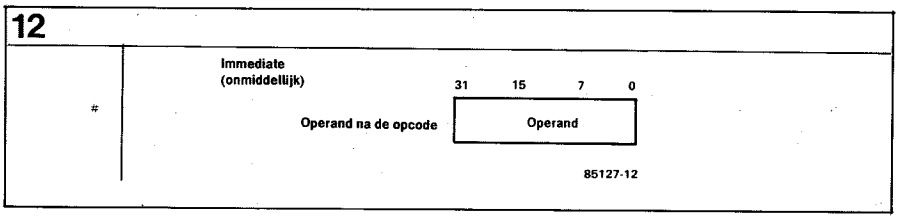
Absoluut lang

Bij deze adresseringsvorm bevindt zich de operand in het computergeheugen. Het absolute 32-bits adres is in de vorm van twee woorden aan de opcode toegevoegd. Het eerste toegevoegde woord bevat het linker adreswoord (MSW), het daarop volgende tweede toegevoegde woord bevat het rechter adreswoord (LSW). Absolute adressering dient men slechts in uitzonderlijke situaties toe te passen. Dat neemt niet weg data deze vorm van adresseren nuttig is bij het initia-









Figuur 9. De adresseringsmogelijkheid: programmateller relatief met 8-bits offset en index. Als indexregister kan een dataregister of een adresregister fungeren. De offset is positief of negatief. Ook met deze vorm van adresseren is het mogelijk om verplaatsbare software te schrijven.

Figuur 10. De adresseringsmogelijkheid: absoluut kort. Het 16-bits absolute adres achter de opcode is positief of negatief.

Figuur 11. De adresseringsmogelijkheid: absoluut lang. Het absolute adres direkt achter de opcode beslaat een lang woord.

Figuur 12. De adresseringsmogelijkheid: immediate (onmiddellijk).

Tabel 3. Een kompakt instruktie-overzicht voor de 68000- en de 68008-microprocessor.

Tabel 3: De instrukties van de 68000

Mnemonic	Syntax	Operand-forma(a)t(en)	"y" oorsprong	"x" bestemming	X	N	Z	V	C
ABCD	ABCD Dy,Dx	.B	Dn	Dn	*	u	*	u	*
	ABCD -(Ay),-(Ax)	.B	-(An)	-(An)	*	u	*	u	*
ADD	ADD <ea>,Dn	.B .W .L	alle mogelijkheden (1)	Dn	*	*	*	*	*
	ADD Dn,<ea>	.B .W .L	Dn	"Alterable"	*	*	*	*	*
ADDA	ADDA <ea>,An	.W .L	alle mogelijkheden	An					
ADDI	ADDI #data,<ea>	.B .W .L	#data	"Data Alterable"	*	*	*	*	*
ADDQ	ADDQ #data,<ea>	.B .W .L	#data (1, 2)	"Alterable"	*	*	*	*	*
ADDX	ADDX Dy,Dx	.B .W .L	Dn	Dn	*	*	*	*	*
	ADDX -(Ay),-(Ax)	.B .W .L	-(An)	-(An)	*	*	*	*	*
AND	AND <ea>,Dn	.B .W .L	"Data"	Dn		*	*	0	0
	AND Dn,<ea>	.B .W .L	Dn	"Alterable"		*	*	0	0
ANDI	ANDI #data,<ea>	.B .W .L	#data	"Data Alterable"		*	*	0	0
	ANDI #data,SR (3)	.B .W	#data	Statusreg.	*	*	*	*	*
ASL	ASL Dx,Dy	.B .W .L	Dn (4)	Dn	*	*	*	*	*
	ASL #data,Dn	.B .W .L	#data (5)	Dn	*	*	*	*	*
	ASL <ea>	.W		"Memory Alterable"	*	*	*	*	*
ASR	ASR Dx,Dy	.B .W .L	Dn (4)	Dn	*	*	*	*	*
	ASR #data,Dn	.B .W .L	#data (5)	Dn	*	*	*	*	*
	ASR <ea>	.W		"Memory Alterable	*	*	*	*	*
Bcc	Bcc <Label>	d8,d16	PC+dx indien "cc"						
BCHG	BCHG Dn,<ea>	.B .L (9)	Dn	"Data Alterable"			*		
	BCHG #data,<ea>	.B .L (9)	#data	"Data Alterable"			*		
BCLR	BCLR Dn,<ea>	.B .L (9)	Dn	"Data Alterable"			*		
	BCLR #data,<ea>	.B .L (9)	#data	"Data Alterable"			*		
BRA	BRA <Label>	d8,d16	PC+dx						
BSET	BSET Dn,<ea>	.B .L (9)	Dn	"Data Alterable"			*		
	BSET #data,<ea>	.B .L (9)	#data	"Data Alterable"			*		
BSR	BSR <Label>	d8,d16	PC --> -(SP),PC=PC+dx						
BTST	BTST Dn,<ea>	.B .L (9)	Dn	"Data"			*		
	BTST #data,<ea>	.B .L (9)	#data	"Data"			*		
CHK	CHK <ea>,Dn	.W	indien Dn<0 of Dn>(ea), dan TRAP	"Data"		*	u	u	u
CLR	CLR <ea>	.B .W .L	"Data Alterable"			0	1	0	0
CMP	CMP <ea>,Dn	.B .W .L	alle mogelijkheden (1)	Dn		*	*	*	*
CMPA	CMPA <ea>,An	.B .W .L	alle mogelijkheden	An		*	*	*	*
CMPI	CMPI #data,<ea>	.B .W .L	#data	"Data Alterable"		*	*	*	*
CMPM	CMPM (Ay)+,(Ax)+	.B W .L	(An)+	(An)+		*	*	*	*
DBcc	DBcc Dn,<Label>	.W	indien "cc", dan Dn=Dn-1						
		.W	indien Dn<>-1, dan PC=PC+dx						
DIVS	DIVS <ea>,Dn	.W	"Data"	Dn		*	*	*	0
DIVU	DIVU <ea>,Dn	.W	"Data"	Dn		*	*	*	0
EOR	EOR Dn,<ea>	.B .W .L	Dn	"Data Alterable"		*	*	0	0
EORI	EORI #data,<ea>	.B .W .L	#data	"Data Alterable"		*	*	0	0
	EORI #data,SR (3)	.B .W	#data	Statusreg.	*	*	*	*	*
EXG	EXG Rx,Ry	.L	Dn of An	Dn of An		*	*	0	0
EXT	EXT Dn	.W .L	Dn			*	*	0	0
JMP	JMP <ea>		<ea> --> PC	"Control"					
JSR	JSR <ea>		PC --> -(SP),PC=<ea>	"Control"					
LEA	LEA <ea>,An	.L	"Control"	An					
LINK	LINK An,#-Offset		An						
LSL	LSL Dx,Dy	.B .W .L	Dn	Dn	*	*	*	0	*
	LSL #data,Dn	.B .W .L	#data	Dn	*	*	*	0	*
	LSL <ea>	.W		"Memory Alterable"	*	*	*	0	*
LSR	LSR Dx,Dy	.B .W .L	Dn	Dn	*	*	*	0	*
	LSR #data,Dn	.B .W .L	#data	Dn	*	*	*	0	*
	LSR <ea>	.W		"Memory Alterable"	*	*	*	0	*
MOVE	MOVE <ea>,<ea>	.B .W .L	alle mogelijkheden (1)	"Data Alterable"		*	*	0	0
	MOVE <ea>,CCR	.W	"Data"	Flags in Statusreg.	*	*	*	*	*
	MOVE <ea>,SR	.W	"Data"	Statusreg.	*	*	*	*	*
	MOVE SR,<ea>	.W	Statusreg.	"Data Alterable"					
	MOVE USP,An	.L	User-Stack-Pointer	An					
	MOVE An,USP (6)	.L	An	User-Stack-Pointer					
MOVEA	MOVEA <ea>,An	.W .L	alle mogelijkheden	An					
MOVEM	MOVEM <Reg.-lijst>,<ea>	.W .L		"Control Alterable" of -(An)					
	MOVEM <ea>,<Reg.-lijst>	.W .L	"Control" of (An)+						

liseren en de verdere besturing van I/O-komponenten.

Immediate (onmiddellijk)

Bij deze adresseringsmogelijkheid volgt de operand in het programmageheugen onmiddellijk (vandaar: immediate) op de opcode. De operand kan een byte, een woord of een lang woord zijn. Afhankelijk van het operand-formaat is er sprake van één (byte, woord) of twee (lang woord) toegevoegde woorden.

De instrukties van de 68000

Nadat de mnemonics (tabel 1), de adresseringsmogelijkheden (figuren 1. . .12) en de adresseringskategoriën (tabel 2) bekend zijn, bekijken we de 68000-instrukties in detail. Tabel 3 toont alles in één oogopslag:

* mnemonics
* mnemonics-variaties
* assembler-syntax
* operandformaat/formaten
* bron en bestemming van de operand
* toegestane adresseringsmogelijkheden
* vlaggen van het statusregister, die 1 of 0 worden of ongewijzigd blijven

Tabel 3 dient men altijd te gebruiken indien machinetaalprogramma's via de assembler worden gemaakt. Zowel de beginners als de gevorderden onder u vinden in deze tabel alle wetenswaardigheden over de 68000-instrukties. Samen met de foutmeldingen van de EC68K-assembler kunnen programmeerfouten eenvoudig worden opgespoord. Deze tabel heeft bij ons zijn sporen al verdiend!
In tabel 4 zijn de voorwaarden en de voorwaarden-kodes opgesomd. Let goed op het verschil tussen absolute data en voorteken-afhankelijke data.
En nu laten we aan de hand van een aantal assemblerprogramma's zien hoe simpel het is om met de EC68K-assembler machinetaalprogramma's te maken.

Programmavoorbeeld 1

Allereerst laten we zien hoe men met de EC68K-editor een programma in de computer zet en in aansluiting daarop assembleert. Figuur 13 toont het source-programma, zoals dit met de tekstverwerker van de EC68K is opgesteld. Figuur 14 toont het geassembleerde programma. Het programma verzorgt de omzetting van een via het toetsenbord ingegeven decimaal getal in een hexadecimaal getal. De uitvoering van het programma omvat de volgende stappen:

* Neem dit demonstratieprogramma op in het menu van het monitorprogramma
* laad een decimaal getal als getallen-string in een buffer
* zet de getallen-string om in een drijvende-komma-getal
* vertaal dit drijvende-komma-getal in een binair getal
* geef het binaire getal hexadecimaal weer
* wacht op een willekeurige ingedrukte toets
* keer terug naar het hoofdmenu

In regel 80 is de programmastart gedefinieerd. Via de assembler-pseudo-instruk-

MOVEP	MOVEP Dn,d16(An)	.W .L	Dn	d16(An)					
	MOVEP d16(An),Dn	.W .L	d16(An)	Dn					
MOVEQ	MOVEQ #data,Dn	.L	#data (7)	Dn		*	*	Ø	Ø
MULS	MULS <ea>,Dn	.W	"Data"	Dn		*	*	Ø	Ø
MULU	MULU <ea>,Dn	.W	"Data"	Dn		*	*	Ø	Ø
NBCD	NBCD <ea>	.B		"Data Alterable"	*	u	*	u	*
NEG	NEG <ea>	.B .W .L		"Data Alterable"	*	*	*	*	*
NEGX	NEGX <ea>	.B .W .L		"Data Alterable"	*	*	*	*	*
NOP	NOP		PC=PC+2						
NOT	NOT <ea>	.B .W .L		"Data Alterable"		*	*	Ø	Ø
OR	OR <ea>,Dn	.B .W .L	"Data"	Dn		*	*	Ø	Ø
	OR Dn,<ea>	.B .W .L	Dn	"Memory alterable"		*	*	Ø	Ø
ORI	ORI #data,<ea>	.B .W .L	#data	"Data Alterable"		*	*	Ø	Ø
	ORI #data,SR (3)	.B .W	#data	Statusreg.	*	*	*	*	*
PEA	PEA <ea>	.L	"Control"						
RESET (6)	RESET								
ROL	ROL Dx,Dy	.B .W .L	Dn (4)	Dn		*	*	Ø	*
	ROL #data,Dn	.B .W .L	#data (5)	Dn		*	*	Ø	*
	ROL <ea>	.W		"Memory Alterable"		*	*	Ø	*
ROR	ROR Dx,Dy	.B .W .L	Dn	Dn		*	*	Ø	*
	ROL #data,Dn	.B .W .L	#data (5)	Dn		*	*	Ø	*
	ROL <ea>	.W		"Memory Alterable"		*	*	Ø	*
ROXL	ROXL Dx,Dy	.B .W .L	Dn	Dn	*	*	*	Ø	*
	ROXL #data,Dn	.B .W .L	#data (5)	Dn	*	*	*	Ø	*
	ROXL <ea>	.W		"Memory Alterable"	*	*	*	Ø	*
ROXR	ROXR Dx,Dy	.B .W .L	Dn	Dn	*	*	*	Ø	*
	ROXR #data,Dn	.B .W .L	#data (5)	Dn	*	*	*	Ø	*
	ROXR <ea>	.W		"Memory Alterable"	*	*	*	Ø	*
RTE (6)	RTE		(SP)+ --> SR;(SP)+ --> PC		*	*	*	*	*
RTR	RTR		(SP)+ --> CCR,(SP)+ --> PC		*	*	*	*	*
RTS	RTS		(SP)+ --> PC						
SBCD	SBCD Dy,Dx	.B	Dn	Dn	*	u	*	u	*
	SBCD -(Ay),-(Ax)	.B	-(An)	-(An)	*	u	*	u	*
Scc	Scc <ea>	.B	indien "cc", dan $FF --> (ea) anders $ØØ --> (ea)	"Data Alterable"					
STOP (6)	STOP #data	.W	#data --> SR; STOP		*	*	*	*	*
SUB	SUB <ea>,Dn	.B .W .L	alle mogelijkheden	Dn	*	*	*	*	*
	SUB Dn,<ea>	.B .W .L	Dn	"Alterable"	*	*	*	*	*
SUBA	SUBA <ea>,An	.W .L	alle mogelijkheden	An					
SUBI	SUBI #data,<ea>	.B .W .L	#data	"Data Alterable"	*	*	*	*	*
SUBQ	SUBQ #data,<ea>	.B .W .L	#data	"Alterable" (1)	*	*	*	*	*
SUBX	SUBX Dy,Dx	.B .W .L	Dn	Dn	*	*	*	*	*
	SUBX -(Ay),(ax)	.B .W .L	-(An)	-(An)	*	*	*	*	*
SWAP	SWAP Dn	.W	Dn						
TAS	TAS <ea>	.B		"Data Alterable"		*	*	Ø	Ø
TRAP	TRAP #<Vektor>		PC --> -(SP),SR --> -(SP) #<Vektor> --> PC						
TRAPV	TRAPV		indien V=1,dan TRAP						
TST	TST <ea>	.B .W .L		"Data Alterable"		*	*	Ø	Ø
UNLK	UNLK An			An					

Opmerkingen:

X, N, Z, V, C: "*" betekent: wordt beïnvloed; "u" betekent: ongewijzigd.
Overige mogelijkheden: "Ø", "1" of niet beïnvloed.

(1) Voor de adresseringswijze An is .B niet toegestaan.
(2) Onmiddellijke (immediate) data: 1...8 (in operatiewoord).
(3) Indien operandformaat .W: uitsluitend in de supervisor-mode.
(4) Een dataregister doet dienst als schuifteller/rotatieteller. De inhoud van de teller varieert van 0...63. In het geval van een tellerinhoud 0 wordt 64 keer geschoven of geroteerd.
(5) Bij deze operatiewijze kan slechts 1...8 keer geschoven of geroteerd worden.
(6) Bevoorrechte (privileged) instruktie. Slechts uitvoerbaar in de supervisor-mode.
(7) Onmiddellijke (immediate) data uitgebreid tot 32 bits op basis van het teken.
(8) Deze instruktie is niet nodig bij de 68008.
(9) Voor de adresseringswijze Dn is het operandformaat uitsluitend .L; voor de overige adresseringsmogelijkheden: uitsluitend .B.

tie "entry" delen we het operating system (systeemprogramma) mee, op welk adres het demonstratieprogramma gestart moet worden nadat een run-kommando is gegeven. In de regels 100 en 110 leggen we met de karakterstring "Lib." vast dat ons demonstratieprogramma in het menu van het monitorprogramma opgenomen dient te worden. Let erop dat de karakterstring "Lib:....." altijd precies 20 karakters lang is!
Vanaf regel 120 begint het eigenlijke programma. Allereerst richten we AØ op de toetsenbord-ingangsbuffer, die in regel 220 is gedefinieerd (40 karakters lang). In regel 130 geven we de karakter-string "Decimaal getal?" (voorafgegaan door de stuurkodes <cr> en <lf>) weer op het beeldscherm. De melding wordt afgesloten via de Ø aan het eind van regel 200.
In regel 140 laden we de ASCII-buffer met het opgegeven decimale getal. Indien bijvoorbeeld het decimale getal 123 is opgegeven, staat er $31,$32,$33,$ØØ in deze buffer. Deze string lezen we met behulp van de assembler-pseudo-instruktie "finput" in het dataregister DØ. In DØ staat het getal nu in de vorm van een drijvendekomma-getal "genoteerd". In regel 160 wordt dit getal in een hexadecimaal getal omgezet ("fix"). Vervolgens geven we de in regel 210 vastgelegde tekst weer door middel van regel 170. In aansluiting daarop wordt het hexadecimale getal in DØ in de vorm van een lang woord (OUTHEX.L) weergegeven. Ons programma is nu klaar en het zou nu direkt via de RTS-instruktie terugkeren naar de plaats, van waaruit dit demonstratieprogramma is aangeroepen. Daarom wachten we in regel 185 op het indrukken van een willekeurige toets.
Nadat het programma is ingetypt, kunt u dit programma assembleren: <esc>assemble<cr>. In het geval van intoetsfouten geeft de assembler een foutmelding en plaatst hij de cursor op de positie waar de fout is gekonstateerd. Gemakkelijker kán het gewoon niet! Als alles korrekt is ingegeven, dan geeft u het kommando <esc>exit<cr> op. U ziet nu het menu van de monitor in beeld verschijnen; dit menu is uitgebreid met het demonstratieprogramma "dec/hex".

Programmavoorbeeld 2

Figuur 15 toont het tweede programmavoorbeeld. Hier laten we zien hoe eenvoudig het is om getallen in geformatteerde vorm weer te geven. Daartoe maken we gebruik van de formatprecompiler, die deel uitmaakt van het operating system van de EC68K. Het programma doorloopt de volgende stappen:

* Laad het dataregister DØ met de data $FFFF;
* geef de inhoud van DØ decimaal weer op het beeldscherm;
* de cijfers van het getal worden voorafgegaan door het teken + of —;

Allereerst leggen we weer het programma-startadres vast. Daarna (regel 110) wordt de onderste helft van het register DØ (de rechter 16 bits) met de data $FFFF geladen. De decimale getallenstring dient in een buffer te worden gezet, waarvan de inhoud vervolgens op het beeldscherm weergegeven dient te worden. Vandaar dat we in regel 120 via de

Tabel 4: Voorwaarden en voorwaarde-kodes

Suffix "cc"	Voorwaarde (Engels)	Voorwaarde (Nederlands)	is zo indien:
EQ	equal to	gelijk aan (=)	Z=1
NE	not equal to	ongelijk aan (≠)	Z=0
MI	minus	negatief (−)	N=1
PL	plus	positief (+)	N=0
GT (*)	greater than	groter dan (>)	Z AND (N EXOR V)=0
LT (*)	less than	kleiner dan (<)	N EXOR V=0
GE (*)	greater than or equal to	groter dan of gelijk aan (≥)	N EXOR V=1
LE (*)	less than or equal to	kleiner dan of gelijk aan (≤)	Z OR (N EXOR V)=1
HI	higher than	in absolute zin groter dan (>)abs	C AND Z=0
LS	lower than or same as	in absolute zin kleiner dan of gelijk aan (≤)abs	C OR Z=1
CS	carry set	carry 1	C=1
CC	carry clear	carry 0	C=0
VS (*)	overflow set	overflow 1	V=1
VC (*)	overflow clear	overflow 0	V=0
T	always true	altijd waar	
F	always false	nooit waar	

(*) = twee's-komplement-rekenen

Tabel 4. Een overzicht van voorwaarden voor een bepaalde aktie, inklusief de voorwaarde-kodes die getest moeten worden.

Figuur 13. Zo ziet een assembler-source-programma er uit dat met de editor van de EC-68K is geschreven.

Figuur 14. Dit programma zet een decimaal getal in een hexadecimaal getal om en geeft dat laatste getal weer op het beeldscherm.

Figuur 15. Dit programma zet een hexadecimaal getal om in een decimaal getal (drijvende-komma-getal), dat in geformatteerde vorm op het beeldscherm wordt weergegeven.

adreswijzer AØ naar de ASCII-buffer wijzen. Let s.v.p. goed op de gebruikte adresseringsvorm: programmateller relatief! In regel 130 openen en initialiseren we de ASCII-uitgangsbuffer ("fpinit"). De buffer wordt in overeenstemming met de formatteringsformule van regel 180 geformatteerd. De formule is als volgt opgebouwd:

c: carriage return
l: line feed
'D0 is decimaal': karakter-string
x: spatie
s: voorteken (+ of −)
6d: geef zes cijfers weer
.: druk een punt af

Nadat de uitgangsbuffer is geïnitialiseerd (regel 190), geven we de inhoud van het dataregister DØ weer in de vorm van een getallen-string (decimaal getal), zoals dat zich in de buffer bevindt ("fpint"). In regel 150 wordt de uitgangsbuffer gesloten. Het resultaat drukken we in regel 160 op het beeldscherm af ("disp").

Programmavoorbeeld 3

Dit demonstratieprogramma (figuur 16) is een variant op het vorige demonstratieprogramma van figuur 15. Daarom kunnen we de beschrijving kort houden. Het programma omvat de volgende stappen:

* Laad het dataregister DØ met de data $FFF
* geef de inhoud van DØ als een drijvende-komma-getal weer

13

```
0010 ********** demo.0 ******************************
0020 * 1) decimaal getal in de ASCII-buffer inlezen *
0030 * 2) ASCII/floating point-konversie            *
0040 * 3) floating point/hex-integer-konversie      *
0050 * 4) hex-getal uitgeven                        *
0060 ********************************************
0070 *
0080 org $23000
0090 entry start
0100 dc.b 'Lib:dec/hex.....'
0110 dr.l start
0120 start: lea.l buffer(pc),a0  wijs naar ASCII-buffer
0130 disp text1(pc)  tekst 1 weergeven
0140 input (a0)  getal in ASCII-buffer inlezen
0150 finput (a0),d0  ASCII/floating point-konversie
0160 fix d0,d0  floating point/hex-integer-konversie
0170 disp text2(pc)  tekst 2 weergeven
0180 outhex.l d0  hexadecimaal weergeven
0185 inchr.b d0  wacht op willekeurige toets
0190 rts
0200 text1: dc.b 13,10,'decimaal getal:  ? ',0
0210 text2: dc.b 13,10,'hexadecimaal getal: ',0
0220 buffer: ds.b 40
0230 end
```

14

```
68000 Assembler  v2.1 demo0.3 13:26:47 10/10/85 page:00001

000000                  00006          LIST
000000                  00010 ********** DEMO.0 ******************************
000000                  00020 * 1) DECIMAAL GETAL IN DE ASCII-BUFFER INLEZEN *
000000                  00030 * 2) ASCII/FLOATING POINT-KONVERSIE            *
000000                  00040 * 3) FLOATING POINT/HEX-INTEGER-KONVERSIE      *
000000                  00050 * 4) HEX-GETAL UITGEVEN                        *
000000                  00060 ********************************************
000000                  00070 *
023000   =00023000      00080          ORG       $23000
023000   =00023018      00090          ENTRY     START
023000 4C69623A6465632F68 00100        DC      .B 'Lib:dec/hex.....'
023014 00000004         00110          DR      .L START
023018 41FA0046         00120 START:   LEA     .L BUFFER(PC),A0  WIJS NAAR ASCII-BUFFER
02301C F07A0016         00130          DISP      TEXT1(PC)  TEKST 1 WEERGEVEN
023020 F250             00140          INPUT     (A0)  GETAL IN ASCII-BUFFER INLEZEN
023022 A00A0400         00150          FINPUT    (A0),D0  ASCII/FLOATING POINT-KONVERSIE
023026 A0050000         00160          FIX       D0,D0  FLOATING POINT/HEX-INTEGER-KONVERSIE
02302A F07A001E         00170          DISP      TEXT2(PC)  TEKST 2 WEERGEVEN
02302E F600             00180          OUTHEX  .L D0  HEXADECIMAAL WEERGEVEN
023030 F100             00185          INCHR   .B D0  WACHT OP WILLEKEURIGE TOETS
023032 4E75             00190          RTS
023034 0D0A64657A696D616C 00200 TEXT1:   DC      .B 13,10,'decimaal getal:  ? ',0
02304A 0D0A6865786164657A 00210 TEXT2:   DC      .B 13,10,'hexadecimaal getal: ',0
023060    :00000028     00220 BUFFER:  DS      .B 40
023088                  00230          END
```

15

```
68000 Assembler  v2.1 demo1.3 13:29:36 10/10/85 page:00001

000000                  00006          LIST
000000                  00010 ********** DEMO.1 **************************
000000                  00020 * 1) LAAD D0 MET $FFFF                     *
000000                  00030 * 2) GEEF D0 ALS FLOATING-POINT-GETAL WEER *
000000                  00040 * 3) PRINTUSING '######'                   *
000000                  00050 ********************************************
000000                  00060 *
023000   =00023000      00070          ORG       $23000
023000   =00023000      00080          ENTRY     START
023000                  00090 *
023000 4280             00100 START:   CLR     .L D0
023002 303CFFFF         00110          MOVE    .W #$FFFF,D0  LAAD D0
023006 41FA002E         00120          LEA     .L BUFFER(PC),A0  OPEN DE ASCII-BUFFER
02300A A0160E80000E     00130          FPINIT    FM(PC),A0  INITIALISEER DE ASCII-BUFFER
023010 A0190008         00140          FPINT     D0,A0  IN DE ASCII-BUFFER DECIMAAL SCHRIJVEN
023014 A01B0008         00150          FPEND     A0  SLUIT DE ASCII-BUFFER
023018 F050             00160          DISP      (A0)  ASCII-BUFFER-INHOUD OP SCHERM WEERGEVEN
02301A 4E75             00170          RTS
02301C                  00172 *
02301C                  00174 * FORMAAT EN ASCII-BUFFER DEFINIEREN
02301C                  00176 *
02301C 434C27643020697374 00180 FM:      FORMAT    CL'd0 is decimaal 'XS6D'.'
023036    :00000028     00190 BUFFER:  DS      .B 40
02305E                  00200          END
```

* neem hiervoor vier cijfers links van de komma
* en neem hiervoor twee cijfers rechts van de komma

In regel 120 wordt het dataregister DØ geladen. In regel 130 zet de instruktie "float" het binaire getal om in een drijvende-komma-getal en zet dit getal vervolgens in het dataregister D1. Bij de omzetting ontstaat een fout die door de afronding in regel 140 wordt gekompenseerd ("fadd" = floaring point add). Het openen, formatteren en sluiten van de uitgangsbuffer vindt op de eerder beschreven wijze plaats. Hier wordt echter een drijvende-komma-getal ("fpreal") naar de uitgangsbuffer verstuurd. In de formatteringsformule van regel 210 wordt vastgelegd dat het drijvende-komma-getal in het formaat "####.##" wordt weergegeven. In regel 100 staat een pseudo-instruktie die tot nu toe nog niet is besproken: "debug 100". Via deze instruktie delen we de assembler mee dat een aanvullende symbooltabel ten behoeve van de debugger gemaakt dient te worden. De parameter "100" achter de debug-instruktie vertelt de assembler dat deze voor maximaal 100 instrukties een debugger-symbool-tabel moet maken. De debug-instruktie kan op een willekeurige positie in het source-programma worden geplaatst. Op deze wijze kan men in een omvangrijk assembler-programma kleinere of grotere modulen (programmasekties) op komfortabele wijze testen. De debugger wordt gestart door het kommando: <esc>debug<cr>.

16

```
68000 Assembler  v2.1 demo2.2 13:32:42 10/10/85 page:00001

000000                  00006          LIST
000000                  00010 ********** DEMO.2 **************************
000000                  00020 * 1) LAAD D0 MET $FFF                      *
000000                  00030 * 2) GEEF D0 ALS FLOATING-POINT-GETAL WEER *
000000                  00040 * 3) PRINTUSING '####.##'                  *
000000                  00050 ***************************************
000000                  00060 *
023000    =00023000     00070          ORG       $23000
023000    =00023000     00080          ENTRY     START
023000                  00090 *
023000    =00000064     00100          DEBUG     100
023000 4280             00110 START:   CLR     .L D0
023002 303C0FFF         00120          MOVE    .W #$FFF,D0  LAAD D0 HEXADECIMAAL
023006 A0040001         00130          FLOAT     D0,D1  HEX/FLOATING POINT-KONVERSIE
02300A A0000F01F851EB85 00140          FADD      #5.005,D1  AFRONDING
023012 41FA002C         00150          LEA     .L BUFFER(PC),A0  OPEN DE ASCII-BUFFER
023016 A0160E88000E     00160          FPINIT    FM(PC),A0  INITIALISEER DE ASCII-BUFFER
02301C A0170048         00170          FPREAL    D1,A0  RESULTAAT IN ASCII-BUFFER ZETTEN
023020 A01B0008         00180          FPEND     A0  SLUIT DE ASCII-BUFFER
023024 F050             00190          DISP      (A0)  ASCII-BUFFER-INHOUD OP SCHERM WEERGEVEN
023026 4E75             00200          RTS
023028                  00202 *
023028                  00204 * FORMAAT EN ASCII-BUFFER DEFINIEREN
023028                  00206 *
023028 434C27643020697374 00210 FM:       FORMAT    CL'd0 is REAL'XS5D.2D
023040    :00000028     00220 BUFFER:    DS      .B 40
023068                  00230          END
```

17

```
68000 Assembler  v2.1 fmul.2 13:35:55 10/10/85 page:00001

000000                  00006          LIST
000000                  00010 ************************************
000000                  00020 * FLOATING-POINT-VERMENIGVULDIGING *
000000                  00030 * MET GEFORMATTEERDE WEERGAVE      *
000000                  00040 ************************************
000000                  00050 *
023000    =00023000     00060          ORG       $23000  PROGRAMMABEGIN
023000    =00023000     00070          ENTRY     START  BELANGRIJK VOOR 'run'-KOMMANDO
023000                  00080 *DEBUG 100  STERRETJE VOOR DEBUGGER WEGHALEN
023000 41FA006A         00090 START:   LEA     .L BOODSCHAP(PC),A0  WIJS NAAR EERSTE GETALLENREEKS
023004 F050             00100          DISP      (A0)  GEEF EERSTE GETALLENREEKS WEER
023006 F07A0089         00110 START:   DISP      OP1(PC)  GEEF TWEEDE GETALLENREEKS WEER
02300A 41FA0038         00120          LEA     .L BUFFER(PC),A0 WIJS NAAR DE INGANGSBUFFER
02300E F250             00130          INPUT     (A0)  LEES OP1 IN DE INGANGSBUFFER
023010 A00A0400         00140          FINPUT    (A0),D0  ZET OP1 ALS FP-GETAL IN 'd0'
023014 F07A008C         00150          DISP      OP2(PC)  GEEF DERDE GETALLENREEKS WEER
023018 41FA002A         00160          LEA     .L BUFFER(PC),A0  WIJS NAAR DE ASCII-BUFFER
02301C F250             00170          INPUT     (A0)  LEES OP2 IN DE INGANGSBUFFER (ASCII)
02301E A00A0481         00180          FINPUT    (A0),D1  ZET OP2 ALS FP-GETAL IN 'd1'
023022 A0020001         00190          FMUL      D0,D1  VERMENIGVULDIG OP1 EN OP2
023026 A0000F01F851EB85 00200          FADD      #5.005,D1  AFRONDEN
02302E 41FA0014         00210          LEA     .L BUFFER(PC),A0  WIJS NAAR DE UITGAVE-BUFFER
023032 A0160E88007D     00220          FPINIT    FM(PC),A0  INITIALISEER DE FORMATTERINGSBUFFER
023038 A0170048         00230          FPREAL    D1,A0  SCHRIJF RESULTAAT IN FORMATTERINGSBUFFER
02303C A01B0008         00240          FPEND     A0  FORMATTERING AFSLUITEN
023040 F050             00250          DISP      (A0) INHOUD VAN FORMATTERINGSBUFFER WEERGEVEN
023042 4E75             00260          RTS
023044    :00000028     00270 BUFFER:    DS      .B 40  INGANGSBUFFER
02304C                  00280 * EERSTE GETALLENREEKS
02304C 0D0A3D3D3D466C6965 00290 BOODSCHAP:DC      .B 13,10,'=floating point vermenigvuldiging=',13,10,0
023091                  00300 * TWEEDE GETALLENREEKS
023091 0D0A6F706572616E64 00310 OP1:       DC      .B 13,10,'operand 1: ? ',0
0230A2                  00320 * DERDE GETALLENREEKS
0230A2 0D0A6F706572616E64 00330 OP2:       DC      .B 13,10,'operand 2: ? ',0
0230B3                  00340 * FORMATTERINGSBUFFER
0230B3 434C2752657375 6C74 00350 FM:       FORMAT    CL'resultaat:'4D.2D'.'
0230CB                  65535          END
```

Programmavoorbeeld 4

Figuur 17 toont nog een ander voorbeeld: de vermenigvuldiging van twee drijvende-komma-getallen. Ook hier wordt duidelijk, hoe gemakkelijk het is om van de format-precompiler gebruik te maken. Aangezien dit demonstratieprogramma in wezen slechts een voortborduren inhoudt op de vorige twee demonstratieprogramma's, zullen we er hier niet nader op ingaan.

Programmavoorbeeld 5

Als u uw EC-68K voor de eerste keer inschakelt, dan moet de klok nog worden gelijkgezet. Met behulp van het programma van figuur 18a kunt u de klok gelijkzetten en de datum opgeven. Allereerst nemen we dit demonstratieprogramma op in het menu van de monitor (regels 43...46). De ingave van de dag vindt in de regels 50...120 plaats. Het programma kontroleert of er een zinvolle dagparameter (1...31) is ingegeven.
Vervolgens wacht het programma op de ingave van de maand (regels 140...210). Ook hier wordt gekontroleerd of het ingegeven maandnummer (1...12) korrekt is. Na de opgave van de maand moet het jaar worden ingegeven. De mogelijkheden

Figuur 16. Dit programma zet een hex-getal om in een decimaal getal van het type "real", en toont dit geformatteerd op het beeldscherm. Links van de decimale punt staan 3 cijfers en rechts ervan twee.

Figuur 17. Dit programma realiseert de vermenigvuldiging van twee drijvende-komma-getallen. De getallen (operanden) zijn via het toetsenbord ingegeven. Het resultaat wordt op het beeldscherm weergegeven. Het getallenformaat is gelijk aan dat in figuur 16.

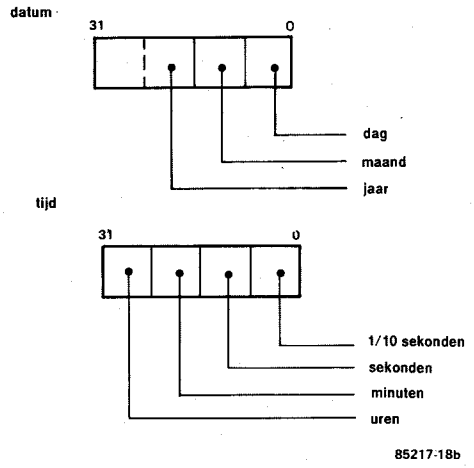
Figuur 18a. Met dit programma kan men de klok op de CPU-kaart gelijkzetten en de datum instellen.

Figuur 18b. Het dataformaat voor de datum en de tijdsaanduiding van de klok. Voor het gelijkzetten en instellen van de klok moeten twee lange woorden in het klok-IC worden geschreven.

18a

```
68000 Assembler  v2.1 set-clock.2 13:39:11 10/10/85 page:00001

000000                  00010          LIST
000000                  00012 ***************************
000000                  00014 *  'Real Time Clock'     *
000000                  00016 * KLOK EN DATUM INSTELLEN *
000000                  00018 ***************************
023800    =00023800     00020          ORG       $23800
023800    =00023818     00040          ENTRY     START
023800 4C69623A7365745F63 00043          DC      .B 'Lib:set_clock
023814 00000004         00046          DR      .L START
023818 F07A00F2         00050 START:   DISP      DAY(PC)
02381C 610000D4         00060          BSR       KBD_INPUT_D0
023820 0280000000FF     00070          ANDI    .L #$FF,D0
023826 0C400001         00080          CMPI      #1,D0
02382A 6D00FFEC         00090          BLT       START
02382E 0C40001F         00100          CMPI      #31,D0
023832 6E00FFE4         00110          BGT       START
023836 2200             00120          MOVE    .L D0,D1
023838                  00130 *
023838 F07A00F0         00140 START1:  DISP      MONTH(PC)
02383C 610000B4         00150          BSR       KBD_INPUT_D0
023840 0280000000FF     00160          ANDI    .L #$FF,D0
023846 0C400001         00170          CMPI      #1,D0
02384A 6D00FFEC         00180          BLT       START1
02384E 0C40000C         00190          CMPI      #12,D0
023852 6E00FFE4         00200          BGT       START1
023856 2400             00210          MOVE    .L D0,D2
023858                  00230 *
023858 F07A00F0         00231 START2:  DISP      YEAR(PC)
02385C 61000094         00232          BSR       KBD_INPUT_D0
023860 0280000FFFF      00233          ANDI    .L #$FFFF,D0
023866 0C4007C0         00234          CMPI      #1984,D0
02386A 6D00FFEC         00235          BLT       START2
02386E 0C4007CF         00236          CMPI      #1999,D0
023872 6E00FFE4         00237          BGT       START2
023876 2600             00238          MOVE    .L D0,D3
023878                  00240 *
023878 E142             00242          ASL       #8,D2
02387A 4843             00244          SWAP      D3
02387C 8441             00245          OR        D1,D2
02387E 8682             00246          OR      .L D2,D3
023880 FB83             00247          SETDATE .L D3
023882                  00248 *
023882                  00249 *
023882 F07A00EA         00250 START3:  DISP      SECOND(PC)
023886 6100006A         00251          BSR       KBD_INPUT_D0
02388A 0280000000FF     00252          ANDI    .L #$FF,D0
023890 0C400000         00253          CMPI      #0,D0
023894 6D00FFEC         00254          BLT       START3
023898 0C40003B         00255          CMPI      #59,D0
02389C 6E00FFE4         00256          BGT       START3
0238A0 2200             00257          MOVE    .L D0,D1
0238A2                  00258 *
0238A2 F07A00EE         00259 START4:  DISP      MINUTE(PC)
0238A6 6100004A         00260          BSR       KBD_INPUT_D0
0238AA 0280000000FF     00261          ANDI    .L #$FF,D0
0238B0 0C400000         00262          CMPI      #0,D0
0238B4 6D00FFEC         00263          BLT       START4
0238B8 0C40003B         00264          CMPI      #59,D0
0238BC 6E00FFE4         00265          BGT       START4

68000 Assembler  v2.1 set-clock.2 13:40:28 10/10/85 page:00002

0238C0 2400             00266          MOVE    .L D0,D2
0238C2                  00267 *
0238C2 F07A00F0         00268 START5:  DISP      HOUR(PC)
0238C6 6100002A         00269          BSR       KBD_INPUT_D0
0238CA 0280000000FF     00270          ANDI    .L #$FF,D0
0238D0 0C400000         00271          CMPI      #0,D0
0238D4 6D00FFEC         00272          BLT       START5
0238D8 0C400017         00273          CMPI      #23,D0
0238DC 6E00FFE4         00274          BGT       START5
0238E0 2600             00275          MOVE    .L D0,D3
0238E2 E181             00276          ASL     .L #8,D1
0238E4 4842             00277          SWAP      D2
0238E6 8481             00287          OR      .L D1,D2
0238E8 4843             00297          SWAP      D3
0238EA E183             00307          ASL     .L #8,D3
0238EC 8682             00297          OR      .L D2,D3
0238EE F983             00307          SETTIME .L D3
0238F0                  00317 *
0238F0                  00327 *
0238F0 4E75             00337          RTS
0238F2                  00347 *
0238F2 41FA000E         00357 KBD_INPUT_LEA    .L BUFFER(PC),A0
0238F6 F250             00367          INPUT     (A0)
0238F8 A00A0600         00377          FINPUT    (A0)+,D0
0238FC A0050000         00387          FIX       D0,D0
023900 4E75             00298          RTS
023902                  00300 *
023902                  00310 *
023902    :0000000A     00320 BUFFER:  DS      .B 10
02390C 0D0A67656265206465 00330 DAY:     DC      .B 13,10,'Geef de dag in (1...31): ',0
02392A 0D0A67656265206465 00340 MONTH:   DC      .B 13,10,'Geef de maand in (1...12): ',0
02394A 0D0A67656265206461 00350 YEAR:    DC      .B 13,10,'Geef het jaar in (1984...1999): ',0
02396E 0D0A0A676562526864 00360 SECOND:  DC      .B 13,10,10,'Geef de sekonden in (0...59): ',0
023992 0D0A67656265206469 00370 MINUTE:  DC      .B 13,10,'Geef de minuten in (0...59): ',0
0239B4 0D0A67656265206469 00380 HOUR:    DC      .B 13,10,'Geef de uren in (0...24): ',0
0239D6                  65535          END
```

b



85217-18b

variëren van 1984 tot en met 1999. (U kunt dus voorlopig vooruit...) U hoeft geen rekening te houden met schrikkeljaren; dat doet de klok zelf wel voor u. (Met de zomer- en wintertijd ligt dat anders.)
In de programmaregels 240...247 maken we een lang woord dat is samengesteld uit het jaar (2 bytes), de maand (1 byte) en de dag (1 byte). De instruktie "SETDATE .L" schrijft de opgegeven datum in de klok.
En dan wordt de klok gelijkgezet. De klok geeft het uur, het aantal minuten, het aantal sekonden en de tienden van sekonden weer. In de regels 250...257 staat het programma voor de ingave van het aantal sekonden. Het programma aksepteert de waarden 0...59. Vervolgens (regels 259...266) het programma voor het ingeven van het aantal minuten (0...59). Het programma in de regels 268...274 is er voor de ingave van het uur. Nadat alles volledig is ingegeven, stelt het programma uit de opgegeven aantallen een lang woord samen (regels 275...297). De instruktie "settime" laadt de klok met de gelijkzetdata.
Het algemene ingaveprogramma staat in de regels vanaf nummer 357. Aan het eind van het programma staan alle getallenstrings die door het programma geleverd worden. Figuur 18b toont het weergaveformaat voor de instelling van de klok.

Programmavoorbeeld 6 — Dump-hulpprogramma —

Figuur 19 toont een hulpprogramma, waarmee men de geheugeninhoud in hexadecimale vorm en als ASCII-data weergeven kan worden. Aan het begin vraagt het programma om een startadres. Zodra dit startadres is opgegeven, verzendt het dump-programma 128 bytes naar het beeldscherm en wacht op een willekeurige ingedrukte toets. Zodra een toets wordt ingedrukt, herhaalt de gang van zaken zich. Door het indrukken van de toets <esc> kan men het dump-programma weer verlaten.

Programmavoorbeeld 7 De EC-68K-calculator kraakt uitdrukkingen tussen haakjes

Ter afsluiting van dit artikel schrijven (en beschrijven) we een klein programma waarmee de EC68K als een soort rekenmachine kan worden gebruikt. We maken daarbij uiteraard de nodige vereenvoudigingen, teneinde het programma voor beginners overzichtelijk te houden. We beperken ons dan ook tot de vier elementaire rekenoperaties optellen, aftrekken, delen en vermenigvuldigen. Daar staat

19

```
68000 Assembler  v2.1 dump.1 13:50:31 10/10/85 page:00001

000000                  00006          LIST
000000                  00010 ********************
000000                  00020 * HEX-DUMP UTILITY *
000000                  00030 ********************
000000                  00040 *
023000    =00023000     00050          ORG       $23000
023000    =00023018     00060          ENTRY     START
023000 4C69623A6865786475 00070          DC     .B 'Lib:hexdump
023014 00000004         00080          DR     .L START
023018 43FA009F         00090 START:   LEA    .L HEXD(PC),A1
02301C F051             00100          DISP      (A1)
02301E 7000             00110          MOVEQ     #0,D0
023020 2040             00120          MOVEA  .L D0,A0
023022 F488             00130          INHEX  .L A0
023024 4281             00140 LOOP1:   CLR    .L D1
023026 4282             00150          CLR    .L D2
023028 123C0008         00160          MOVE   .B #128/16,D1
02302C 740F             00170 LOOP:    MOVEQ  .L #16-1,D2
02302E 43FA0074         00180          LEA       CRLF(PC),A1
023032 F051             00190          DISP      (A1)
023034 F688             00200          OUTHEX .L A0
023036 43FA007D         00210          LEA    .L COLON(PC),A1
02303A F051             00220          DISP      (A1)
02303C 1018             00230 HEXLOOP: MOVE   .B (A0)+,D0
02303E F600             00240          OUTHEX .B D0
023040 43FA0065         00250          LEA       SPACE(PC),A1
023044 F051             00260          DISP      (A1)
023046 51CAFFF4         00270          DBF       D2,HEXLOOP
02304A 43FA0058         00280          LEA       CRLF(PC),A1
02304E F051             00290          DISP      (A1)
023050 4282             00300          CLR    .L D2
023052 41E8FFF0         00310          LEA    .L -16(A0),A0
023056 740F             00320          MOVEQ  .L #16-1,D2
023058 43FA0052         00330          LEA    .L SP8(PC),A1
02305C F051             00340          DISP      (A1)
02305E 43FA0055         00350          LEA    .L COLON(PC),A1
023062 F051             00360          DISP      (A1)
023064 1018             00370 ASCLOOP: MOVE   .B (A0)+,D0
023066 0C000020         00380          CMPI   .B #' ',D0
02306A 6D000028         00390          BLT       NOASCII
02306E 0C00007F         00400          CMPI   .B #$7F,D0
023072 6E000020         00410          BGT       NOASCII
023076 F300             00420          OUTCHR .B D0
023078 43FA002F         00430          LEA    .L SP2(PC),A1
02307C F051             00440          DISP      (A1)
02307E 51CAFFE4         00450 ASCII:   DBF       D2,ASCLOOP
023082 51C9FFA8         00460          DBF       D1,LOOP
023086 F100             00470          INCHR  .B D0
023088 43FA001A         00480          LEA       CRLF(PC),A1
02308C F051             00490          DISP      (A1)
02308E 6600FF94         00500          BNE       LOOP1
023092 4E75             00510          RTS
023094 103C002E         00520 NOASCII: MOVE   .B #'.',D0
023098 F300             00530          OUTCHR .B D0
02309A 43FA000D         00540          LEA    .L SP2(PC),A1
02309E F051             00550          DISP      (A1)
0230A0 6000FFDC         00560          BRA       ASCII
0230A4 0D0A00           00570 CRLF:    DC     .B 13,10,0
0230A7 2000             00580 SPACE:   DC     .B ' ',0


68000 Assembler  v2.1 dump.1 13:51:51 10/10/85 page:00002

0230A9 202000           00590 SP2:     DC     .B '  ',0
0230AC 20202020202020202000 00600 SP8:     DC     .B '        ',0
0230B5 3A202000         00610 COLON:   DC     .B ': ',0
0230B9 0D0A656E7465722073 00620 HEXD:     DC     .B 13,10,'enter start address: $',0
0230D2                  65535          END
```

echter tegenover dat willekeurige rekenkundige uitdrukkingen tussen haakjes moeten kunnen worden aangepakt, zoals bijvoorbeeld:

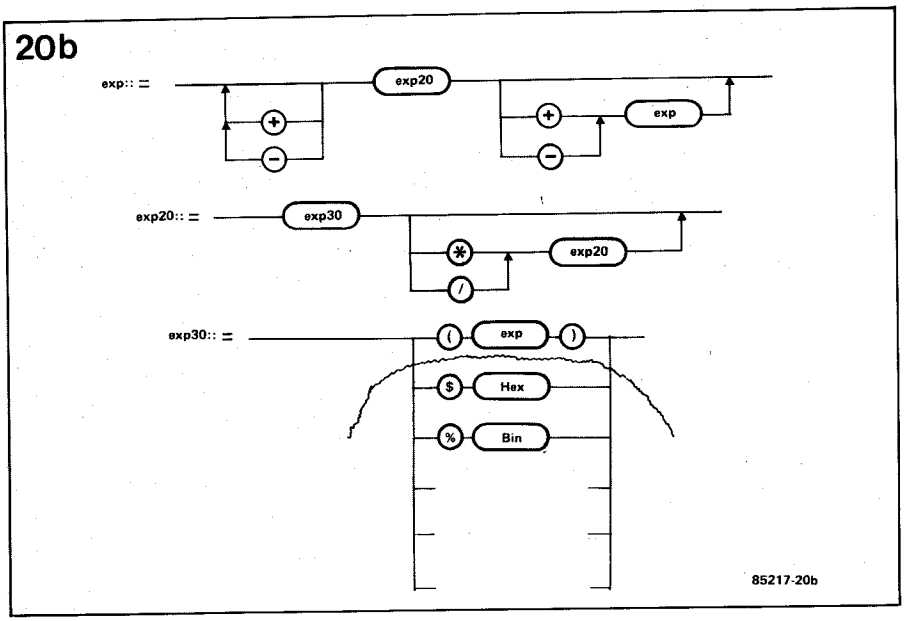
Ø.23*(1.17+(23.1/3.14)*Ø.5)

Voordat een programma, dat deze haakjestermen berekent, geschreven kan worden, moeten we eerst een stroomschema tekenen. Figuur 20b laat een stroomschema zien, dat u waarschijnlijk wel bekend zal voorkomen van de syntax-beschrijving van Pascal-compilers. Uit het stroomschema blijkt overduidelijk dat de afzonderlijke deelprogramma's "exp", "exp20" en "exp30" zichzelf aanroepen. Het vakjargon spreekt van "re-entrant" of "rekursief". Vergelijk het programma van figuur 20a met het stroomschema van figuur 20b, en u zult snel begrijpen hoe de computer het teken van de operand onderzoekt en hoe tussenresultaten op de stack worden gezet, ten behoeve van latere verwerking.

Zoals al is opgemerkt kan het syntaxschema "exp30" nog verder worden uit-

20a

```
68000 Assembler  v2.1 ec68k-calc.1 14:24:52 10/10/85 page:00001

000000                  00002          LIST
000000                  00003 **********************
000000                  00004 * EC-68K REKENMACHINE *
000000                  00006 **********************
000000                  00008 *
023000    =00023000     00010          ORG       $23000
023000    =00023000     00020          ENTRY     START
023000    =00000064     00025          DEBUG     100
023000 F07A0154         00030 START:   DISP      TEXT1(PC)
023004 41FA00D0         00040 LOOP:    LEA       BUF(PC),A0
023008 F250             00050          INPUT     (A0)
02300A 61000020         00060          BSR       EXP
02300E 41FA00C6         00070          LEA       BUF(PC),A0
023012 A0160E8000AA     00080          FPINIT    FM(PC),A0
023018 A0000F02F851EB85 00085          FADD      #5.005,D2
023020 A0170008         00090          FPREAL    D2,A0
023024 A01B0008         00100          FPEND     A0
023028 F050             00110          DISP      (A0)
02302A 60D8             00120          BRA    .S LOOP
02302C 61000032         00130 EXP:     BSR       EXP20
023030 0C10002B         00140          CMPI   .B #'+',(A0)
023034 6708             00150          BEQ    .S EXP2
023036 0C10002D         00160          CMPI   .B #'-',(A0)
02303A 6712             00170          BEQ    .S EXP3
02303C 4E75             00180          RTS
02303E 2F02             00190 EXP2:    MOVE   .L D2,-(A7)
023040 5248             00200          ADDQ      #1,A0
023042 6100FFE8         00210          BSR       EXP
023046 221F             00220          MOVE   .L (A7)+,D1
023048 A0000042         00230          FADD      D1,D2
02304C 4E75             00240          RTS
02304E 2F02             00250 EXP3:    MOVE   .L D2,-(A7)
023050 5248             00260          ADDQ      #1,A0
023052 6100FFD8         00270          BSR       EXP
023056 221F             00280          MOVE   .L (A7)+,D1
023058 A0010081         00290          FSUB      D2,D1
02305C 2401             00300          MOVE   .L D1,D2
02305E 4E75             00310          RTS
023060 61000032         00320 EXP20:   BSR       EXP30
023064 0C10002A         00330          CMPI   .B #'*',(A0)
023068 6708             00340          BEQ    .S EXP202
02306A 0C10002F         00350          CMPI   .B #'/',(A0)
02306E 6712             00360          BEQ    .S EXP203
023070 4E75             00370          RTS
023072 2F02             00380 EXP202:  MOVE   .L D2,-(A7)
023074 5248             00390          ADDQ      #1,A0
023076 6100FFE8         00400          BSR       EXP20
02307A 221F             00410          MOVE   .L (A7)+,D1
02307C A0020042         00420          FMUL      D1,D2
023080 4E75             00430          RTS
023082 2F02             00440 EXP203:  MOVE   .L D2,-(A7)
023084 5248             00450          ADDQ      #1,A0
023086 6100FFD8         00460          BSR       EXP20
02308A 221F             00470          MOVE   .L (A7)+,D1
02308C A0030081         00480          FDIV      D2,D1
023090 2401             00490          MOVE   .L D1,D2
023092 4E75             00500          RTS
023094 0C100028         00510 EXP30:   CMPI   .B #'(',(A0)
023098 6706             00520          BEQ    .S EXP31

68000 Assembler  v2.1 ec68k-calc.1 14:26:13 10/10/85 page:00002

02309A A00A0602         00530          FINPUT    (A0)+,D2
02309E 4E75             00540          RTS
0230A0 5248             00550 EXP31:   ADDQ      #1,A0
0230A2 6100FF88         00560          BSR       EXP
0230A6 0C100029         00570          CMPI   .B #')',(A0)
0230AA 6706             00580          BEQ    .S EXP32
0230AC F07A0008         00590          DISP      ERROR1(PC)
0230B0 4E75             00600          RTS
0230B2 5248             00610 EXP32:   ADDQ      #1,A0
0230B4 4E75             00620          RTS
0230B6 0D0A4B65696E652029 00630 ERROR1:  DC     .B 13,10,'geen )',0
0230C8 434C27416E74776F72 00640 FM:      FORMAT    CL'antwoord='4D.2DCLCL
0230D6    :00000000     00650 BUF:     DS     .B 128
023156 0C45632D36386B2052 00660 TEXT1:   DC     .B 12,'EC-68K rekenmachine',13,10,0
023168                  65535          END
```

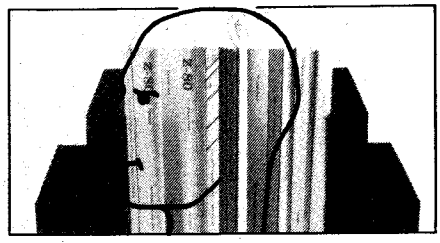


Figuur 19. Een hexadecimaal- en ASCII-dumpprogramma.

Figuur 20a en 20b. De EC-68K is in staat om rekenkundige uitdrukkingen die tussen haakjes staan (hier uitsluitend de vier elementaire bewerkingen) aan te pakken. Het assembler-programma is gebaseerd op het aangegeven syntax-stroomschema. De EC-68K-assembler is dermate krachtig dat men het syntax-stroomschema zonder potlood en papier zo in een assembler-programma kan vertalen.

gebreid. Het is dan bijvoorbeeld mogelijk om met hexadecimale of binaire getallen te werken. Hier (figuur 20) is dit soort getallen buiten beschouwing gelaten.

In dit artikel hebben we laten zien, hoe eenvoudig het is om de 68000 te programmeren. Talrijke voorbeeldprogramma's, die direkt op de EC68K kunnen worden uitgevoerd, laten zien welke mogelijkheden de gebruiker heeft. Wie op welke wijze dan ook met de 68000 werkt, vergeet het computer-verleden en schept nieuwe perspektieven voor de toekomst. De EC-68K vormt een goedkope en goede springplank voor de grote sprong naar de wereld van de 16/32-bitters.



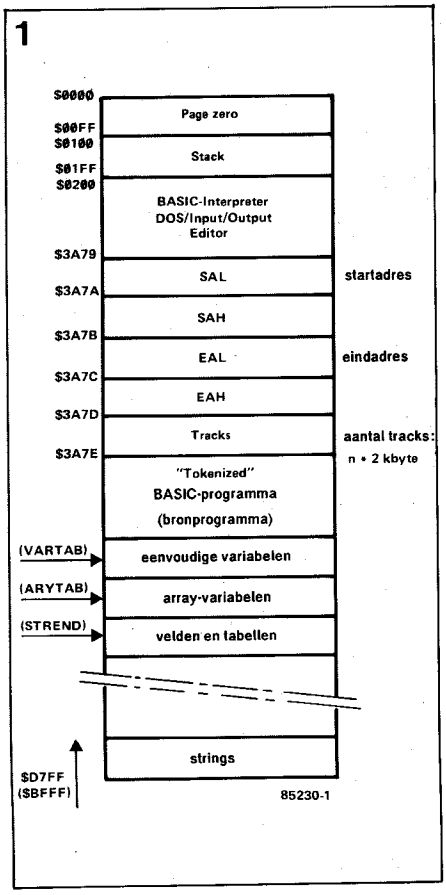
BASIC intern

Wie sinds jaar en dag zijn computer in BASIC heeft geprogrammeerd en graag zou willen weten hoe dat in zijn werk gaat, vraagt zich beslist wel eens af hoe zijn BASIC-interpreter funktioneert, of hoe een BASIC-programma in het werkgeheugen is opgeslagen. In dit artikel laten we zien, op basis van de industriestandaard Microsoft-BASIC en een 6502-computer (Octopus 65), hoe BASIC er "achter de coulissen" uitziet. Vele overzichtelijke tabellen laten in weinig woorden zien hoe een BASIC-interpreter intern is opgebouwd, en hoe een BASIC-source-programma door de computer verwerkt wordt.

De geheugenindeling

Figuur 1 toont de geheugenindeling van de Octopus 65, nadat het DOS en de BASIC-interpreter van de diskette geladen zijn. Voor een Commodore-computer geldt een soortgelijke geheugenindeling. Het enige verschil is, dat de BASIC-interpreter bij de Octopus 65 vrijwel "onderin" het geheugen zit (vanaf $0200) en bij de Commodore vrijwel "bovenin" het geheugen (vanaf $C000).

Bij de Octopus 65 en bij Commodore-computers is pagina nul voor de adreswijzers (pointers), de ingangsbuffer en voor andere tijdelijke data gereserveerd. Pagina 1 is gereserveerd voor de systeemstack en voor de registratie van parameters tijdens FOR-NEXT-programmalussen en tijdens de uitvoering van GOSUB-instrukties. De BASIC-interpreter, het disk operating system (DOS) en de BASIC-full-screen-editor nemen samen ongeveer 14 K geheugen van de Octopus 65 in beslag. Het geheugenbereik $3A79.. ..$D7FF (ca. 39 K) is bestemd voor het BASIC-source-programmma en voor de tabellen voor de variabelen en de arrays. Wie de in Elektuur beschreven grafische kleurenkaart op zijn computer heeft aangesloten, kan uitsluitend het geheugenbereik t/m $BFFF gebruiken, aangezien het geheugenbereik $C000...$CFFF voor de grafische software is gereserveerd. Nadat het BASIC-programma met een RUN-kommando is gestart, bouwt de BASIC-interpreter vanaf het einde (in het geheugen) van het source-programma de



Figuur 1. De geheugenindeling van de Octopus 65 nadat de BASIC-interpreter en DOS van de diskette zijn geladen. De variabelen worden in het geheugen direkt na het BASIC-source-programma geschreven.

2

```
Label    Dec.   Hex.   Sample  Comment

           0   0000     4C    ;JMP instruction
           2  (0002)   0474    ;address: BASIC warmstart
           3   0003     4C    ;JMP instruction
           4  (0004)   0ACC    ;address: print string from (YA) until zero
           6   0006    1047   ;address for USR: convert floating to integer
           8   0008    20B2   ;address for USR: print result
CHARAC    10   000A           ;00=INPUT/READ, other is GET's char.
                              ;work area for AND/OR, low byte
                              ;1st delimiter in string set up
                              ;odd/even flag in power calculation/EXP
ENDCHR    11   000B           ;flag in tokenise routine: REM=0
                              ;work area for AND/OR, hi byte
                              ;2nd delimiter in string set up
COUNT     12   000C           ;length of current line in tokenisation
                              ;tokenconverter in tokenise inputbuffer
                              ;invertervalue for AND/OR; OR=FF, AND=00
                              ;number of subscripts in current array
DIMFLG    13   000D           ;flag:DIM<>0, LET/get value of variable=0
VALTYP    14   000E     FF    ;variable type: FF=string, 00=numeric
INTFLG    15   000F     00    ;numeric type: 80=integer, 00=floating
GARBFL    16   0010           ;quote flag in LIST
                              ;garbage collect counter
DORES     16   0010           ;DATA-flag in tokenise routine
SUBFLG    16   0010           ;if non zero, no integers or arrays allowed
          17   0011           ;unused
INPFLG    18   0012     00    ;flag: 00=INPUT, 40=GET, 98=READ
DOMASK    19   0013           ;save area for flag in evaluate expression
TANSGN    19   0013           ;3rd/4th quadrant flag in goniometric func.
CNTWFL    20   0014     00    ;flag: inhibit output if MSbit set
NULLS     21   0015     00    ;number of nulls to send after CR
TRMPOS    22   0016           ;current column number of cursor
LINWID    23   0017     84    ;maximum column number of cursor
NCMWID    24   0018     70    ;maximum number of input characters
LINNUM    25  (0019)           ;integer address for GOTO or GOSUB
POKER     25  (0019)           ;integer address for PEEK and POKE
BUF       27   001B           ;start of BASIC's input buffer
           !    !             ;the buffer is 70 characters long
          97   0061           ;end of input buffer
          98   0062     00    ;terminator for input buffer
TEMPPT    99   0063           ;current descr. stack pointer
LASTPT   100  (0064)  0066    ;pointer to last descr. on descr. stack
TEMPST   102   0066           ;string descriptor stack
           !    !             ;three strings deep
         110   006E           ;
INDEX    111   006F           ;save area for offset in array
INDEX1   111  (006F)          ;temp. pointer for memory move or string
                              ;temporary pointer in rechain
                              ;return address save in evaluate expression
                              ;pointer in search for string arrays
                              ;pointer in space allocation for string
INDEX2   113  (0071)          ;pointer for number movements and comparisons
RESHO    115   0073           ;FLP accu #3 (fraction only)
           !    !             ;for multiply and divide
         119   0077           ;used to store intermediate result
ADDEND   117  (0075)          ;intermediate result in array size calculation
TXTTAB   120  (0078)  3A7E    ;pointer: start of BASIC program
VARTAB   122  (007A)          ;pointer: start of variables/end of program
ARYTAB   124  (007C)          ;pointer: start of arrays/end of variables
STREND   126  (007E)          ;pointer: start of free RAM/end of arrays
FRETOP   128  (0080)          ;pointer: start of strings (moving down)
FRESPC   130  (0082)          ;utility pointer for string allocation
MEMSIZ   132  (0084)  BFFF    ;pointer: top of RAM
CURLIN   134  (0086)          ;current BASIC line number $87=FF: direct mode
OLDLIN   136  (0088)          ;BASIC line number for CONT
OLDTXT   138  (008A)          ;CHRGET pointer for CONT
DATLIN   140  (008C)          ;line number of current DATA line
DATPTR   142  (008E)          ;pointer to current DATA-value
INPPTR   144  (0090)          ;save area for CHRGET pointer with
                              ;INPUT, READ and GET
VARNAM   146   0092           ;current variable name, 1st character
VARNAM   147   0093           ;current variable name, 2nd character
         148   0094           ;number conversion: save for table index (Y)
VARPNT   148  (0094)          ;pointer to current variable
         149   0095           ;index save in LIST
FORPNT   150  (0096)          ;variable name for FOR...NEXT
                              ;pointer for number move from accu to memory
VARTXT   152  (0098)          ;saved CHRGET pointer during READ/GET/INPUT
OPMASK   154   009A           ;symbol check: bits 0, 1, 2 are <,= and >
DEFPNT   155  (009B)          ;pointer: temporary storage for FN, DEF, TAN
GRBPNT   155  (009B)          ;source pointer in garbage collect
TEMPF3   155   009B           ;9B-A0 also used as save area for FLP accu1
DSCPNT   157  (009D)          ;pointer: temporary storage
                              ;temporary pointer in reserve string space
         159   009F           ;unused
FOUR6    160   00A0     03    ;stepsize in garbage collect
```

2

```
 Label   Dec.   Hex.   Sample  Comment

JMPER    161   00A1     4C   ;JMP-instruction for functions
         162  (00A2)          ;pointer: address of arithmetic function
SIZE     162   00A2          ;step size in garbage collect
OLDOV    163   00A3          ;save area for rounding byte of FLP acc1
TEMPF1   164   00A4          ;A4-A9 also work area for polynome result
HIGHDS   165  (00A5)         ;temparary destination pointer for memory move
                             ;temporary pointer to variable in array search
HIGHTR   167  (00A7)         ;temporary source pointer for memory move
                             ;pointer in create a variable
                             ;search pointer in array values
         169   00A9          ;end of work area for polynome result
LOWDS    170   00AA          ;bit counter in array size calculation
DECCNT   170   00AA          ;string conv: # of digits after decimal point
                             ;number conv: exponent after E or:
                             ;number conv: position of period
TENEXP   171   00AB          ;string conv: exponent base 10
                             ;number conv: correct exponent after E
DPTFLG   172   00AC          ;string conv: period flag in MSbit
                             ;string conv: 2nd period flag in bit 6
LOWTR    172  (00AC)         ;pointer in LET or search for linenumber
                             ;pointer in search for variable
GRBTOP   172  (00AC)         ;destination pointer in garbage collect
EXPSGN   173   00AD          ;string conv: neg. exp. flag in MSbit
FACEXP   174   00AE          ;exponent of FLP-accu #1
                             ;length of string (1st decriptor byte)
FACHO    175   00AF          ;MSB mantissa of FLP-accu, bit7 always 1
                             ;pointer to string text lo (2nd descriptor)
                             ;hi++ byte of integer (if 4 byte integer)
FACMOH   176   00B0          ;mantissa of FLP-accu #1
                             ;pointer to string text hi (3rd descriptor)
                             ;hi+ byte of integer (if 4 byte integer)
FACMO    177   00B1          ;mantissa of FLP-accu #1
                             ;hi byte of integer
DSCTMP   177  (00B1)         ;pointer to string descriptor
FACLO    178   00B2          ;mantissa of FLP-accu #1, LSB
                             ;lo byte of integer
FACSGN   179   00B3          ;sign of FLP-accu #1 in bit 7
DEGREE   180   00B4          ;number of terms of polynome
SGNFLG   180   00B4          ;sign of string number to be converted
BITS     180   00B4          ;overflow byte on normalizing FLP-accu #1
         181   00B5          ;unused
ARGEXP   182   00B6          ;exponent of FLP-accu #2
ARGHO    183   00B7          ;MSB mantissa of FLP-accu, bit7 always 1
ARGMOH   184   00B8          ;mantissa of FLP-accu #2
ARGMO    185   00B9          ;mantissa of FLP-accu #2
ARGLO    186   00BA          ;mantissa of FLP-accu #2, LSB
ARGSGN   187   00BB          ;sign of FLP-accu #2 in bit 7
ARISGN   188   00BC          ;signcomparison of FLP-acc:0=eq. FF=opp.
STRNG1   188  (00BC)         ;pointer to start of string in set up
FACOV    189   00BD          ;rounding byte for FLP-accu #1
BUFPTR   190   00BE          ;save for output index in tokenise routine
CURTOL   190  (00BE)         ;size of current array in bytes
STRNG2   190  (00BE)         ;pointer to end of string in set up string
POLYPT   190  (00BE)         ;pointer to constants in series evaluation
FBUFPT   190  (00BE)         ;number conv: save for index in string
CHRGET   192   00C0          ;BASIC's CHRGET routine, see $????-$YYYY
          !     !            ;in coldstart routine
CHRGOT   198   00C6          ;BASIC's CHRGOT entry
TXTPTR   199  (00C7)   001B  ;CHRGET's pointer
          !     !            ;
CHRRTS   215   00D7          ;end of CHRGET/CHRGOT
RNDX     216   00D8          ;current RND number seed, exponent
         217   00D9          ;current RND number seed, MSB mantissa
         218   00DA          ;current RND number seed, mantissa
         219   00DB          ;current RND number seed, mantissa
         220   00DC          ;current RND number seed, LSB mantissa
LOFBUF   255   00FF          ;output area for number to string conversion
FBUFFR   256   0100          ;work area for number to string conversions
                !            ;string is always terminated by a zero byte
         269   010D          ;end of work area for string conversion
         310   0136          ;theoretical bottom of BASIC stack
                !            ;
STKEND   511   01FF          ;top of BASIC and machine stack
```

Figuur 2. De BASIC-interpreter gebruikt pagina nul ($0000...$00FF) en pagina 1 ($0100...$01FF) om data, adreswijzers (=pointers) en vlaggen in op te slaan. Alle geheugenplaatsen zijn hier in één tabel gegeven.

tabellen voor de variabelen op. De BASIC-interpreter deelt de in het programma gebruikte variabelen in meerder groepen in. Hierover later meer.

Page Ø en page 1

Figuur 2 toont in tabelvorm de indeling van "page Ø" en "page 1". Deze tabel bestaat uit vijf velden:

* Label
* Decimaal label-adres
* Hexadecimaal label-adres
* Geheugeninhoud
* Opmerkingen (comment)

3

```
                        .BA $0200
                        ;*********************************
                        ;*                               *
                        ;* Table of addresses of main    *
                        ;* keyword routines.             *
                        ;* All are -1 because they are   *
                        ;* reached through a RTS ins-    *
                        ;* truction.                     *
                        ;*                               *
                        ;*********************************
0200- 29 08     STMDSP  .SI END-1          ;address for END
0202- 47 07             .SI FOR-1          ;address for FOR
0204- 4A 0C             .SI NEXT-1         ;address for NEXT
0206- F8 08             .SI DATA-1         ;address for DATA
0208- 2B 0B             .SI INPUT-1        ;address for INPUT
020A- 23 0F             .SI DIM-1          ;address for DIM
020C- 57 0B             .SI READ-1         ;address for READ
020E- A5 09             .SI LET-1          ;address for LET
0210- A5 08             .SI GOTO-1         ;address for GOTO
0212- 6C 08             .SI RUN-1          ;address for RUN
0214- 28 09             .SI IF-1           ;address for IF
0216- 09 08             .SI RESTOR-1       ;address for RESTORE
0218- 88 08             .SI GOSUB-1        ;address for GOSUB
021A- D2 08             .SI RETURN-1       ;address for RETURN
021C- 3B 09             .SI REM-1          ;address for REM
021E- 27 08             .SI STOP-1         ;address for STOP
0220- 4B 09             .SI ONGOTO-1       ;address for ON
0222- 6F 37             .SI EDIT-1         ;address for EDIT
0224- 9B 16             .SI TRAP-1         ;address for TRAP
0226- 3B 22             .SI EXIT-1         ;address for EXIT
0228- 52 22             .SI DISK-1         ;address for DISK
022A- 34 12             .SI DEF-1          ;address for DEF
022C- 92 16             .SI POKE-1         ;address for POKE
022E- 34 0A             .SI PRINT-1        ;address for PRINT
0230- 52 08             .SI CONT-1         ;address for CONT
0232- B8 06             .SI LIST-1         ;address for LIST
0234- 7A 06             .SI CLEAR-1        ;address for CLEAR
0236- 61 06             .SI SCRATH-1       ;address for NEW
```

4

```
                        ;*********************************
                        ;*                               *
                        ;* Table of addresses of arith-  *
                        ;* metic function routines.      *
                        ;*                               *
                        ;*********************************
0238- 34 1B     FUNDSP  .SI SGN            ;address for SGN
023A- C7 1B             .SI INT            ;address for INT
023C- 53 1B             .SI ABS            ;address for ABS
023E- F7 22             .SI A22F7          ;address for USR
0240- 04 12             .SI FRE            ;address for FRE
0242- 25 12             .SI POS            ;address for POS
0244- 45 1E             .SI SQR            ;address for SQR
0246- 66 1F             .SI RND            ;address for RND
0248- B3 18             .SI LOG            ;address for LOG
024A- C1 1E             .SI EXP            ;address for EXP
024C- A2 1F             .SI COS            ;address for COS
024E- A9 1F             .SI SIN            ;address for SIN
0250- F2 1F             .SI TAN            ;address for TAN
0252- 56 20             .SI A2056          ;address for ATN
0254- 88 16             .SI PEEK           ;address for PEEK
0256- F6 15             .SI LEN            ;address for LEN
0258- E9 12             .SI STRD           ;address for STR$
025A- 27 16             .SI VAL            ;address for VAL
025C- 05 16             .SI ASC            ;address for ASC
025E- 66 15             .SI CHRD           ;address for CHR$
0260- 7A 15             .SI LEFTD          ;address for LEFT$
0262- A6 15             .SI RIGHTD         ;address for RIGHT$
0264- B1 15             .SI MIDD           ;address for MID$
```

5

```
                        ;*********************************
                        ;*                               *
                        ;* Table of addresses and prio-  *
                        ;* rities of operators.          *
                        ;* All are -1 because they are   *
                        ;* reached through a RTS ins-    *
                        ;* truction.                     *
                        ;*                               *
                        ;*********************************
0266- 79        OPTAB   .BY $79            ;priority for +
0267- D8 16             .SI FADDT-1        ;address for +
0269- 79                .BY $79            ;priority for -
026A- C1 16             .SI FSUBT-1        ;address for -
026C- 7B                .BY $7B            ;priority for *
026D- F3 18             .SI FMULTT-1       ;address for *
026F- 7B                .BY $7B            ;priority for /
0270- 0C 1A             .SI FDIVT-1        ;address for /
0272- 7F                .BY $7F            ;priority for ^
0273- 4E 1E             .SI FPWRT-1        ;address for ^
0275- 50                .BY $50            ;priority for AND
0276- 8B 0E             .SI ANDOP-1        ;address for AND
0278- 46                .BY $46            ;priority for OR
0279- 88 0E             .SI OROP-1         ;address for OR
027B- 7D        NEGTAB  .BY $7D            ;priority for unary -
027C- 87 1E             .SI NEGOP-1        ;address for unary -
027E- 5A        NOTTAB  .BY $5A            ;priority for NOT
027F- E9 0D             .SI NOTOP-1        ;address for NOT
0281- 64        PTDORL  .BY $64            ;priority for <,=,>
0282- B8 0E             .SI DOREL-1        ;address for <,=,>
```

De indeling van de tabel is zodanig gemaakt dat zowel in BASIC als in assembler de inhoud van de "page 0"-geheugenplaatsen snel en zonder omrekeningen kan worden opgevraagd.

Gereserveerde woorden ("token")

Zoals bij iedere hogere programmeertaal kent ook BASIC meerdere gereserveerde woorden, zoals bijvoorbeeld RUN, GOSUB, RETURN, enzovoorts. Deze gereserveerde woorden worden als één byte in het geheugen opgeslagen. Dat zijn twee vliegen in één klap: het BASIC-source-programma wordt korter en de interpreter kan het programma sneller afwerken. In het vakjargon wordt een gereserveerd woord "token" genoemd. Het token-byte onderscheidt zich van de overige bytes van de BASIC-programmatekst door het feit dat bit b7 logisch één is. Daardoor kan de computer op eenvoudige wijze de gereserveerde woorden van de overige programmatekst onderscheiden. Alle gereserveerde woorden met bijbehorend token-byte zijn in figuur 6 gegeven.

Sprongadressen

Zodra de BASIC-interpreter in het BASIC-bronprogramma een token tegenkomt, wordt dit meteen verwerkt. Bij ieder token hoort een sprongadres, dat in een tabel aan het begin van de BASIC-interpreter staat. Er zijn drie kategorieën sprongadressen:

1) Sprongadressen voor de hoofd-tokens (figuur 3)
2) Sprongadressen voor de rekenkundige tokens (figuur 4)
3) Sprongadressen voor de prioriteitsvolgorde van de rekenkundige bewerkingen

Van alle sprongadressen is één afgetrokken. Dit omdat niet via een JMP-, maar via een RTS-instruktie naar de sprongadressen gesprongen wordt. Voordat er gewerkt kan worden aan een token, moet dus eerst het sprongadres op de stack worden geschreven.
De hoofd-tokens sturen het verloop van een BASIC-programma. De rekenkundige tokens nemen de rekenkundige operaties voor hun rekening. Merk op dat ook de array-instrukties LEN, LEFT$, RIGHT$ en MID$ tot deze kategorie behoren.

Figuur 3. De sprongadressen voor de gereserveerde woorden die de BASIC-interpreter herkent.

Figuur 4. De sprongadressen voor de gereserveerde rekenkundige woorden die de BASIC-interpreter herkent.

Figuur 5. De sprongadressen en de prioriteiten (bewerkingsvolgorde) voor de rekenkundige en de logische operatoren (bewerkingen).

Figuur 6. Een overzicht van alle gereserveerde woorden die de BASIC-interpreter als zodanig herkent. Bij elk gereserveerd woord hoort een bepaalde kode: de token. Van de laatste letter van elk gereserveerd woord is bit 7 van deze geheugenkode logisch één.

Ook de rekenkundige bewerkingen zoals +, -, *, /, enzovoorts worden als een token opgeslagen. Deze tokens onderscheiden zich van de overige gereserveerde woorden door het feit dat behalve het sprongadres ook nog een prioriteitsbyte in de tabel is gespecificeerd. Hierdoor wordt een vermenigvuldiging met een hogere prioriteit bewerkt dan een optelling of aftrekking.

Gereserveerde BASIC-woorden en tokens

Figuur 6 laat zien hoe de gereserveerde BASIC-woorden in het geheugen zijn vastgelegd. Op de laatste letter van elk gereserveerd woord na zijn de letters in ASCII-vorm vastgelegd; voor de laatste letter van een gereserveerd woord is de ASCII-waarde met $80 verhoogd, dus bit 7 is logisch 1 gemaakt. Op deze wijze is de interpreter in staat om het eind van een gereserveerd woord te herkennen. Het token wordt tijdens de vertaling van een BASIC-regel voor elk gereserveerd woord berekend. Er wordt via een index naar een gereserveerd woord verwezen naar een tabel, die in het geheugen bij het adres $0284 begint. Het einde van de tabel is met de data $00 aangegeven.
Als de BASIC-interpreter een programma doorloopt, probeert deze eerst de tokens te vinden en afhankelijk daarvan de momentele BASIC-regel te bewerken. Wiskundige uitdrukkingen met haakjes kunnen niet simpel van links naar rechts worden afgewerkt. In zo'n geval moeten eerst allerlei tussenuitkomsten worden berekend. Daarvoor wordt de stack gebruikt. In het algemeen kan men zeggen dat de stack wordt gebruikt voor tussenuitkomsten; die blijven daar net zo lang totdat er verder mee kan worden gewerkt.
Het wordt allemaal nòg ingewikkelder als de BASIC-interpreter een FOR-NEXT-lus uitvoert. De FOR-variabele, de STEP-grootte en het regelnummer met de FOR worden op de stack genoteerd. Figuur 7a laat zien hoe een FOR-blok is opgebouwd.
Ook het terugkeeradres van een GOSUB-kommando moet op de stack worden geplaatst. Figuur 7b toont de opbouw van de stack onder invloed van de BASIC-interpreter.

De interne presentatie van variabelen

Net zoals een assembler een symbooltabel opbouwt, reserveert de BASIC-interpreter ten behoeve van de gebruikte variabelen meerdere stacks. Deze bevinden zich in het geheugen direkt achter het BASIC-source-programma. Bij BASIC is er sprake van twee hoofdgroepen van variabelen, die beide weer zijn onderverdeeld in:

1) Eenvoudige variabelen
a) Drijvende-komma-variabelen (floating point) (real)
b) Geheel-getal-variabelen (integer)
c) String-variabelen

2) Array-variabelen
a) Drijvende-komma-variabelen (real)
b) Geheel-getal-variabelen (integer)

6

```
                        ;*********************************
                        ;*                               *
                        ;* Table of keywords in ASCII    *
                        ;* Last letter of each keyword   *
                        ;* has MSB set to indicate end   *
                        ;* of keyword.                   *
                        ;*                               *
                        ;*********************************
0284- 45 4E C4  RESLST .BY 'EN' $C4      ;token $80, END
0287- 46 4F D2         .BY 'FO' $D2      ;token $81, FOR
028A- 4E 45 58         .BY 'NEX' $D4     ;token $82, NEXT
028D- D4
028E- 44 41 54         .BY 'DAT' $C1     ;token $83, DATA
0291- C1
0292- 49 4E 50         .BY 'INPU' $D4    ;token $84 ,INPUT
0295- 55 D4
0297- 44 49 CD         .BY 'DI' $CD      ;token $85, DIM
029A- 52 45 41         .BY 'REA' $C4     ;token $86, READ
029D- C4
029E- 4C 45 D4         .BY 'LE' $D4      ;token $87, LET
02A1- 47 4F 54         .BY 'GOT' $CF     ;token $88, GOTO
02A4- CF
02A5- 52 55 CE         .BY 'RU' $CE      ;token $89, RUN
02A8- 49 C6            .BY 'I' $C6       ;token $8A, IF
02AA- 52 45 53         .BY 'RESTOR' $C5  ;token $8B, RESTORE
02AD- 54 4F 52
02B0- C5
02B1- 47 4F 53         .BY 'GOSU' $C2    ;token $8C, GOSUB
02B4- 55 C2
02B6- 52 45 54         .BY 'RETUR' $CE   ;token $8D, RETURN
02B9- 55 52 CE
02BC- 52 45 CD         .BY 'RE' $CD      ;token $8E, REM
02BF- 53 54 4F         .BY 'STO' $D0     ;token $8F, STOP
02C2- D0
02C3- 4F CE            .BY 'O' $CE       ;token $90, ON
02C5- 45 44 49         .BY 'EDI' $D4     ;token $91, EDIT
02C8- D4
02C9- 54 52 41         .BY 'TRA' $D0     ;token $92, TRAP
02CC- D0
02CD- 45 58 49         .BY 'EXI' $D4     ;token $93, EXIT
02D0- D4
02D1- 44 49 53         .BY 'DIS' $CB     ;token $94, DISK
02D4- CB
02D5- 44 45 C6         .BY 'DE' $C6      ;token $95, DEF
02D8- 50 4F 4B         .BY 'POK' $C5     ;token $96, POKE
02DB- C5
02DC- 50 52 49         .BY 'PRIN' $D4    ;token $97, PRINT
02DF- 4E D4
02E1- 43 4F 4E         .BY 'CON' $D4     ;token $98, CONT
02E4- D4
02E5- 4C 49 53         .BY 'LIS' $D4     ;token $99, LIST
02E8- D4
02E9- 43 4C 45         .BY 'CLEA' $D2    ;token $9A, CLEAR
02EC- 41 D2
02EE- 4E 45 D7         .BY 'NE' $D7      ;token $9B, NEW
02F1- 54 41 42         .BY 'TAB' $A8     ;token $9C, TAB(
02F4- A8
02F5- 54 CF            .BY 'T' $CF       ;token $9D, TO
02F7- 46 CE            .BY 'F' $CE       ;token $9E, FN
02F9- 53 50 43         .BY 'SPC' $A8     ;token $9F, SPC(
02FC- A8
02FD- 54 48 45         .BY 'THE' $CE     ;token $A0, THEN
0300- CE
0301- 4E 4F D4         .BY 'NO' $D4      ;token $A1, NOT
0304- 53 54 45         .BY 'STE' $D0     ;token $A2, STEP
0307- D0
0308- AB               .BY $AB           ;token $A3, +
0309- AD               .BY $AD           ;token $A4, -
030A- AA               .BY $AA           ;token $A5, *
030B- AF               .BY $AF           ;token $A6, /
030C- DE               .BY $DE           ;token $A7, ^
030D- 41 4E C4         .BY 'AN' $C4      ;token $A8, AND
0310- 4F D2            .BY 'O' $D2       ;token $A9, OR
0312- BE               .BY $BE           ;token $AA, >
0313- BD               .BY $BD           ;token $AB, =
0314- BC               .BY $BC           ;token $AC, <
0315- 53 47 CE         .BY 'SG' $CE      ;token $AD, SGN
0318- 49 4E D4         .BY 'IN' $D4      ;token $AE, INT
031B- 41 42 D3         .BY 'AB' $D3      ;token $AF, ABS
031E- 55 53 D2         .BY 'US' $D2      ;token $B0, USR
0321- 46 52 C5         .BY 'FR' $C5      ;token $B1, FRE
0324- 50 4F D3         .BY 'PO' $D3      ;token $B2, POS
0327- 53 51 D2         .BY 'SQ' $D2      ;token $B3, SQR
032A- 52 4E C4         .BY 'RN' $C4      ;token $B4, RND
032D- 4C 4F C7         .BY 'LO' $C7      ;token $B5, LOG
0330- 45 58 D0         .BY 'EX' $D0      ;token $B6, EXP
0333- 43 4F D3         .BY 'CO' $D3      ;token $B7, COS
0336- 53 49 CE         .BY 'SI' $CE      ;token $B8, SIN
0339- 54 41 CE         .BY 'TA' $CE      ;token $B9, TAN
033C- 10 10 90         .BY $10 $10 $90   ;token $BA, (used to be ATN)
033F- 50 45 45         .BY 'PEE' $CB     ;token $BB, PEEK
0342- CB
0343- 4C 45 CE         .BY 'LE' $CE      ;token $BC, LEN
0346- 53 54 52         .BY 'STR' $A4     ;token $BD, STR$
0349- A4
034A- 56 41 CC         .BY 'VA' $CC      ;token $BE, VAL
034D- 41 53 C3         .BY 'AS' $C3      ;token $BF, ASC
0350- 43 48 52         .BY 'CHR' $A4     ;token $C0, CHR$
0353- A4
0354- 4C 45 46         .BY 'LEFT' $A4    ;token $C1, LEFT$
0357- 54 A4
```

c) String-variabelen
d) Gedimensioneerde arrays

De opbouw van eenvoudige variabelen is in de figuren 8a...8c gegeven. Iedere variabele beslaat acht opeenvolgende geheugenplaatsen. In het geval van drijvende-komma-variabelen zijn alle geheugenplaatsen bezet, terwijl bij de geheel-getal-variabelen en de string-variabelen slechts een deel wordt gebruikt. Ongebruikte variabele-geheugenplaatsen worden door de interpreter automatisch met de data $00 gevuld.
De eerste twee bytes van een drijvende-komma-variabele (figuur 8a) zijn voor de naam bestemd. Dan volgt de exponent, met het teken in bit 7. Merk op dat hier het komplemnent van het teken van de exponent staat! Vervolgens de mantisse, die in de volgende vier bytes is vastgelegd. Het teken van de mantisse is niet "gekomplementeerd". Teneinde met drijvende-komma-getallen snel te kunnen werken, worden deze intern in positieve en negatieve getallen opgesplitst. De formules voor de beide presentatievormen luiden:
Positief drijvende-komma-getal "+Y":

Vervolg figuur 6

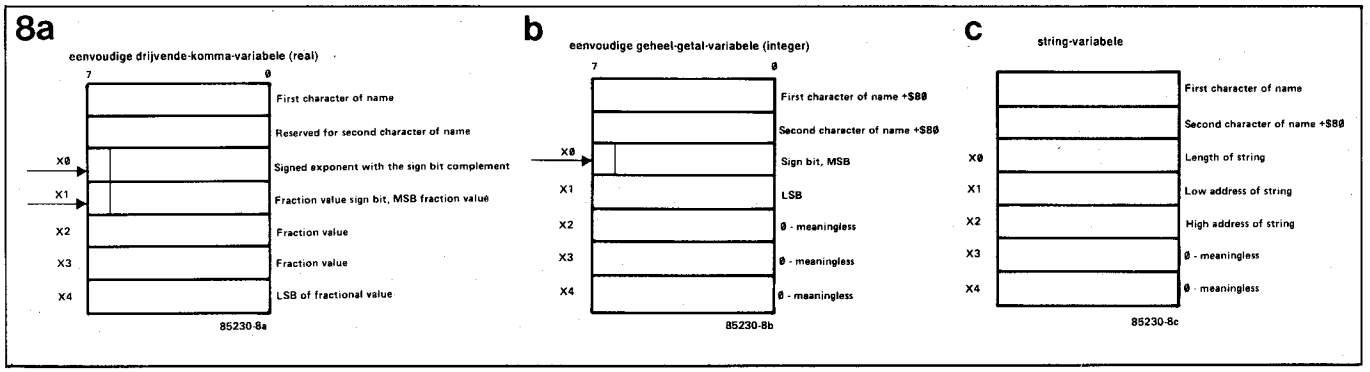
```
0359- 52 49 47          .BY 'RIGHT' $A4   ;token $C2, RIGHT$
035C- 48 54 A4
035F- 4D 49 44          .BY 'MID' $A4     ;token $C3, MID$
0362- A4
0363- 00                .BY $00           ;end of table indicator
                        .FI "OHIO BASIC.MO2"
```

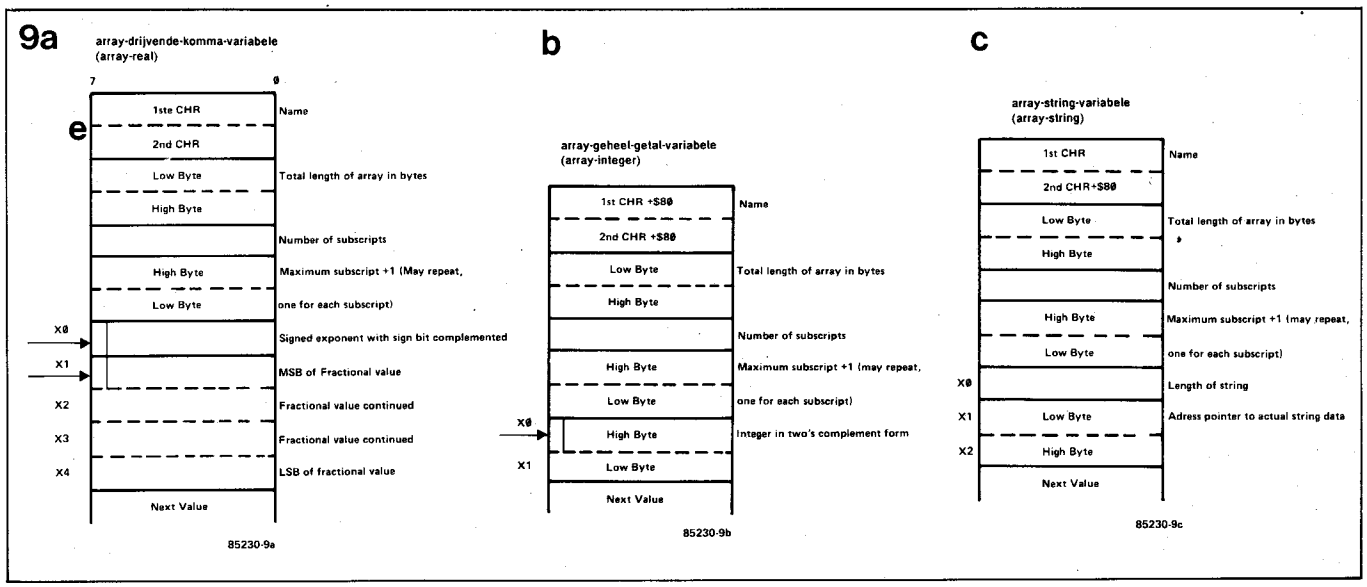
7a

```
stack layout of a FOR block:
      SP+01          81    ;FOR token
     (SP+02)               ;pointer to FOR variable
      SP+04                ;STEP size, exponent
      SP+05                ;STEP size, MSB mantissa
      SP+06                ;STEP size, mantissa
      SP+07                ;STEP size, mantissa
      SP+08                ;STEP size, LSB mantissa
      SP+09                ;sign of STEP size
      SP+0A                ;TO-value, exponent
      SP+0B                ;TO-value, MSB mantissa
      SP+0C                ;TO-value, mantissa
      SP+0D                ;TO-value, mantissa
      SP+0E                ;TO-value, LSB mantissa
     (SP+0F)               ;line number of FOR statement
     (SP+11)               ;CHRGET pointer of FOR statement
                           ;(hi and lo bytes reversed)
```

b

```
stack layout of a GOSUB block:
      SP+01          8C    ;GOSUB token
     (SP+02)               ;pointer of saved statement
     (SP+04)               ;CHRGET pointer of GOSUB statement
                           ;(hi and lo in correct order)
```





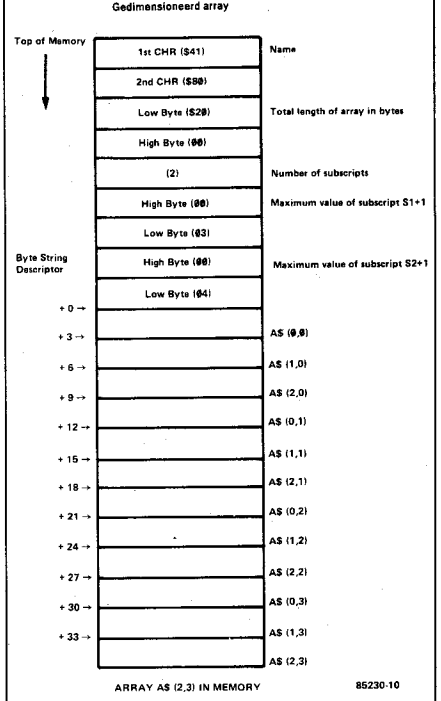
$Y=2\uparrow(x0-129)*(1+x1/128)+x2/(128*256)+x3/(128*256\uparrow2)+x4/(128*256\uparrow3)$

Negatief drijvende-komma-getal "-Y":

$Y=-(2\uparrow(x0-129))*(1+(x1-128)/128+x2/(128*256)+x3/(128*256\uparrow2)+x4/(128*256\uparrow3))$

Het onderscheid tussen de diverse variabelen is erg eenvoudig. Door het zetten van bit 7 in de variabele-naam kan de interpreter uitmaken of het om een drijvende-komma-, een geheel-getal- danwel een string-variabele gaat. Bij de drijvende-komma-variabelen zijn de beide eerste tekens van de naam ongewijzigd op de stack geplaatst (figuur 8a). Bij geheel-getal-variabelen zijn eveneens de eerste twee tekens van de naam op de stack geplaatst, echter met bit 7 voor beide tekens gezet (figuur 8b). Van string-variabelen wordt bit 7 van het tweede teken van de naam "1" gemaakt (figuur 8c). Aangezien in het geval van geheel-getal-variabelen met 16 bits rekennauwkeurigheid wordt gewerkt, kunnen slechts getallen binnen het bereik —32768...+32767 worden gerepresen-

10



11

```
10 REM ************************************************************************
20 REM * Demo voor de interne registratie van een floating-point-variabele *
30 REM ************************************************************************
40 REM
50:
60 PRINT:PRINT"################################################"
70 INPUT"Geef een floating-point-getal: ";a
80 PRINT:PRINT"Het getal staat in de variabelen-stack."
90 :
100 REM Adreswijzer op de variabelen-stack richten
110 x=PEEK(123)*256+PEEK(122)+2
120 :
130 REM Lees het array voor de dec/hex-omzetting
140 DIM h$(15): GOSUB 480
150 :
160 PRINT"De interne registratie van de variabele is: "
170 PRINT"---------------------------"
180 PRINT"decimaal       hexadecimaal"
190 PRINT"---------------------------"
200 x0=PEEK(x+0): PRINT "x0=";x0,: n=x0: GOSUB 530
210 x1=PEEK(x+1): PRINT "x1=";x1,; n=x1: GOSUB 530
220 x2=PEEK(x+2): PRINT "x2=";x2,; n=x2: GOSUB 530
230 x3=PEEK(x+3): PRINT "x3=";x3,; n=x3: GOSUB 530
240 x4=PEEK(x+4): PRINT "x4=";x4,; n=x4: GOSUB 530: PRINT
250 IF PEEK(x+1)>=128 then 320
260 GOSUB 480
270 y=2^(x0-129)*(1+x1/128+x2/(128*256)+x3/(128*256^2)+x4/(128*256^3))
280 PRINT: PRINT"Het resultaat is: "
290 PRINT"2^(x0-129)*(1+x1/128+x2/(128*256)+x3/(128*256^2)";
300 PRINT"+x4/(128*256^3)) = "
310 GOTO 380
320 GOSUB 480
330 y=-(2^(x0-129))
340 y=y*(1+(x1-128)/128+x2/(128*256)+x3/(128*256^2)+x4/(128*256^3))
350 PRINT: PRINT"Het resultaat is: "
360 PRINT"-(2^(x0-129))*(1+(x1-128)/128+x2/(128*256)+x3/(128*256^2)";
370 PRINT"+x4/(128*256^3) ="
380 PRINT y
390 RUN
400 :
410 PRINT"De interne registratie opnieuw 'decimaal' ingeven:"
420 INPUT "x0= ";x0
430 INPUT "x1= ";x1
440 INPUT "x2= ";x2
450 INPUT "x3= ";x3
460 INPUT "x4= ";x4
470 RETURN
480 :
490 FOR k=0 TO 15
500 READ h$: h$(k)=h$
510 NEXT
520 RETURN
530 xx=n: cnt=1
540 n=n-16
550 IF n<0 THEN 570
560 cnt=cnt+1: GOTO540
570 cnt=cnt-1
580 PRINT "$"; h$(cnt);
590 n=xx-(cnt*16)
600 IF n<0 THEN n=0
610 PRINT h$(n)
620 RETURN
630 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
```

Figuren 7a en 7b. In het geval van een FOR-NEXT programmalus of een GOSUB-instruktie moeten allerlei data tijdelijk worden opgeslagen. De BASIC-interpreter plaatst deze data op de stack. De opbouw van de stack is duidelijk zichtbaar.

Figuur 8a. De registratie van een eenvoudige drijvende-komma-variabele (real).

Figuur 8b. De registratie van een eenvoudige geheel-getal-variabele (integer).

Figuur 8c. De registratie van een eenvoudige string-variabele.

Figuur 9a. De registratie van een array-drijvende-komma-variabele (array real).

Figuur 9b. De registratie van een array-geheel-getal-variabele (array integer).

Figuur 9c. De registratie van een array-string-variabele (array string).

Figuur 10. Een gedimensioneerd array is nog gekompliceerder opgebouwd dan de andere soorten variabelen. Hier moet nog aanvullende informatie in een tabel worden samengevat. De rekentijd en de geheugenbehoefte zijn bij meerdimensionale arrays, gezien de omvang van de tabellen, zeer groot.

Figuur 11. Dit demonstratieprogramma laat zien hoe een eenvoudige drijvende-komma-variabele intern is opgebouwd. Het programma leest een variabele van de variabelen-stack en geeft deze variabele byte voor byte weer. Ook de formules voor de berekening van positieve en negatieve drijvende-komma-getallen worden zichtbaar gemaakt.

teerd. De interpreter registreert een string met twee "notities" op de stack: de lengte van de string en de adreswijzer (pointer) voor de string.
De array-variabelen zijn op dezelfde manier als de eenvoudige variabelen opgebouwd (figuur 9a...9c). Het onderscheid tussen de drie typen variabelen gebeurt via het "1" maken van bit 7 van een of beide tekens van de naam. Aangezien array-variabelen niet vaak meer dan 7 bytes innemen, is het duidelijk dat bij kleine computers het geheugen snel vol is. In het bijzonder bij gedimensioneerde arrays (figuur 10) noteert de interpreter heel veel op de variabelen-stack. Het zou te ver voeren om alle details van array-variabelen te beschrijven. De figuren 9 en 10 geven een eerste indruk van de wijze waarop de interpreter arrays intern registreert.

Van de theorie naar de praktijk

Figuur 11 toont een BASIC-programmaatje dat drijvende-komma-variabelen konform de interne registratie laat zien. Dit programma doet het volgende:

12a

```
##########################################
Geef een floating-point-getal: ? 0

Het getal staat in de variabelen-stack.
De interne registratie van de variabele is:
---------------------------
decimaal       hexadecimaal
---------------------------
x0= 0          $00
x1= 0          $00
x2= 0          $00
x3= 0          $00
x4= 0          $00

De interne registratie opnieuw 'decimaal' ingeven:
x0= ? 0
x1= ? 0
x2= ? 0
x3= ? 0
x4= ? 0

Het resultaat is:
2^(x0-129)*(1+x1/128+x2/(128*256)+x3/(128*256^2)+x4/(128*256^3)) =
 0
```

b

```
##########################################
Geef een floating-point-getal: ? 1

Het getal staat in de variabelen-stack.
De interne registratie van de variabele is:
---------------------------
decimaal       hexadecimaal
---------------------------
x0= 129        $81
x1= 0          $00
x2= 0          $00
x3= 0          $00
x4= 0          $00

De interne registratie opnieuw 'decimaal' ingeven:
x0= ? 129
x1= ? 0
x2= ? 0
x3= ? 0
x4= ? 0

Het resultaat is:
2^(x0-129)*(1+x1/128+x2/(128*256)+x3/(128*256^2)+x4/(128*256^3)) =
 1
```

c

```
##########################################
Geef een floating-point-getal: ? 3.14159

Het getal staat in de variabelen-stack.
De interne registratie van de variabele is:
---------------------------
decimaal       hexadecimaal
---------------------------
x0= 130        $82
x1= 73         $49
x2= 15         $0F
x3= 207        $CF
x4= 130        $82

De interne registratie opnieuw 'decimaal' ingeven:
x0= ? 130
x1= ? 73
x2= ? 15
x3= ? 207
x4= ? 130

Het resultaat is:
2^(x0-129)*(1+x1/128+x2/(128*256)+x3/(128*256^2)+x4/(128*256^3)) =
 3.14159
```

d

```
##########################################
Geef een floating-point-getal: ? 1e5

Het getal staat in de variabelen-stack.
De interne registratie van de variabele is:
---------------------------
decimaal       hexadecimaal
---------------------------
x0= 145        $91
x1= 67         $43
x2= 80         $50
x3= 0          $00
x4= 0          $00

De interne registratie opnieuw 'decimaal' ingeven:
x0= ? 145
x1= ? 67
x2= ? 80
x3= ? 0
x4= ? 0

Het resultaat is:
2^(x0-129)*(1+x1/128+x2/(128*256)+x3/(128*256^2)+x4/(128*256^3)) =
 100000
```

Figuur 12a...12d. Deze voorbeelden laten zien hoe het demonstratieprogramma van figuur 11 werkt.

* Lees een via het toetsenbord opgegeven drijvende-komma-variabele in
* Wijs naar de variabelen-stack
* Lees de drijvende-komma-variabele byte voor byte van de stack
* Geef elk byte decimaal en hexadecimaal weer
* Lees de interne registratie opnieuw in via het toetsenbord
* Geef het resultaat weer, inklusief de formule voor de berekening

Het programma is in figuur 11 uitgebreid gedokumenteerd. Daarom wordt het hier niet opnieuw beschreven. De figuren 12a...12d laten zien hoe dit demonstratieprogramma werkt.

BASIC intern

Ter afsluiting laten we zien hoe een BASIC-programma in het geheugen is "genoteerd" (figuur 13). In de regels 10 en 20 worden twee string-variabelen (A$ en B$) gedefinieerd. In de regels 30 en 40 definiëren we twee drijvende-komma-variabelen. In regel 50 staat een geheel-getal-variabele. In de overige programmaregels worden zaken afgedrukt en berekeningen gemaakt. Het programma wordt gestart en de rekenresultaten worden afgedrukt.
Met het kommando <DISK!"RE M"> (reenter monitor) verlaten we BASIC en springen naar het monitorprogramma van de Octopus 65. Een hexdump (geheugenweergave) laat zien hoe het BASIC-programma intern in de vorm van konkrete bytes is opgeslagen. Aan de hand van de figuren 1 en 6 kunt u de struktuur van het source-programma gemakkelijk doorgronden. Op de adressen $3A79...$3A7C staan het begin- en eindadres van het programma. Op het adres $3A7D staat vermeld hoeveel tracks er voor het BASIC-programma op de schijf benodigd zijn. Vanaf het adres $3A7E begint het BASIC-programma met de regel-startkode $00. De beide volgende geheugenplaatsen bevatten het adres plus één van de volgende BASIC-regel in het geheu-

Figuur 13. Zo is een BASIC-programma in het geheugen van de computer opgeslagen. De tekst van het source-programma is sterk ingekort: gereserveerde woorden zijn door kodes (token) vervangen. De tabel voor de variabelen begint direkt na het BASIC-programma vanaf het adres $3B5B.

13

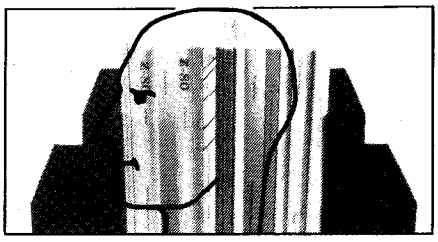
```
: m
HEXDUMP: 3a79,3b99
     9  A  B  C  D  E  F  0  1  2  3  4  5  6  7  8
3A79: 7F 3A 5B 3B 01 00 99 3A 0A 00 41 24 AB 22 45 4C  :ä;...:..A$."EL
3A89: 45 4B 54 4F 52 22 00 A3 3A 14 00 42 24 AB 22 43 EKTOR"..:..B$."C
3A99: 4F 4D 50 55 54 49 4E 47 22 00 B2 3A 1E 00 50 49 OMPUTING"..:..PI
3AA9: AB 33 2E 31 34 31 35 39 00 BF 3A 28 00 52 41 44 .3.14159..:(.RAD
3AB9: 49 55 53 AB 34 00 CA 3A 32 00 46 25 AB 31 30 30 IUS.4..:2.F%.100
3AC9: 00 DC 3A 3C 00 41 AB 50 49 A5 52 41 44 49 55 53 ..:<.A.PI.RADIUS
3AD9: A7 32 00 E4 3A 46 00 97 20 41 00 ED 3A 50 00 97 .2..:F.. A..:P..
3AE9: 20 41 24 00 F6 3A 5A 00 97 20 42 24 00 00 3B 64  A$..:Z.. B$..;d
3AF9: 00 97 20 41 24 A3 22 2D 2D 2D 22 A3 42 24 00 16 .. A$."---".B$..
3B09: 3B 6E 00 81 20 58 AB 31 20 9D 20 35 00 23 3B 78 ;n.. X.1 . 5.#;x
3B19: 00 97 20 22 48 41 4C 4C 4F 00 29 3B 82 00 82 00 .. "HALLO.);....
3B29: 2F 3B 8C 00 80 00 00 00 00 00 00 00 00 00 00 00 /;..............
3B39: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
3B49: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
3B59: 00 00 41 00 07 87 3A 00 00 42 00 09 98 3A 00 00 ..A...:..B...:..
3B69: 50 49 82 49 0F CF 82 52 41 83 00 00 00 00 C6 00 PI.I...RA.......
3B79: 00 64 00 00 00 41 00 86 49 0F CF 82 58 00 83 40 .d...A..I...X..@
3B89: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
3B99: 00
```

```
10 A$="ELEKTOR"                    OK
20 B$="COMPUTING"                  RUN
30 PI=3.14159                       50.26544
40 RADIUS=4                        ELEKTOR
50 F%=100                          COMPUTING
60 A=PI*RADIUS^2                   ELEKTOR---COMPUTING
70 PRINT A                         HALLO
80 PRINT A$                        HALLO
90 PRINT B$                        HALLO
100 PRINT A$+"---"+B$              HALLO
110 FOR X=1 TO 5                   HALLO
120 PRINT "HALLO
130 NEXT                           OK
140 END                            DISK!"re m
```

```
: m
HEXDUMP: d7e0,d800
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
D7E0: 00 00 45 4C 45 4B 54 4F 52 2D 2D 2D 43 4F 4D 50 ..ELEKTOR---COMP
D7F0: 55 54 49 4E 47 00 00 00 00 00 00 00 00 00 00 00 UTING...........
D800: 00
```

gen, die bij $3A8F begint. Het regelnummer is in de beide volgende geheugenplaatsen ($3A71, $3A72) hexadecimaal vastgelegd: $000A = decimaal 10. Dan volgt de "ingedikte" tekst van de BASIC-regel. Alle gereserveerde BASIC-woorden kunt u gemakkelijk herkennen aan het gezette bit 7. Het is-gelijk-teken = en de gereserveerde BASIC-woorden zijn door tokens vervangen, welke u in figuur 6 kunt terugvinden. Met drie nullen vanaf adres $3B2E wordt het BASIC-programma afgesloten.

De variabelen-stack begint op het adres $3B5B. Daar vindt u de variabelen A$, B$, PI, RA(DIUS) en F%. Ook variabelen die tijdens het programma worden berekend, legt de interpreter op de stack (bijvoorbeeld de variabele A). In regel 100 van het BASIC-programma hangen we drie strings aan elkaar. De interpreter moet daartoe eerst de string maken, voordat deze kan worden weergegeven. Aan het eind van het werkgeheugen ($D7E2...) treffen we de string aan zoals die in beeld of op het printer-papier verschijnt.



de opvolgers

De 65-serie weer up to date

Eigenlijk hadden we gepland om in dit tweede deel van onze artikelserie heel mooi systematisch door te gaan met de beschrijving van de 65SC802 en de 65SC816. Maar daar zijn twee dingen tussen gekomen: ten eerste plaatsgebrek in deze aflevering van Elektuur Computing, en ten tweede: verschillende lezers hebben ons verzocht om hen kopiën van datasheets van de nieuwe processors toe te sturen, omdat ze zich nu al met deze nieuwe wonderen der techniek willen gaan bezighouden. Er was zelfs een lezer, die het heeft klaargespeeld om een monster van de 65SC816 te "versieren"; via een vriend in de USA georganiseerd. En we hebben vervolgens zó geredeneerd: Oké, onze lezers bestaan niet alleen uit een groot aantal computerhobbyisten, maar óók uit softwareprofessionals. En hoe eerder we die laatste kategorie van lezers alle noodzakelijke informatie kunnen verschaffen over deze nieuwe microprocessoren, des te sneller is er nieuwe software beschikbaar voor de EC-65K.

Welke informatie heeft een ervaren computerist nodig, afgezien van de informatie in deel 1 van deze artikelserie, om zich direkt al met de software voor de nieuwe processoren bezig te houden? Het antwoord op deze vraag treft u aan op de nu volgende pagina's.

We koncentreren ons in dit deel 2 op kompakte, hapklare technische informatie en stellen alle toelichtingen op deze informatie uit tot de derde aflevering van deze artikelserie, in Elektuur Computing 4.

Terzake

* Tabel 1 bevat de mnemonics voor de nieuwe instrukties (en voor oude instrukties met nieuwe adresseringsvormen). De oude 6502-mnemonics en de oude adresseringsvormen zijn bekend verondersteld en in deze tabel weggelaten (zie deel A van de WDC-datasheet).
* Tabel 2 toont de adresseringsmogelijkheden.
* Tabel 3 bevat de hexadecimale opcodematrix.
* Tabel 4 geeft een toelichting op de afzonderlijke instrukties, met de bijbehorende opcodes. Tevens ziet men welke bits van het statusregister worden beïnvloed.

Zoals gezegd: in de volgende aflevering van Elektuur Computing volgt nadere informatie bij deze tabellen. Op dat tijdstip zou, volgens GTE Europe, óók de 4-MHz-versie van de 65SC816 in produktie-aantallen beschikbaar zijn. Dat komt dus mooi uit, gezien de voor Elektuur Computing 4 voorspelde start van de bouw van de EC-65K.

Literatuur:

GTE-datasheet
65SC802/65SC816
GTE Microcircuits

WDC-datasheet
W65SC816
Western Digital Corporation

Met dank aan de beide firma's voor hun hulp. De tabellen in dit artikel zijn afkomstig uit de genoemde datasheets.

Instruction Set

W65SC816 Instructions (256 OP Codes)

B. New W65SCXXX Instructions (13 Op Codes)

1. BRA Branch Relative always
2. PLX Pull X from Stack
3. PLY Pull Y from Stack
4. PHX Push X on Stack
5. PHY Push Y on Stack
6. STZ Store Zero in Memory (Direct; Direct, X; Abs; Abs, X)
7. TRB Test and Reset Memory Bits Determined by Accumulator A (Direct and Absolute).
8. TSB Test and Set Memory Bits Determined by Accumulator A (Direct and Absolute).

C. New W65SCXXX Addressing Modes (14 Op Codes)

2. BIT Test Bits in Memory with Accumulator (Direct, X; Absolute, X; Immediate).
2. DEC Decrement (Accumulator)
3. Group I Instructions (Direct Indirect (8 Op Codes))
4. INC Increment (Accumulator)
5. JMP Jump to New Location (Absolute Indexed Indirect)

D. Group I Instructions with New Addressing Modes (48 Op Codes)

- Direct Indirect Long Indexed with Y (8 Op Codes)
- Direct Indirect Long (8 Op Codes)
- Absolute Long and Absolute Long Indexed with X (16 Op Codes)
- Stack Relative (8 Op Codes)
- Stack Relative Indirect Indexed Y. (8 Op Codes)

1. ADC Add Memory to Accumulator with Carry
2. AND "AND" Memory with Accumulator
3. CMP Compare Memory and Accumulator
4. EOR "Exclusive-or" Memory with Accumulator
5. LDA Load Accumulator with Memory
6. ORA "Or" Memory with Accumulator
7. SBC Subtract Memory from Accumulator with Borrow
8. STA Store Accumulator in Memory

E. New Push and Pull Instructions (7 Op Codes)

1. PEA Push Effective Absolute Address or Immediate Data Word on Stack
2. PEI Push Effective Indirect Address or Direct Data Word on Stack
3. PER Push Effective Program Counter Relative Indirect Address or Program Counter Relative Data Word on Stack
4. PLB Pull Data Bank Register from Stack
5. PLD Pull Direct Register from Stack
6. PHB Push Data Bank Register on Stack
7. PHD Push Direct Register on Stack
8. PHK Push Program Bank Register on stack

F. Status Register Instructions (2 Op Codes)

1. REP Reset Status Bits Defined by Immediate Byte 1 = Reset
   0 = Do not change
2. SEP Set Status Bits Defined by Immediate Byte 1 = Set
   0 = Do not change

G. New Register Transfer Instructions (8 Op Codes)

1. TCD Transfer C Accumulator to Direct Register D
2. TDC Transfer Direct Register D to C Accumulator
3. TCS Transfer C Accumulator to Stack Register
4. TSC Transfer Stack Register to Accumulator C
5. TXY Transfer X to Y
6. TYX Transfer Y to X
7. XBA Exchange B and A
8. SCE Exchange Carry Bit C with Emulation Bit E.

H. New Branch, Jump and Return Instructions (6 Op Codes)

1. BRL Branch Relative Long Always (16 Bit Relative—32768 to + 32767) (Addressing Mode)
2. JML Jump Indirect Long
3. JMP Jump Absolute Long
4. JSL Jump to Subroutine Long (Uses RTL for Return)
5. JSR Jump to Subroutine (Indexed Indirect)
6. RTL Return from Subroutine Long

I. New Block Move Instructions (2 Op Codes)

1. MVN Move Block from Source (X Addressed) to Destination (Y Addressed), Block Length Defined by C, X, Y are Incremented.
2. MVP Move Block from Source (X Addressed) to Destination (Y Addressed), Block Length Defined by C, X, Y are Decremented.

J. New Co-Processor Operations (1 Op Code)

1. COP Co-Processor Instruction with Associated COP Vector and ABORT Input Supports Co-Processing Function i.e., Floating Point Processors, etc.

K. New System Control Instructions (3 Op Codes)

1. STP Stop-the-clock Instruction Stops the Oscillator Input (or 02 Input) During 02 = 1. This Mode Is Released When RES Goes to a Zero. System Initialization May Be Desired; However, if After RESET One Performed an RTI, Program Execution Begins With the Instruction Following the STP Op Code in Program Sequence.
2. WAI Wait for Interrupt Pulls RDY Low and Is Cleared by IRQ or NMI Active Input.
3. WDM There is One Reserved Op Code Defined as WDM Which Will Be Used For Future Systems. The W65SC816 Performs a No-Operation.

symbol	addressing mode	symbol	addressing mode
#	immediate	[d]	direct indirect long
A	accumulator	[d],y	direct indirect indexed long
r	program counter relative	a	absolute
rl	program counter relative long	a,x	absolute indexed (with x)
i	implied	a,y	absolute indexed (with y)
s	stack	al	absolute long
d	direct	al,x	absolute indexed long
d,x	direct indexed (with x)	r,s	stack relative
d,y	direct indexed (with y)	(r,s),y	stack relative indirect indexed
(d)	direct indirect	(a)	absolute indirect
(d,x)	direct indexed indirect	(a,x)	absolute indexed indirect
(d),y	direct indirect indexed	xya	block move

Verklaringen bij tabel 3

legend	
instruction mnemonic	addressing mode
base number of bytes	base number of cycles

2

Addressing Mode Summary

Address Mode	Instruction Times In Memory Cycles		Memory Utilization In Number of Program Sequence Bytes	
	Original 8 Bit NMOS 6502	New W65SC816	Original 8 Bit NMOS 6502	New W65SC816
1. Immediate	2	2(3)	2	2(3)
2. Absolute	4(5)	4(3,5)	3	3
3. Absolute Long	—	5(3)	—	4
4. Direct	3(5)	3(3,4,5)	2	2
5. Accumulator	2	2	1	1
6. Implied	2	2	1	1
7. Direct Indirect Indexed (IND), Y	5(1)	5(1,3,4)	2	2
8. Direct Indirect Indexed Long (IND), Y Long	—	6(3,4)	—	2
9. Direct Indexed Indirect (IND, X)	6	6(3,4)	2	2
10. Direct, X	4(5)	4(3,4,5)	2	2
11. Direct, Y	4	4(3,4)	2	2
12. Absolute, X	4(1,5)	4(1,3,5)	3	3
13. Absolute Long, X	—	5(3)	—	4
14. Absolute, Y	4(1)	4(1,3)	3	3
15. Relative	2(1,2)	2(2)	2	2
16. Relative Long	—	3(2)	—	3
17. Absolute Indirect (Jump)	5	5	3	3
18. Direct Indirect	—	5(3,4)	—	2
19. Direct Indirect Long	—	6(3,4)	—	2
20. Absolute Indexed Indirect (Jump)	—	6	—	3
21. Stack	3-7	3-8	1-3	1-4
22. Stack Relative	—	4(3)	—	2
23. Stack Relative Indirect Indexed	—	7(3)	—	2
24. Block Move X, Y, C (Source, Destination, Block Length)	—	7	—	3

NOTES:
1. Page boundary, add 1 cycle if page boundary is crossed when forming address.
2. Branch taken, add 1 cycle if branch is taken.
3. M = 0 or X = 0, 16 bit operation, add 1 cycle, add 1 byte for immediate.
4. Direct register low (DL) not equal zero, add 1 cycle.
5. Read-Modify-Write, add 2 cycles for M = 1, add 3 cycles for M = 0.

3

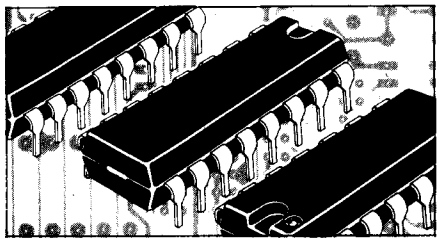
Opcode Matrix

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	MSD
0	BRK s 2 8	ORA (d,x) 2 6	COP s 2 8	ORA r,s 2 4	TSB d 2 5	ORA d 2 3	ASL d 2 5	ORA [d] 2 6	PHP s 1 3	ORA # 2 2	ASL A 1 2	PHD s 1 4	TSB a 3 6	ORA a 3 4	ASL a 3 6	ORA al 4 5	0
1	BPL r 2 2	ORA (d),y 2 5	ORA (d) 2 5	ORA (r,s),y 2 7	TRB d 2 5	ORA d,x 2 4	ASL d,x 2 6	ORA [d],y 2 6	CLC i 1 2	ORA a,y 3 4	INC A 1 2	TCS i 1 2	TRB a 3 6	ORA a,x 3 4	ASL a,x 3 7	ORA al,x 4 5	1
2	JSR a 3 6	AND (d,x) 2 6	JSL al 4 8	AND r,s 2 4	BIT d 2 3	AND d 2 3	ROL d 2 5	AND [d] 2 6	PLP s 1 4	AND # 2 2	ROL A 1 2	PLD s 1 5	BIT a 3 4	AND a 3 4	ROL a 3 6	AND al 4 5	2
3	BMI r 2 2	AND (d),y 2 5	AND (d) 2 5	AND (r,s),y 2 7	BIT d,x 2 4	AND d,x 2 4	ROL d,x 2 6	AND [d],y 2 6	SEC i 1 2	AND a,y 3 4	DEC A 1 2	TSC i 1 2	BIT a,x 3 4	AND a,x 3 4	ROL a,x 3 7	AND al,x 4 5	3
4	RTI s 1 7	EOR (d,x) 2 6	reserve 2 2	EOR r,s 2 4	MVP xya 3 7	EOR d 2 3	LSR d 2 5	EOR [d] 2 6	PHA s 1 3	EOR # 2 2	LSR A 1 2	PHK s 1 3	JMP a 3 3	EOR a 3 4	LSR a 3 6	EOR al 4 5	4
5	BVC r 2 2	EOR (d),y 2 5	EOR (d) 2 5	EOR (r,s),y 2 7	MVN xya 3 7	EOR d,x 2 4	LSR d,x 2 6	EOR [d],y 2 6	CLI i 1 2	EOR a,y 3 4	PHY s 1 3	TCD i 1 2	JMP al 4 4	EOR a,x 3 4	LSR a,x 3 7	EOR al,x 4 5	5
6	RTS s 1 6	ADC (d,x) 2 6	PER s 3 6	ADC r,s 2 4	STZ d 2 3	ADC d 2 3	ROR d 2 5	ADC [d] 2 6	PLA s 1 4	ADC # 2 2	ROR A 1 2	RTL s 1 6	JMP (a) 3 5	ADC a 3 4	ROR a 3 6	ADC al 4 5	6
7	BVS r 2 2	ADC (d),y 2 5	ADC (d) 2 5	ADC (r,s),y 2 7	STZ d,x 2 4	ADC d,x 2 4	ROR d,x 2 6	ADC [d],y 2 6	SEI i 1 2	ADC a,y 3 4	PLY s 1 4	TDC i 1 2	JMP (a,x) 3 6	ADC a,x 3 4	ROR a,x 3 7	ADC al,x 4 5	7
8	BRA r 2 2	STA (d,x) 2 6	BRL rl 3 3	STA r,s 2 4	STY d 2 3	STA d 2 3	STX d 2 3	STA [d] 2 6	DEY i 1 2	BIT # 2 2	TXA i 1 2	PHB s 1 3	STY a 3 4	STA a 3 4	STX a 3 4	STA al 4 5	8
9	BCC r 2 2	STA (d),y 2 6	STA (d) 2 5	STA (r,s),y 2 7	STY d,x 2 4	STA d,x 2 4	STX d,y 2 4	STA [d],y 2 6	TYA i 1 2	STA a,y 3 5	TXS i 1 2	TXY i 1 2	STZ a 3 4	STA a,x 3 5	STZ a,x 3 5	STA al,x 4 5	9
A	LDY # 2 2	LDA (d,x) 2 6	LDX # 2 2	LDA r,s 2 4	LDY d 2 3	LDA d 2 3	LDX d 2 3	LDA [d] 2 6	TAY i 1 2	LDA # 2 2	TAX i 1 2	PLB s 1 4	LDY a 3 4	LDA a 3 4	LDX a 3 4	LDA al 4 5	A
B	BCS r 2 2	LDA (d),y 2 5	LDA (d) 2 5	LDA (r,s),y 2 7	LDY d,x 2 4	LDA d,x 2 4	LDX d,y 2 4	LDA [d],y 2 6	CLV i 1 2	LDA a,y 3 4	TSX i 1 2	TYX i 1 2	LDY a,x 3 4	LDA a,x 3 4	LDX a,y 3 4	LDA al,x 4 5	B
C	CPY # 2 2	CMP (d,x) 2 6	REP # 2 3	CMP r,s 2 4	CPY d 2 3	CMP d 2 3	DEC d 2 5	CMP [d] 2 6	INY i 1 2	CMP # 2 2	DEX i 1 2	WAI i 1 3	CPY a 3 4	CMP a 3 4	DEC a 3 6	CMP al 4 5	C
D	BNE r 2 2	CMP (d),y 2 5	CMP (d) 2 5	CMP (r,s),y 2 7	PEI s 2 6	CMP d,x 2 4	DEC d,x 2 6	CMP [d],y 2 6	CLD i 1 2	CMP a,y 3 4	PHX s 1 3	STP i 1 3	JML (a) 3 6	CMP a,x 3 4	DEC a,x 3 7	CMP al,x 4 5	D
E	CPX # 2 2	SBC (d,x) 2 6	SEP # 2 3	SBC r,s 2 4	CPX d 2 3	SBC d 2 3	INC d 2 5	SBC [d] 2 6	INX i 1 2	SBC # 2 2	NOP i 1 2	XBA i 1 3	CPX a 3 4	SBC a 3 4	INC a 3 6	SBC al 4 5	E
F	BEQ r 2 2	SBC (d),y 2 5	SBC (d) 2 5	SBC (r,s),y 2 7	PEA s 3 5	SBC d,x 2 4	INC d,x 2 6	SBC [d],y 2 6	SED i 1 2	SBC a,y 3 4	PLX s 1 4	XCE i 1 2	JSR (a,x) 3 6	SBC a,x 3 4	INC a,x 3 7	SBC al,x 4 5	F
MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	MSD

Operation, Operation Codes, and Status Register

OPERATION	#	a	al	d	A	i	(d),y	[d],y	(d,x)	d,x	d,y	a,x	al,x	a,y	r	rl	(a)	(d)	[d]	(a,x)	s	d,s	(d,s),y	xyc	PROCESSOR STATUS CODE 7 6 5 4 3 2 1 0	MNE-MONIC
																									N V M X D I Z C	E = 0
OPERATION	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	N V 1 B D I Z C	E = 1
A + M + C → A	69	6D	6F	65			71	77	61	75		7D	7F	79				72	67			63	73		N V . . . . Z C	ADC
AVM → A	29	2D	2F	25			31	37	21	35		3D	3F	39				32	27			23	33		N . . . . . Z .	AND
C ← 15/7 0 ← 0		0E		06	0A					16		1E													N . . . . . Z C	ASL
BRANCH IF C = 0															90										. . . . .	BCC
BRANCH IF C = 1															B0										. . . . .	BCS
BRANCH IF Z = 1															F0										. . . . .	BEQ
AVM (NOTE 1)	89	2C		24						34		3C													M7 M6 . . . . Z .	BIT
BRANCH IF N = 1															30										. . . . .	BMI
BRANCH IF Z = 0															D0										. . . . .	BNE
BRANCH IF N = 0															10										. . . . .	BPL
BRANCH ALWAYS															80										. . . . .	• BRA
BREAK (NOTE 2)																					00				. . . • 0 I . .	BRK
BRANCH LONG ALWAYS																82									. . . . .	* BRL
BRANCH IF V = 0															50										. . . . .	BVC
BRANCH IF V = 1															70										. . . . .	BVS
0 → C						18																			 0	CLC
0 → D						D8																			. . . . 0 . .	CLD
0 → I						58																			. . . . . 0 .	CLI
0 → V						B8																			. . . . .	CLV
A-M	C9	CD	CF	C5			D1	D7	C1	D5		DD	DF	D9				D2	C7			C3	D3		N . . . . . Z C	CMP
CO-PROCESSOR																					02				. . . . .	* COP
X-M	E0	EC		E4																					N . . . . . Z C	CPX
Y-M	C0	CC		C4																					N . . . . . Z C	CPY
DECREMENT		CE		C6	3A					D6		DE													N . . . . . Z .	DEC
X - 1 → X						CA																			N . . . . . Z .	DEX
Y - 1 → Y						88																			N . . . . . Z .	DEY
AVM → A	49	4D	4F	45			51	57	41	55		5D	5F	59				52	47			43	53		N . . . . . Z .	EOR
INCREMENTS		EE		E6	1A					F6		FE													N . . . . . Z .	INC
X + 1 → X						E8																			N . . . . . Z .	INX
Y + 1 → Y						C8																			N . . . . . Z .	INY
JUMP LONG TO NEW LOC.																	DC								. . . . .	* JML
JUMP TO NEW LOC.		4C	5C														6C			7C					. . . . .	JMP
JUMP LONG TO SUB.			22																						. . . . .	JSL
JUMP TO SUB.		20																		FC					. . . . .	JSR
M → A	A9	AD	AF	A5			B1	B7	A1	B5		BD	BF	B9				B2	A7		A3	B3			N . . . . . Z .	LDA
M → X	A2	AE		A6							B6			BE											N . . . . . Z .	LDX
M → Y	A0	AC		A4						B4		BC													N . . . . . Z .	LDY
0 → 15/7 0 → C		4E		46	4A					56		5E													0 . . . . . Z C	LSR
M → M BACKWARD																								54	. . . . .	* MVN
M → M FORWARD																								44	. . . . .	* MVP
NO OPERATION						EA																			. . . . .	NOP
AVM → A	09	0D	0F	05			11	17	01	15		1D	1F	19				12	07			03	13		N . . . . . Z .	ORA
Mpc + 1, Mpc + 2 → Ms, Ms - 1																					F4				. . . . .	* PEA
S - 2 → S																										
M(d), M(d + 1) → Ms, Ms - 1																					D4				. . . . .	* PEI
S - 2 → S																										
Mpc + rl, Mpc + rl + 2 → Ms, MS -1																					62				. . . . .	* PER
S - 2 → S																										
A → Ms, S - 2 → S S - 1 → S																					48				. . . . .	PHA
DBR → Ms, S - 1 → S																					8B				. . . . .	PHB
D → Ms, Ms - 1, S - 2 → S																					0B				. . . . .	* PHD
PBR → Ms, S - 1 → S																					4B				. . . . .	* PHK
P → Ms, S - 1 → S																					08				. . . . .	PHP
X → Ms, S - 1 → S																					DA				. . . . .	• PHX
Y → Ms, S - 1 → S																					5A				. . . . .	• PHY
S + 1 → S, Ms → A																					68				N . . . . . Z .	PLA
S + 1 → S, Ms → DBR																					AB				N . . . . . Z .	* PLB
S + 2 → S, Ms - 1, Ms → D																					2B				N . . . . . Z .	* PLD
S + 1 → S, Ms → P																					28				N V M X D I Z C	PLP
S + 1 → S, Ms → X																					FA				N . . . . . Z .	PLX
S + 1 → S, Ms → Y																					7A				N . . . . . Z .	PLY
MVP → P	C2																								N V M X D I Z C	* REP
15/7 0 ← C		2E		26	2A					36		3E													N . . . . . Z C	ROL
C → 15/7 0		6E		66	6A					76		7E													N . . . . . Z C	ROR
RTRN FROM INT.																					40				N V M X D I Z C	RTI
RTRN FROM SUB. LONG																					6B				. . . . .	* RTL
RTRN SUBROUTINE																					60				. . . . .	RTS
A - M - C → A	E9	ED	EF	E5			F1	F7	E1	F5		FD	FF	F9				F2	E7			E3	F3		N V . . . . Z C	SBC
1 → C						38																			 1	SEC
1 → D						F8																			. . . . 1 . .	SED
1 → I						78																			. . . . 1 . .	SEI
MVP → P	E2																								N V M X D I Z C	* SEP
A → M		8D	8F	85			91	97	81	95		9D	9F	99				92	87			83	93		. . . . .	STA
STOP (1 → φ2)						DB																			. . . . .	• STP
X → M		8E		86							96														. . . . .	STX
Y → M		8C		84						94															. . . . .	STY
00 → M		9C		64						74		9E													. . . . .	• STZ
A → X						AA																			N . . . . . Z .	TAX
A → Y						A8																			N . . . . . Z .	TAY
C → D						5B																			N . . . . . Z .	* TCD
C → S						1B																			. . . . .	* TCS
D → C		1C		14		7B																			N . . . . . Z .	* TDC
																									. . . . . Z .	• TRB
AVM → M		0C		04																					. . . . . Z .	• TSB
S → C					3B																				N . . . . . Z .	* TSC
S → X					BA																				N . . . . . Z .	TSX
X → A					8A																				N . . . . . Z .	TXA
X → S					9A																				. . . . .	TXS
X → Y					9B																				N . . . . . Z .	* TXY
Y → A					98																				N . . . . . Z .	TYA
Y → X					BB																				N . . . . . Z .	* TYX
0 → RDY					CB																				. . . . .	• WAI
NO OPERATION (RESERVED)					42																				. . . . .	* WDM
B ⟶ A					EB																				N . . . . . Z .	* XBA
C ⟶ E					FB																				. . . . . E	* XCE

Notes:
1. Bit immediate N and V flags not affected. When M = 0, M_{15} → N and M_{14} → V.
2. Break Bit (B) in Status register indicates hardware or software break.
3. * = New W65C816/802 Instructions
   • = New W65C02 Instructions
   Blank = NMOS 6502



IBM-PC: kleuren-videokaart

Naast het keyboard vormt de video-interface een van de belangrijkste schakels tussen mens en machine. Waarom is de video-interface niet op het moederbord aangebracht bij de IBM-PC en zijn compatibles? Big Blue zal geweten hebben dat een video-interface als vast onderdeel van het moederbord een beperking van de mogelijkheden inhoudt. Een aparte videoprint levert meer keuzemogelijkheden. Indien gewenst kunnen zelfs meerdere videokaarten tegelijkertijd in een en dezelfde PC werkzaam zijn. De kleuren-video-adapter levert vrijwel alle mogelijkheden die de doorsnee-gebruiker zich wensen kan. Deze kaart wordt in dit artikel uitvoerig besproken.

Wat zijn zoal de keuzemogelijkheden: Allereerst is er de monochrome kaart zonder grafische mogelijkheden. In de praktijk blijkt dat de kaart over redelijke grafische kwaliteiten beschikt, omdat de karakterset een groot aantal grafische karakters heeft (63). Voor tekstverwerking en administratieve software-pakketten met beperkte grafische mogelijkheden een prima keuze.
Natuurlijk is er ook een zwart/wit-videokaart met grafische mogelijkheden en dat is dan meteen een van de beste in zijn soort. Deze (Hercules-)kaart brengt maar liefst 720 × 384 punten op het scherm; een garantie voor zeer fraaie grafische plaatjes (niet in kleur).
Beide monochrome kaarten maken gebruik van een zeer fraaie karakterset, opgebouwd uit een 7 × 9 matrix.
De kleuren-videokaart heeft zowel alfanumerieke als grafische mogelijkheden en ze heeft beschikking over dezelfde uitgebreide karakterset als de monochrome kaarten. De karakterset is echter door het gebruik van een eenvoudiger matrix iets minder fraai. Daar staat tegenover dat ieder karakter onafhankelijk in een van de 16 beschikbare kleuren (inkl. zwart en wit) kan worden weergegeven.
Grafisch biedt deze kaart in kleur een "medium" resolutie van 320 × 200 punten. In de "high resolution mode" zijn 640 × 200 in alleen zwart/wit mogelijk. De grafische prestaties komen dus dicht bij die van de Hercules-kaart.
Naast de nu beschreven eigenschappen moeten nog een paar aspekten bij het bepalen van de uiteindelijke keuze worden meegenomen. De PC is van oorsprong een Amerikaanse computer, die gebruik maakt van in Europa ongebruikelijke video-standaards.
De monochrome kaarten hanteren een volledig afwijkende standaard met een rasterfrekwentie van 50 Hz en een lijnfrekwentie van 18,432 kHz. De rasterfrekwentie is geen probleem, maar vrijwel geen enkele gangbare monitor is op zo'n hoge lijnfrekwentie in te stellen. Voor de monochrome kaart zal dus een speciale IBM-kompatibele monitor moeten worden aangeschaft. Een dergelijke monitor is dan ook voorzien van de bij de PC toegepaste 9-polige D-konnektor. Het aansluitschema van de konnektor is weergegeven in figuur 1. Let daarbij op de polariteit van de synchronisatiesignalen. Ook die zijn afwijkend: een positieve impuls voor de horizontale en een negatieve voor de vertikale synchronisatie. Een negatieve impuls voor beide signalen is gebruikelijker!

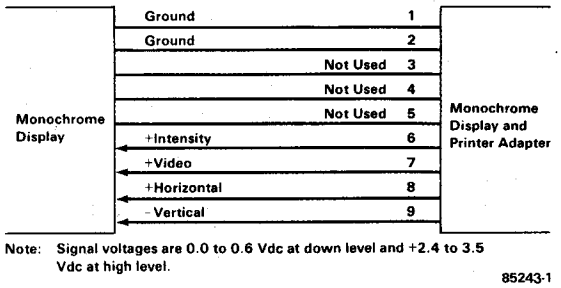
In kleur?

De kleuren-videokaart levert een NTSC-signaal op de composite-video-uitgang. Dat heeft ook zo zijn voor- en nadelen. Aansluiten van een zwart/wit-monitor op deze uitgang is alleen mogelijk zolang in z/w wordt gewerkt. Wanneer de kleurmogelijkheden worden benut, maakt de NTSC-kleurendraaggolf het beeld niet om aan te zien. Gelukkig beschikt de kaart ook over een RGB-uitgang, waarop eventueel een z/w-videocombiner kan worden aangesloten. Het genereren van een PAL-kleurensignaal is zonder een extra hulpschakeling niet mogelijk. Ook hier wordt een 9-polige D-konnektor toegepast. Nu echter zijn beide synchronisatiesignalen positief! Figuur 2 toont de aansluitgegevens.
De monitor die op de kleurenkaart wordt aangesloten, dient geschikt te zijn voor een rasterfrekwentie van 60 Hz. De lijnfrekwentie vormt geen probleem; deze ligt met 15,750 kHz dicht genoeg bij de hier in Europa gebruikelijke 15,625 kHz. Wie beschikt over een NTSC- of RGB-kleurenmonitor kan de grafische mogelijkheden bij "medium" resolutie ten volle benutten.
De in figuur 3 weergegeven videocombiner maakt van de synchronisatie- en de R-, G-, B-signalen een videosignaal (BAS) dat geschikt is voor een zwart/wit-monitor. De meeste monochrome monitoren zijn zowel voor 50- als 60-Hz-videosignalen geschikt. De weergave van grijstinten laat weliswaar soms te wensen over, maar over het algemeen zal een monochrome monitor met de schakeling uit figuur 3 een haarscherp plaatje leveren.
De schakeling bestaat uit een eenvoudige 4-bits D/A-omzetter die de kleureninformatie omzet in een zestiental grijstinten. Groen levert de grootste helderheidstoename, in sterkte gevolgd door rood en blauw. Het intensity-bit verschuift de algehele helderheid een stuk naar boven.
Daarna worden video- en synchronisatie-

Figuur 1. De aansluitgegevens voor een monitor op een monochrome videokaart. De cijfers korresponderen met de pennummering op een 9-polige D-konnektor.

Figuur 2. De aansluitgegevens voor een (kleuren-)monitor op een kleuren-videokaart. Pen 7 kan eventueel worden benut om de in dit artikel beschreven videocombiner te voeden.

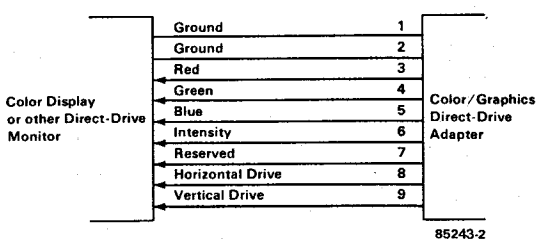
1



Note: Signal voltages are 0.0 to 0.6 Vdc at down level and +2.4 to 3.5 Vdc at high level.

85243-1

2



85243-2

signalen gemengd op de basis van T2. Aan de uitgang van de combiner is de gebruikelijke 1 Vtt over 75 Ω beschikbaar.

De kleurenkaart

De kleuren-videokaart is eigenlijk de enige videokaart die in kombinatie met een normale monochrome monitor kan worden gebruikt. Eventueel kan zelfs een omgebouwde TV worden toegepast. Deze kleurenkaart (zie foto) is derhalve een nadere bespreking waard.

Het blokschema is afgebeeld in figuur 4. De gelukkige bezitter van een IBM Technical Reference Manual zal dit blokschema waarschijnlijk direkt herkennen. Het lijkt als twee druppels water op het blokschema van de originele IBM-kleurenkaart. Het is dan ook niet zo verwonderlijk dat deze alternatieve kaart exakt dezelfde mogelijkheden biedt als het origineel.

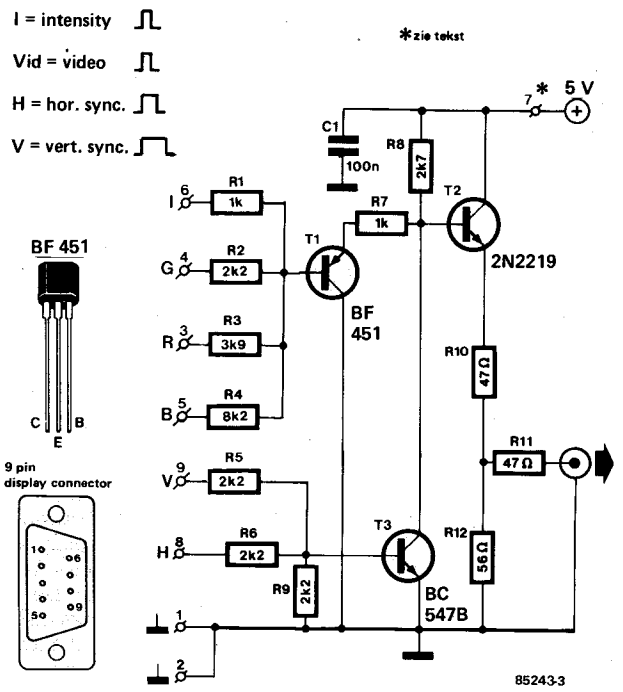
De funktie van een aantal blokken is direkt duidelijk, daarvan laten we hier een bespreking dan ook achterwege.

Wat direkt opvalt, is dat er twee maal een "data latch" in het blokschema voorkomt. De kaart gebruikt dan ook twee bytes per karakter op het scherm. Een byte bevat de gebruikelijke ASCII-kode, het tweede (attribute) byte bevat informatie over de kleur, intensiteit en het eventueel knipperen van het karakter. In tabel 1 zijn de gemeenschappelijke mogelijkheden van zowel de monochrome als de kleurenkaart(en) aangegeven. Met elke andere kode zal een monochrome kaart witte karakters op een witte achtergrond produceren; met andere woorden: de tekst blijft onzichtbaar. De kleurenkaart biedt daarboven nog de mogelijkheid om te keizen uit acht achtergrondkleuren en zestien verschillende voorgrond- of karakterkleuren. In tabel 2 zijn alle mogelijke kleuren nog eens opgesomd.

De beide data latches worden gevoed vanuit een 16-K-"Display Buffer". In de grafische modes wordt de buffer vrijwel in zijn geheel benut. Voor een z/w grafisch scherm met 640 x 200 punten zijn in totaal 128000 bits oftewel precies 16000 bytes nodig. Slechts een zeer klein gedeelte van de 16-K-buffer blijft dus onbenut. Het grafische kleurenscherm heeft een resolutie van 320 x 200 punten. Precies de helft van het z/w scherm. Dit betekent dat er voor iedere kleurenpunt slechts 2 bits beschikbaar zijn. De kleurenweergave wordt hierdoor helaas enorm beperkt. In plaats van de in tabel 2 aangegeven kleuren is slechts een palet van vier kleuren mogelijk. Hierbij speelt het palet-register (zie figuur 4) een rol als preselektor. Men heeft de keuze uit twee paletten, in het paletregister moet worden aangegeven welk palet gewenst is. In tabel 3 zijn de twee mogelijke paletten weergegeven. Een van de paletkleuren is in beide gevallen een van de (16) achtergrondkleuren.

In de alphanumerieke mode worden per karakter twee bytes gebruikt. De tekstweergave gebeurt naar keuze met 25 regels van 40 of 80 karakters. Hiervoor zijn slechts 2000 of 4000 bytes geheugenruimte nodig. De "Display Buffer" kan daardoor, indien gewenst, acht of vier schermen (pagina's) tekst bevatten.

3



Tabel 1.

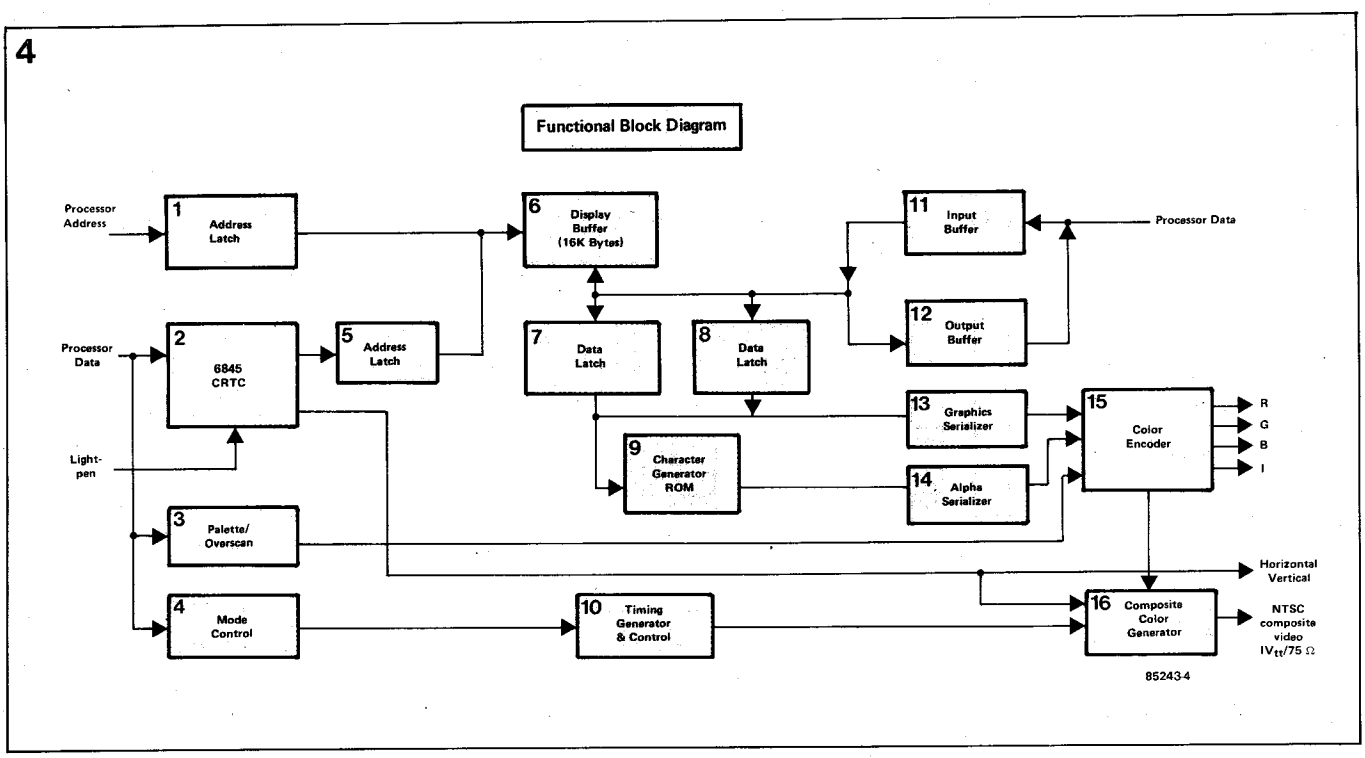
Attribuut funktie (beeldweergave)	Attribuut byte 7	6	5	4	3	2	1	0
	Bl.	R	G	B	I	R	G	B
	vg.	achtergrond			voorgrond (vg.)			
normaal	Bl.	0	0	0	I	1	1	1
geïnverteerd	Bl.	1	1	1	I	0	0	0
zwart beeld	Bl.	0	0	0	I	0	0	0
wit beeld	Bl.	1	1	1	I	1	1	1

Tabel 1 vervolg.

Attribuut funktie (beeldweergave)	Monochrome monitor		Kleuren-monitor	
	achter-grond kleur	karakter kleur	achter-grond kleur	karakter kleur
normaal	zwart	wit	zwart	wit
geïnverteerd	wit	zwart	wit	zwart
zwart beeld	zwart	zwart	zwart	zwart
wit beeld	wit	wit	wit	wit

I = verhoogde intensiteit van de voorgrond (karakter)
Bl. = "blinken" (knipperen) van de voorgrond (karakter)

Met de DIL-schakelaar op het moederbord kan bepaald worden in welke display-mode wordt opgestart. Na de power-on-reset kijkt het BIOS naar deze schakelaar om te zien wat de computer "aan boord" heeft. De DIL-kontakten **5** en **6** leveren de informatie over de videoadapter(s). Indien beide kontakten zijn gesloten (ON) is geen videokaart aanwezig. Deze instelling is dus alleen interessant bij speciale toepassingen van het moeder-board als bijv. besturingscomputer. Beide kontakten open houdt in dat alleen een monochrome adapter of meerdere adapters zijn geplaatst. Alleen de monochrome kaart wordt dan geïnitialiseerd (80 x 25). Kontakt **5 open** en kontakt **6 gesloten** initialiseert de kleurenvideokaart in de **40 x 25 mode**. In de omgekeerde situatie, kontakt **5 gesloten**



Figuuur 3. Met deze simpele schakeling kunnen de aparte signalen voor kleur- en synchronisatie worden samengevoegd tot een composite-videosignaal van hoge kwaliteit. Behalve voor de in dit artikel beschreven videokaart kan deze videocombiner vanzelfsprekend worden toegepast bij alle gangbare IBM-PC-kompatibele kleuren-videokaarten.

Figuur 4. Het blokschema van de kleuren-videokaart.

Tabel 2.

R	G	B	I	kleur
0	0	0	0	zwart
0	0	1	0	blauw
0	1	0	0	groen
0	1	1	0	cyaan
1	0	0	0	rood
1	0	1	0	magenta
1	1	0	0	bruin
1	1	1	0	wit
0	0	0	1	grijs
0	0	1	1	helder blauw
0	1	0	1	helder groen
0	1	1	1	helder cyaan
1	0	0	1	helder rood
1	0	1	1	helder magenta
1	1	0	1	geel
1	1	1	1	geïntensiveerd wit

Tabel 1. De gemeenschappelijke mogelijkheden van monochrome en kleuren-videokaart. Afwijkende bitpatronen voor R, G en B leveren bij de monochrome kaart witte karakters op een witte achtergrond.

Tabel 2. De kleurenkaart kan, dankzij de toepassing van een intensity-bit, zestien verschillende kleuren weergeven.

Tabel 3. De "medium" resolutie grafische mode heeft slechts beperkte kleurmogelijkheden. Men heeft de keuze uit twee paletten.

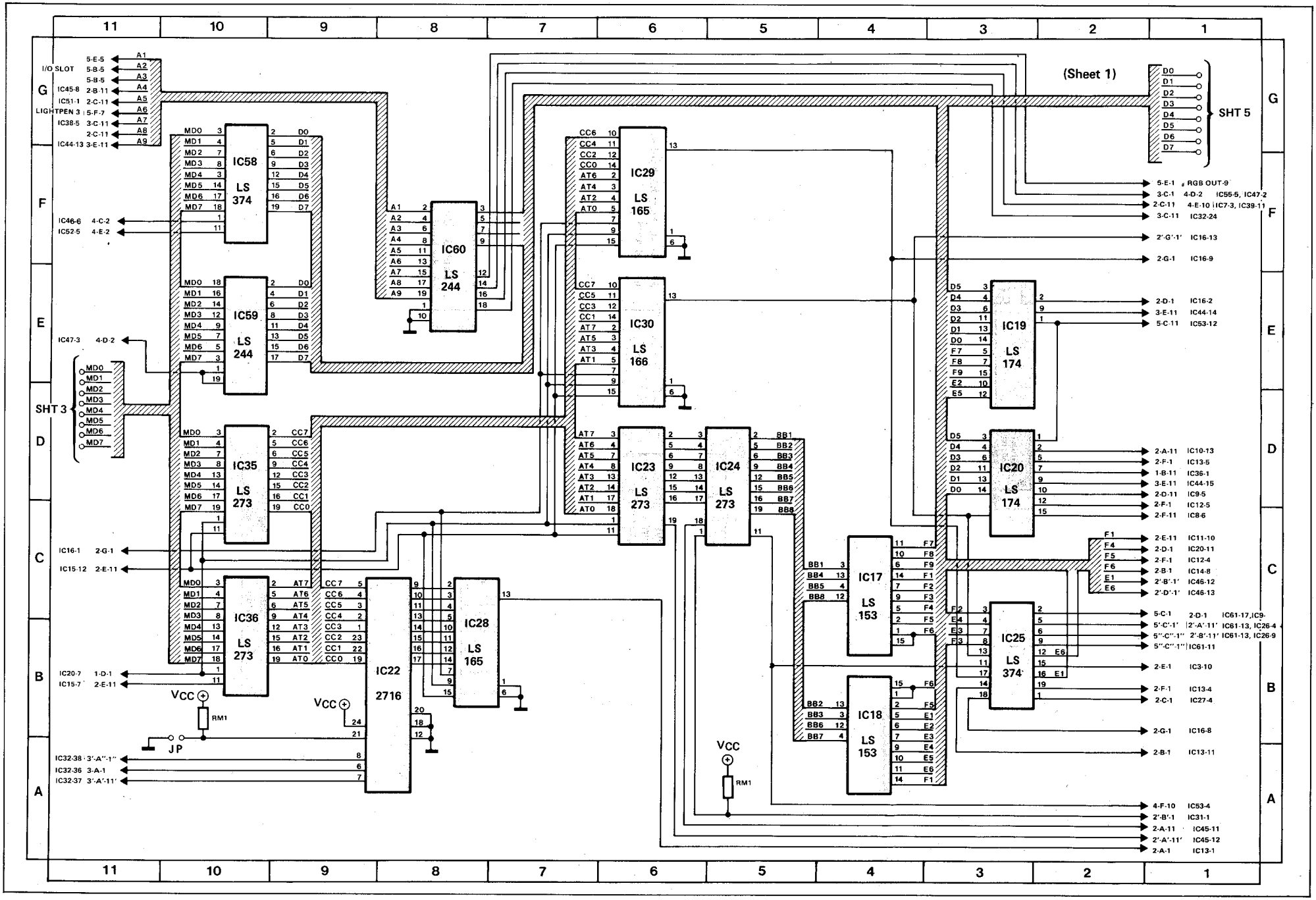
Tabel 3.

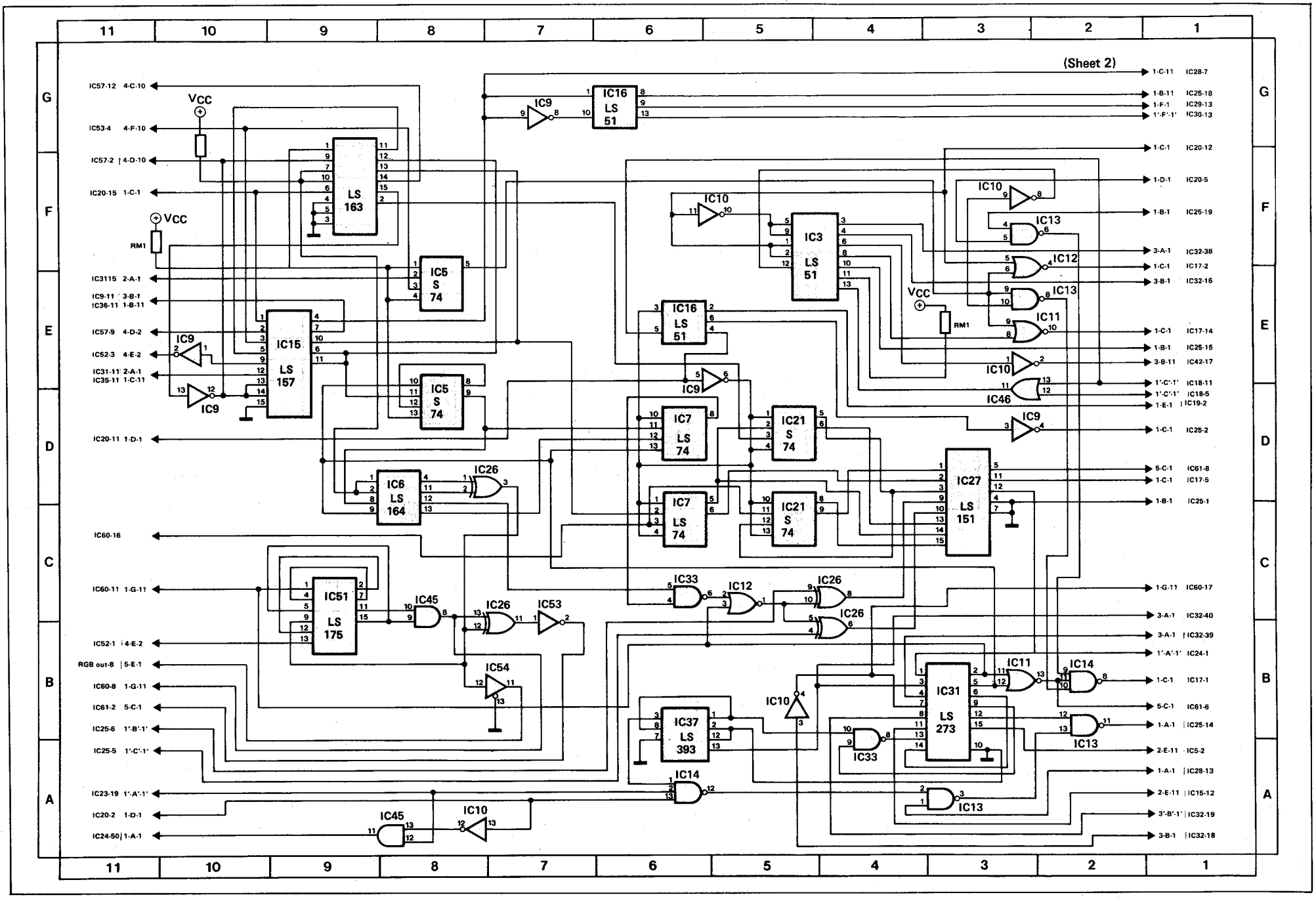
kleur	palet 1	palet 2
kleur 0	achtergrond	achtergrond
kleur 1	groen	cyaan
kleur 2	rood	magenta
kleur 3	bruin	wit

en kontakt **6 open**, zal de computer opstarten met de kleurenkaart in de **80 × 25 mode**.
Nog meer details over deze kleurenkaart is alleen interessant voor de doorgewinterde hardware-man. Daarom nog een korte blik op...

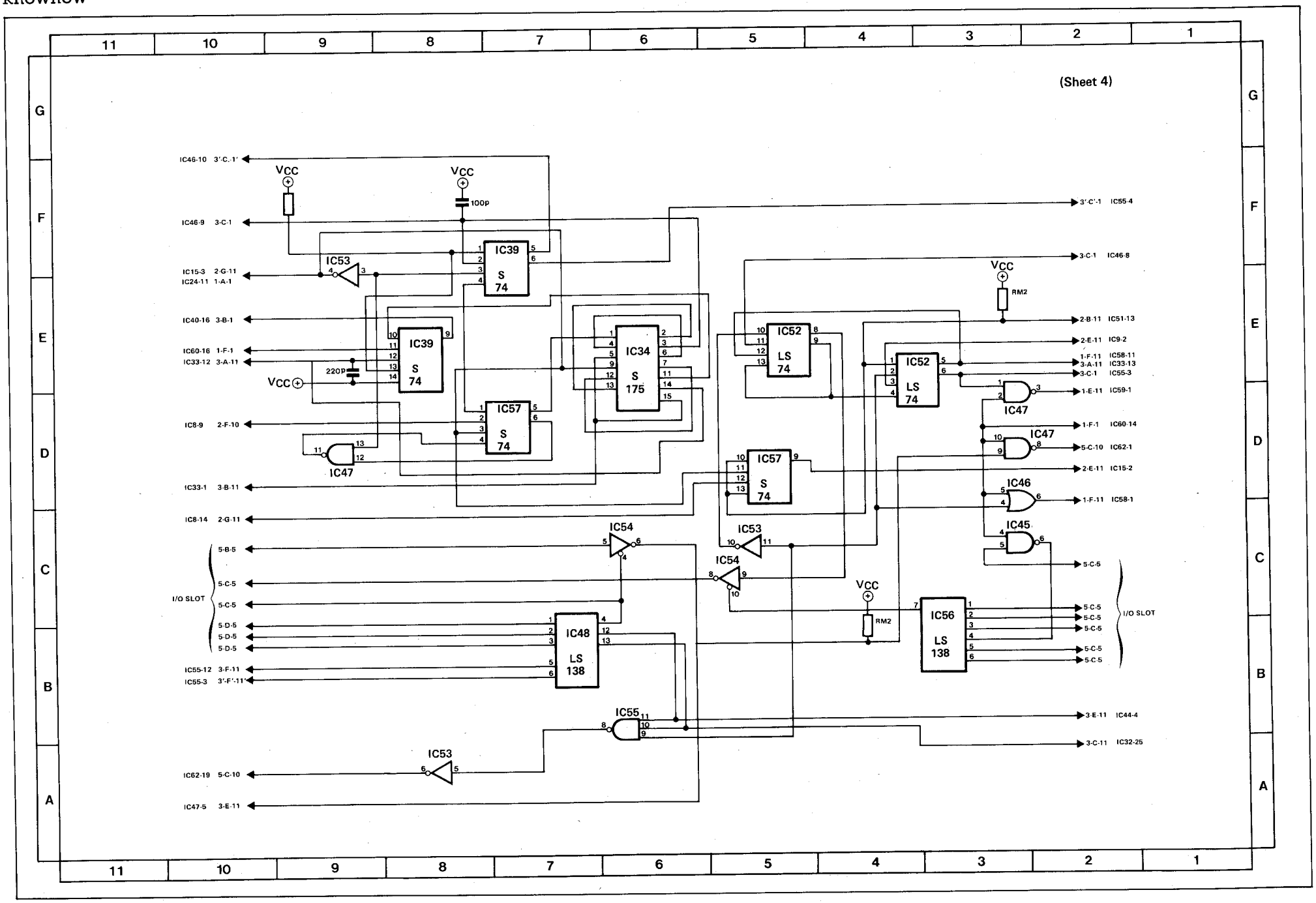
Het schema

Ook hier geldt: de gelukkige bezitter van het eigenlijke onmisbare manual (zie onder kleurenkaart) enz, enz. Helemaal hetzelfde zijn de "oer-"schakeling en het hier gepresenteerde exemplaar natuurlijk niet, maar de gelijkenis is treffend. Maar is dat niet juist een garantie voor de kwaliteit?
Een videokaart met 59 IC's levert een fors schema. Dit strekt zich dan ook uit over vijf pagina's. De beschrijving van een schakeling komt meestal neer op het standaard-verhaal: ICx doet dit en ICy doet dat. Het blokschema geeft de verschillende funkties echter duidelijk genoeg aan om in tabelvorm te kunnen weergeven welk IC bij welke funktie is betrokken.
Doordat het blokschema slechts een grove weergave van alle funkties is, zijn niet alle IC's in tabel 4 aangegeven. Niettemin

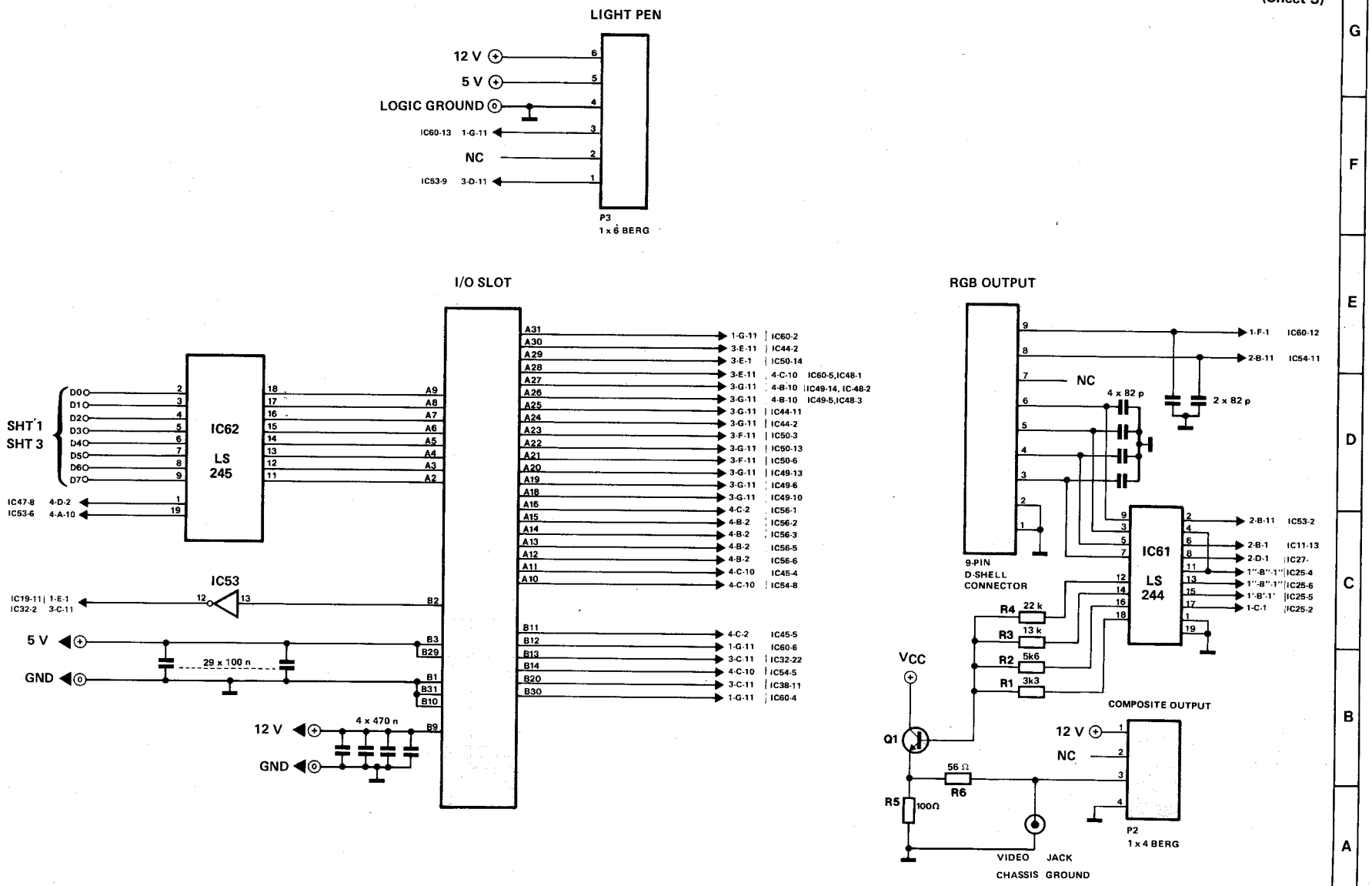

(Sheet 1)
I/O SLOT
LIGHTPEN 3
IC58 LS 374
IC59 LS 244
IC60 LS 244
IC29 LS 165
IC30 LS 166
IC35 LS 273
IC23 LS 273
IC24 LS 273
IC36 LS 273
IC22 2716
IC28 LS 165
IC19 LS 174
IC20 LS 174
IC17 LS 153
IC18 LS 153
IC25 LS 374
SHT 3
SHT 5
RM1
JP
VCC


(Sheet 2)
IC57-12 4-C-10
IC53-4 4-F-10
IC57-2 4-D-10
IC20-15 1-C-1
IC3115 2-A-1
IC9-11 3-B-1
IC36-11 1-B-11
IC57-9 4-D-2
IC52-3 4-E-2
IC31-11 2-A-1
IC35-11 1-C-11
IC20-11 1-D-1
IC60-16
IC60-11 1-G-11
IC52-1 4-E-2
RGB out-8 5-E-1
IC60-8 1-G-11
IC61-2 5-C-1
IC25-6 1'-B'-1'
IC25-5 1'-C'-1'
IC23-19 1'-A'-1'
IC20-2 1-D-1
IC24-50 1-A-1
Vcc
RM1
IC9
IC16 LS 51
IC10
IC3 LS 51
IC13
IC12
IC11
IC46
LS 163
IC5 S 74
IC15 LS 157
IC6 LS 164
IC26
IC7 LS 74
IC21 S 74
IC27 LS 151
IC51 LS 175
IC45
IC53
IC54
IC33
IC37 LS 393
IC14
IC31 LS 273
1-C-11 IC28-7
1-B-11 IC25-18
1-F-1 IC29-13
1'-F'-1' IC30-13
1-C-1 IC20-12
1-D-1 IC20-5
1-B-1 IC25-19
3-A-1 IC32-38
1-C-1 IC17-2
3-B-1 IC32-16
1-C-1 IC17-14
1-B-1 IC25-15
3-B-11 IC42-17
1'-C'-1' IC18-11
1'-C'-1' IC18-5
1-E-1 IC19-2
1-C-1 IC25-2
5-C-1 IC61-8
1-C-1 IC17-5
1-B-1 IC25-1
1-G-11 IC60-17
3-A-1 IC32-40
3-A-1 IC32-39
1'-A'-1' IC24-1
1-C-1 IC17-1
5-C-1 IC61-6
1-A-1 IC25-14
2-E-11 IC5-2
1-A-1 IC28-13
2-E-11 IC15-12
3'-B'-1' IC32-19
3-B-1 IC32-18

(Sheet 3)
I/O SLOT
5-D-5
IC48-5 4-B-10
IC48-6 4'-B-10'
IC54-6 4-A-10
I/O SLOT 5-D-5
IC48-12 4-B-2
IC60-19 1-G-11
IC19-9 1-E-1
IC20-9 1-D-1
LIGHTPEN 1 5-F-7
I/O SLOT 5-B-5 5-28
IC60-15 1-G-11
IC60-18 1-F-1
I/O SLOT 5-B-5 5-24
IC48-13 4-B-2
IC53-12 5-C-11
IC10-2 2-E-1
IC34-15 4-D-10
IC39-12 4-E-10
IC52-5 4-E-2
SHT5
D0
D1
D2
D3
D4
D5
D6
D7
VCC
RM2
IC53
IC44
LS 138
IC45
IC47
IC55
IC38
LS 74
IC33
IC32
6845
IC49
LS 257
IC50
LS 257
IC43
LS 374
IC42
LS 374
RR1
RR2
RR3
RR4
RR5
RR6
RR7
RR8
8 x 33 Ω
RM4
IC54
IC9
IC40
4416
IC41
4416
33
RM3
IC46
IC55
MD0
MD1
MD2
MD3
MD4
MD5
MD6
MD7
SHT 1
5-D-5 I/O SLOT
4-E-2
4'-F-2'
4-F-10
4'-F-10'
1-F-1
2-E-1
2-A-1
2'-A-1'
1'-A-11
1'-A-11'
2-B-1
2-C-1
IC51-4
IC10-3
IC31-8
IC22-6
IC22-7
2-F-1
IC22-8
IC3-3
IC15-7
IC52-6
IC39-6
IC69-14
IC39-2
IC39-5
IC39-9


(Sheet 4)
IC46-10 3'-C-1'
IC46-9 3-C-1
IC15-3 2-G-11
IC24-11 1-A-1
IC40-16 3-B-1
IC60-16 1-F-1
IC33-12 3-A-11
IC8-9 2-F-10
IC33-1 3-B-11
IC8-14 2-G-11
I/O SLOT
5-B-5
5-C-5
5-C-5
5-D-5
5-D-5
5-D-5
IC55-12 3-F-11
IC55-3 3'-F'-11'
IC62-19 5-C-10
IC47-5 3-E-11
VCC
100p
220p
RM2
IC39
S
74
IC53
IC57
IC34
S
175
IC47
IC54
IC48
LS
138
IC52
LS
74
IC56
IC55
IC46
IC45
3'-C'-1 IC55-4
3-C-1 IC46-8
2-B-11 IC51-13
2-E-11 IC9-2
1-F-11 IC58-11
3-A-11 IC33-13
3-C-1 IC55-3
1-E-11 IC59-1
1-F-1 IC60-14
5-C-10 IC62-1
2-E-11 IC15-2
1-F-11 IC58-1
5-C-5
5-C-5
5-C-5
5-C-5
5-C-5
I/O SLOT
3-E-11 IC44-4
3-C-11 IC32-25

(Sheet 5)


LIGHT PEN
12 V
5 V
LOGIC GROUND
IC60-13 1-G-11
NC
IC53-9 3-D-11
P3
1 x 6 BERG
I/O SLOT
RGB OUTPUT
IC62
LS
245
SHT 1
SHT 3
D0
D1
D2
D3
D4
D5
D6
D7
IC47-8 4-D-2
IC53-6 4-A-10
A9
A8
A7
A6
A5
A4
A3
A2
IC53
IC19-11 1-E-1
IC32-2 3-C-11
B2
5 V
GND
29 x 100 n
B3
B29
B1
B31
B10
12 V
GND
4 x 470 n
B9
A31
A30
A29
A28
A27
A26
A25
A24
A23
A22
A21
A20
A19
A18
A16
A15
A14
A13
A12
A11
A10
1-G-11 IC60-2
3-E-11 IC44-2
3-E-1 IC50-14
3-E-11 4-C-10 IC60-5,IC48-1
3-G-11 4-B-10 IC49-14, IC48-2
3-G-11 4-B-10 IC49-5,IC48-3
3-G-11 IC44-11
3-G-11 IC44-2
3-F-11 IC50-3
3-G-11 IC50-13
3-F-11 IC50-6
3-G-11 IC49-13
3-G-11 IC49-6
3-G-11 IC49-10
4-C-2 IC56-1
4-B-2 IC56-2
4-B-2 IC56-3
4-B-2 IC56-5
4-B-2 IC56-6
4-C-10 IC45-4
4-C-10 IC54-8
B11
B12
B13
B14
B20
B30
4-C-2 IC45-5
1-G-11 IC60-6
3-C-11 IC32-22
4-C-10 IC54-5
3-C-11 IC38-11
1-G-11 IC60-4
9-PIN
D-SHELL
CONNECTOR
NC
4 x 82 p
2 x 82 p
1-F-1 IC60-12
2-B-11 IC54-11
IC61
LS
244
2-B-11 IC53-2
2-B-1 IC11-13
2-D-1 IC27-
1''-B''-1'' IC25-4
1''-B''-1'' IC25-6
1'-B'-1' IC25-5
1-C-1 IC25-2
R4 22 k
R3 13 k
R2 5k6
R1 3k3
VCC
Q1
56 Ω
R6
R5 100Ω
VIDEO JACK
CHASSIS GROUND
COMPOSITE OUTPUT
12 V
NC
P2
1 x 4 BERG

verschaft deze tabel in kombinatie met het schema voldoende informatie voor de doorgewinterde hardwarepuzzelaar.
Ten slotte nog een paar opmerkingen over de onderdelenlijst. De met een ster gemerkte komponenten (m.u.v. RM3) kunnen vervallen indien geen gebruik wordt gemaakt van de NTSC-uitgang. Voor RM3 suggereert de komponentenopstelling een weerstandsnetwerk. Hier dienen echter drie weerstanden naast elkaar te worden geplaatst. Het is mogelijk dat de onderdelenlijst van de door de printleverancier meegeleverde onderdelenlijst. De "timing" van de kleurenkaart blijkt nogal kritisch te zijn. Dit uit zich onder andere door het spontaan wijzigen van de geheugeninhoud. De komponentenkeuze volgens tabel 5 voldoet in ons prototype het beste. Ook tijdens duurproeven is de kaart volledig stabiel gebleken.

Tabel 4.

blok	funktie	betrokken IC's
1	Address Latch	IC49, 50
2	6845 CRTC	IC32
3	Palette/Overscan	IC23, 24
4	Mode Control	IC48, 56
5	Address Latch	IC42, 43
6	Display Buffer	IC40, 41
7	Data Latch	IC36
8	Data Latch	IC35, 36
9	Character Generator ROM	IC22
10	Timing Generator & Control	IC38, 47
11	Input Buffer	IC62
12	Output Buffer	IC62
13	Graphics Serializer	IC29, 30
14	Alpha Serializer	IC28, (29, 30: attribuut)
15	Color Encoder	IC17, 18, 25, 61
16	Composite Color Generator	IC7, 21, 27, 61

Figuur 5. Het schema van de kleurenkaart. Maar liefst vijf pagina's zijn nodig om dit overzichtelijk af te beelden.

Tabel 4. Een wegwijzer in het doolhof. Deze tabel geeft aan welke IC's nauw betrokken zijn bij de in het blokschema aangegeven funkties.

Tabel 5. De in deze lijst genoemde onderdelen werden in ons proefmodel toegepast. Kleine afwijkingen met de door de printleverancier meegeleverde lijst zijn mogelijk. Overigens worden alle IC's op de print met Z aangeduid.

Figuur 6. Er zijn verschillende kleuren videoadapters verkrijgbaar. Let op de komponentenopstelling van dit exemplaar. Dit is de kaart die door ons met sukses werd beproefd.

Onderdelenlijst:

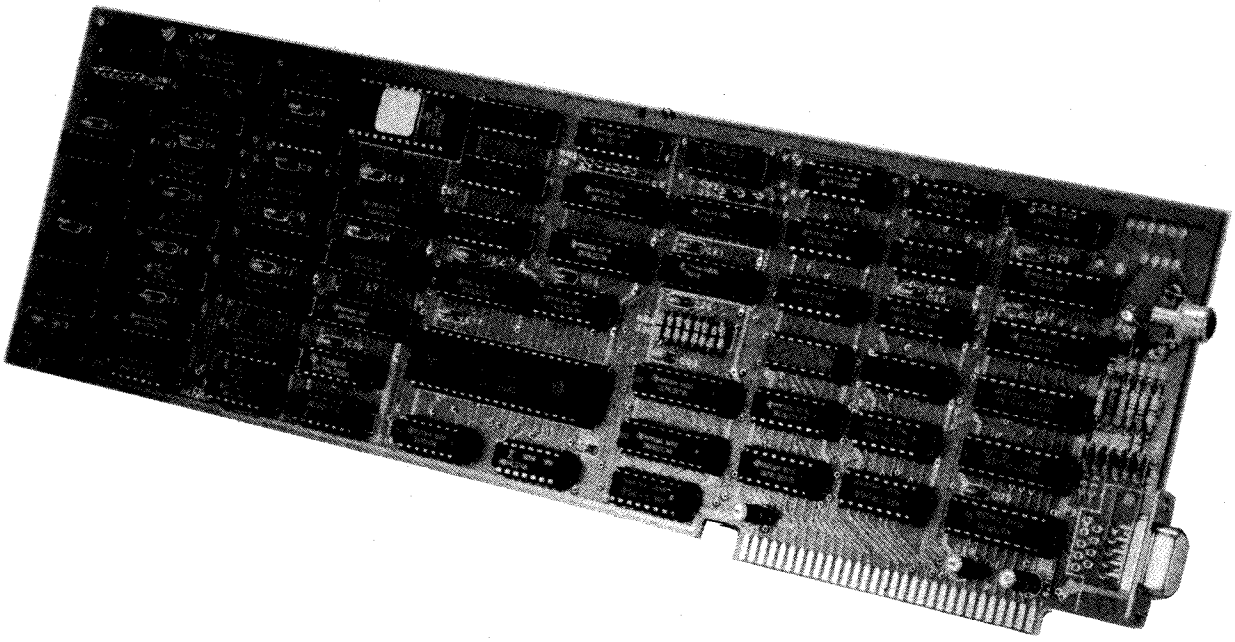
Attentie: Alle IC's zijn op de print met Z aangegeven. De met "*" aangeduide komponenten zijn alleen nodig voor de NTSC-versie.

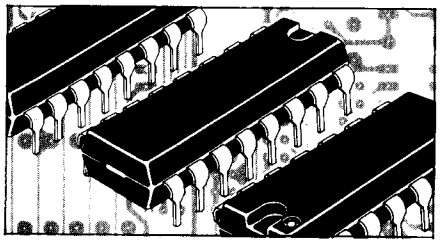
Weerstanden:
RM1,RM2 = weerstandsnetwerk 7 × 2k
RM3 = 3 weerstanden 33 Ω (geen weerstandsnetwerk)
RM4 = weerstandsnetwerk 8 × 33 Ω
*R1 = 3k3
*R2 = 5k6
*R3 = 12k
*R4 = 2k2
*R5 = 100 Ω
*R6 = 56 Ω

Kondensatoren:
C1...C23,C25...C28 = 100 n
C24,C29,C30,C37 = 47 μ/16 V
*C31...C36 = 47 p

Halfgeleiders:
*Q1 = BC547B (2N3904)
IC1,IC2,IC4 = niet aanwezig
IC3,IC16 = 74LS51
IC5,IC38,IC39,IC52,IC57 = 74LS74
IC6 = 74LS164
*IC7 = 74LS74
IC8 = 74LS163
IC9,IC10 = 74LS04
IC11,IC12 = 74LS02
IC13 = 74LS00
IC14,IC55 = 74LS10
IC15 = 74LS157
IC17,IC18 = 74LS153
IC19,IC20 = 74LS174
*IC21 = 74S74
IC22 = 2716 (karaktergenerator)
IC23,IC24,IC31,IC35,IC36 = 74LS273
IC25,IC42,IC43,IC58 = 74LS374
IC26 = 74LS86
*IC27 = 74LS151
IC28...IC30 = 74LS166
IC32 = (MC)6845
IC33,IC47 = 74S00
IC34 = 74S175
IC37 = 74LS393
IC40,IC41 = TMS4416
IC44,IC48,IC56 = 74LS138
IC45 = 74LS08
IC46 = 74S32
IC49,IC50 = 74LS257
IC51 = 74LS175
IC53 = 74S04
IC54 = 74LS125
IC59...IC61 = 74LS244
IC62 = 74LS245

6





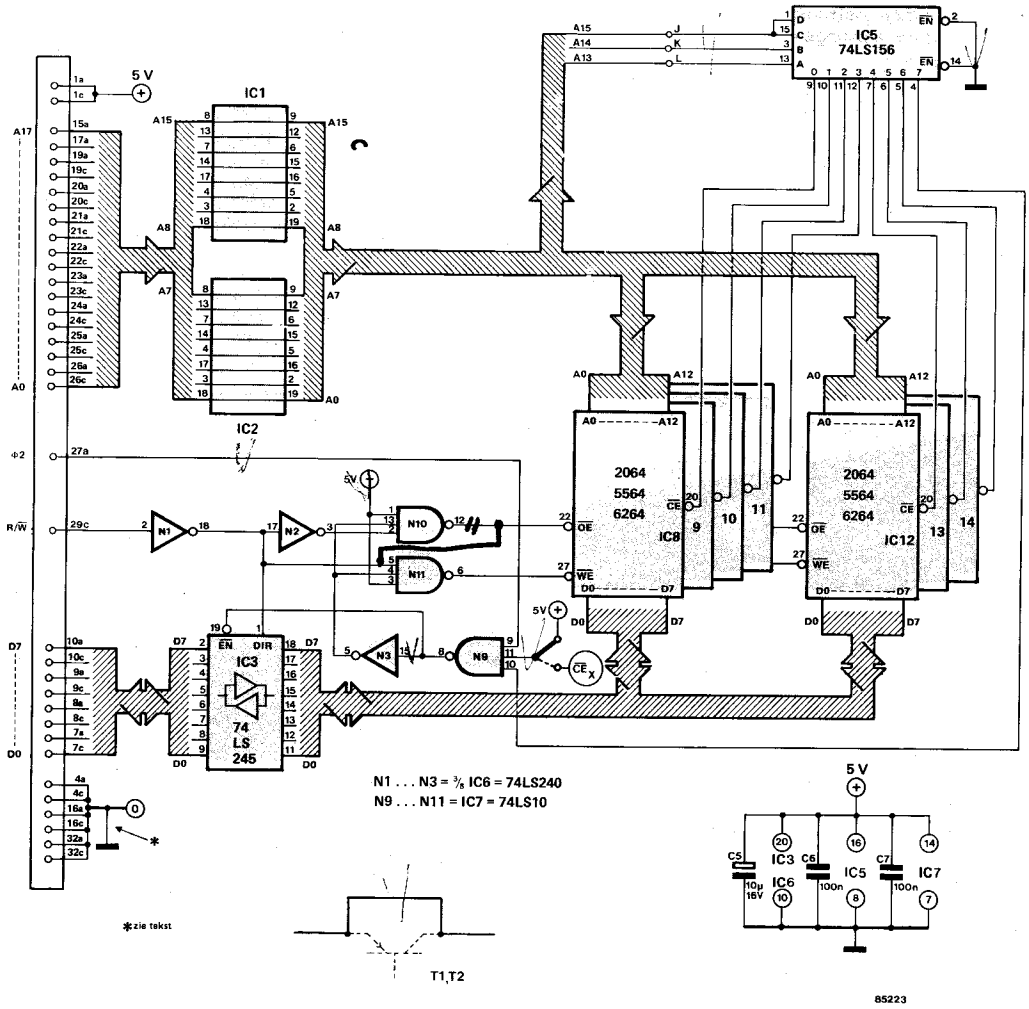
Octopus-65: statische RAM-kaart

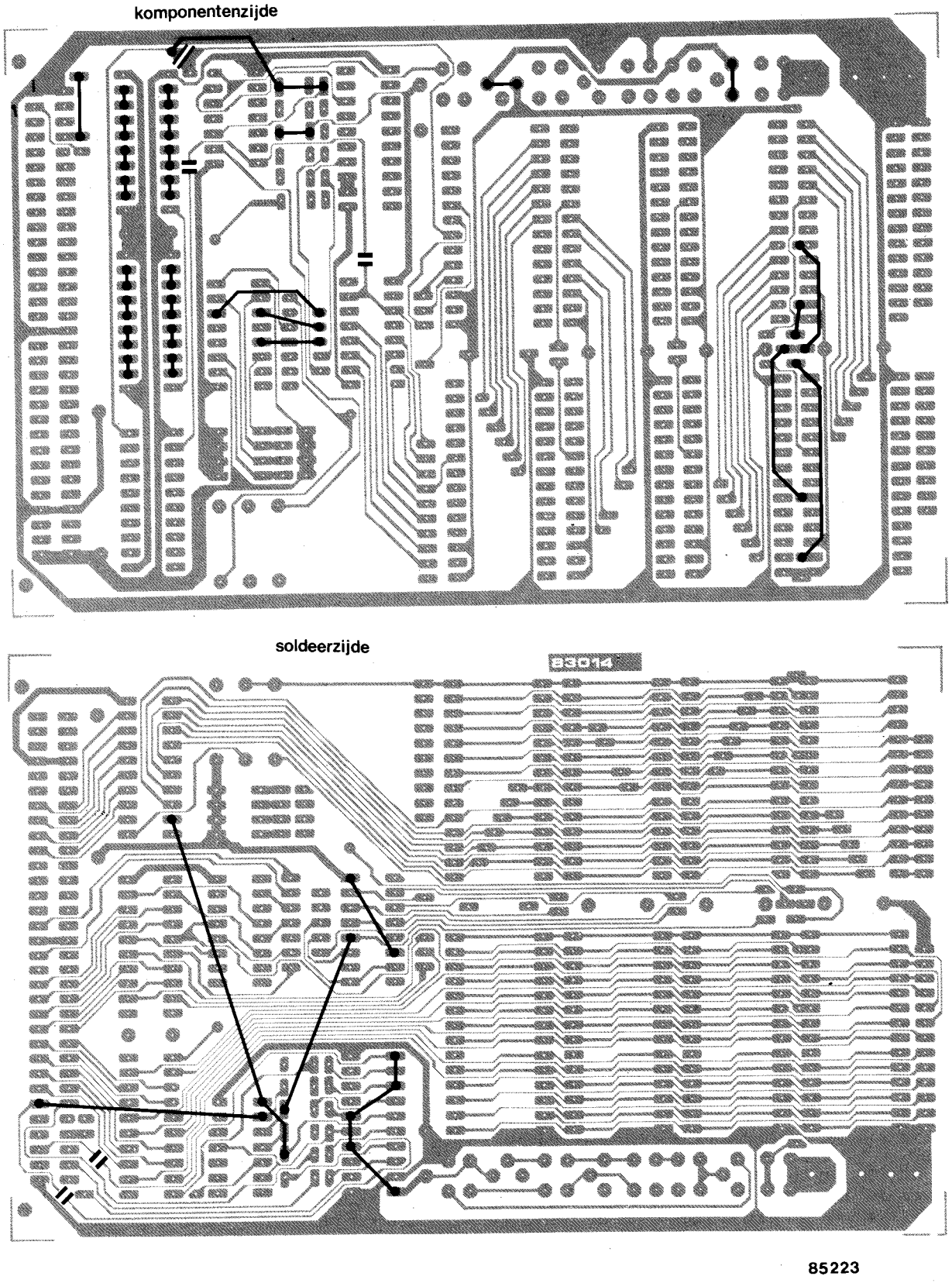
Ofschoon de dynamische RAM's heden ten dage haast aan "schroot"-prijzen worden verkocht, heeft de statische RAM nog niets van zijn charme verloren. Destijds was de "universele geheugenkaart" in Elektuur maart '83 een gewild nabouwprojekt voor gebruikers van zowel RAM's als EPROM's. Vandaar dat we deze kaart als basis hebben gekozen voor de RAM-bezetting van de nieuwe EC-65K (met de 65SC816-CPU). Ook kan de kaart gebruikt worden voor de Octopus-65. In beide gevallen kan er heel wat vervallen van de oorspronkelijke opzet.

Alvorens we beginnen met de op- en ombouw van de statische RAM-kaart, zullen we voor de liefhebbers alvast wat verklappen van de EC-65K. Er worden twee printen gebruikt die de CPU-kaart van de Octopus vervangen. Eén ervan is de nieuwe CPU-kaart, die geschikt is voor alle 65-processoren, en de andere bevat naast een interface ook nog een real-time-clock. Wie in de Octopus al gebruik heeft gemaakt van de "universele geheugenkaart", zal met de EC-65K-modifikatie van deze kaart snel klaar zijn. De DRAM-kaart daarentegen kan men niet meer gebruiken; deze had voor de timing het Φ1-signaal van de processor nodig en dat signaal ontbreekt bij de 65SC816. De oorspronkelijke aansluiting op de 6502 (pen 3) is nu bevorderd tot ABORT-signaal.
Dan nog iets over de geheugenbezetting van de kaart. Voor de Octopus en de EC-65K in basisbezetting (dat betekent: alleen RAM in bank 0) worden dat zeven 8-Kbyte-RAM's (2064, 5564, 6264). Hoe snel de RAM's moeten zijn, hangt natuurlijk af van de klok-snelheid van het systeem. Bij een proefmodel van de EC-65K funktioneerde de geheugenkaart met 150-ns-RAM's nog bij 4 MHz. In principe is de timing dan wel erg krap. We raden u dan ook aan om maar meteen 120-ns-RAM's te kopen. Dat is altijd safe!
Natuurlijk kan men ook meerdere geheugenkaarten opbouwen voor de EC-65K, want tenslotte heeft de 65SC816 de moge-

Figuur 1. De schakeling van de statische RAM-kaart is afgeleid van de "universele geheugenkaart" uit Elektuur. Doordat er is afgezien van een battery-backup-mogelijkheid, kon er heel wat worden uitgespaard.

1





Figuur 2. Voor toepassing in de Octopus zijn hier de diverse draadbruggen en onderbrekingen aangegeven. Wie de kaart later voor de EC-65K wil gebruiken, doet er goed aan om de "kras-gegevens" betreffende pen 16a/c vooraf uit te voeren (zie tekst).

lijkheid om 16 Mbyte te adresseren. Dit is echter heel wat ruimte om op te vullen met statische RAM's. Elke 64-K-bank zou dan acht 8-KB-IC's moeten bevatten, dus tel maar op. In de toekomst zal ook de geheugengrootte van de RAM-IC's wel groeien. De 32-KB-RAM is al "in wording" en zal over niet al te lange tijd leverbaar en betaalbaar zijn voor de hobbyist. Eén kaart met 8 IC's zal dan ongeveer even duur uitvallen als vier kaarten met ieder 8 (8-KB-) IC's. We zullen dus nog even geduld moeten hebben...

Voor de software van de EC-65K hoeft men niet te wachten: die is er "van begin af aan". De software is zó geschreven dat de originele Octopus-software ook kan draaien, zij het met enkele kleine veran-

Onderdelenlijst

Kondensatoren:
C1...C4,C6,C7 = 100 n
C5 = 10 μ/16 V

Halfgeleiders:
IC1,IC2,IC4,IC15: worden niet aangebracht
IC3 = 74LS245
IC5 = 74LS155
IC6 = 74LS240
IC7 = 74LS10
IC8...IC14 = 6264 (2064,5564)

Diversen:
haakse konnektor, 64-polig male (DIN 41612)

Let op: voor een 4-MHz-systeem bevelen we voor IC8...IC14 120-ns-types aan (bijvoorbeeld Hitachi). Voor 2 MHz kan men 150-ns-RAM's gebruiken.

deringen (enkele tijdlussen, die men zelf eenvoudig kan (her)programmeren). De snelheid zal wel fors omhoog gaan: de processor "draait" immers nu op 4 MHz. Nieuwe software die gebruik maakt van de nieuwe instrukties en adresseringsmogelijkheden, zit al in de pen.
Voordat we beginnen met de kaart zelf is er nog een belangrijk punt om rekening mee te houden: de aansluitingen op de bus van de EC-65K zijn ietwat gewijzigd ten opzichte van die bij de Octopus. Wanneer u voor de Octopus net een nieuwe kaart wilt bouwen, moeten de aanwijzingen in dit artikel goed opgevolgd worden!

De modifikaties

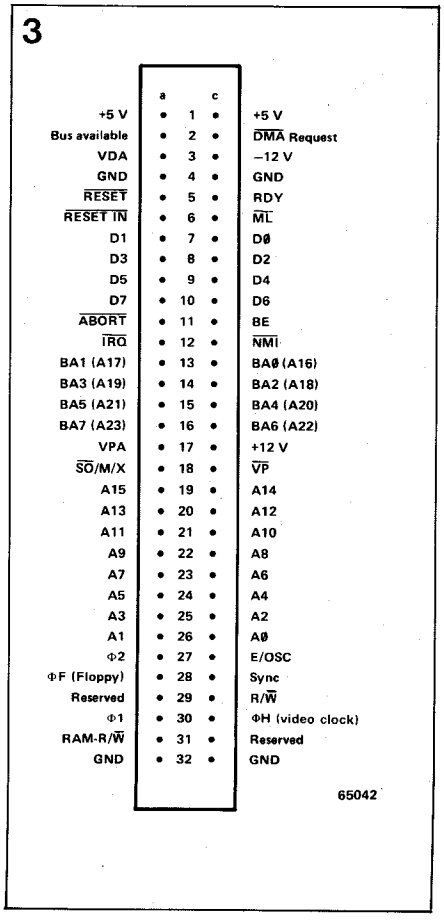
Zoals eerder gezegd, is de kaart afgeleid van de eerder in Elektuur beschreven "universele geheugenkaart". Laten we de uitgeklede versie eens bekijken (figuur 1):
Omdat de adressering nu "netjes" van 0000 tot FFFF loopt (eigenlijk niet tot FFFF, maar voor RAM tot DFFF), kan de adresdekodering extreem simpel worden gehouden. IC5 dekodeert uit de hoogste drie adreslijnen de CE-signalen (Chip Enable) voor de afzonderlijke geheugen-IC's (IC8...IC14). Verder zijn er wat poorten voor het verzorgen van de signalen $\overline{OE}$ (Output Enable) en $\overline{WE}$ (Write Enable), die gemaakt worden uit Φ2 en R/$\overline{W}$. Hieruit worden tevens de signalen DIR (DIRection) en $\overline{EN}$ (ENable) voor de databusbuffer IC3 afgeleid. Ten opzichte van de Elektuur-kaart zijn verder de adresbusbuffers weggelaten (de aansluitingen worden nu overbrugd). Er zit toch al een buffer op de CPU-kaart en de timing zou bij de 4-MHz-versie met 150-ns-RAM's in de problemen kunnen komen als men de tweede set buffers zou monteren.
Voor de SRAM-kaart gebruiken we dezelfde print als bij de originele Elektuur-versie. Wat hieraan veranderd moet worden, zien we in figuur 2. Aan de bovenzijde van de print (de komponentenzijde) moet men een printbaan doorkrassen en aan één kant van het gekraste spoor een draadbrug leggen (hiertoe moet een klein stukje koper blank gemaakt worden). Dan zijn er nog twee printbanen die alleen maar onderbroken hoeven te worden en een aantal doorverbindingen die men moet leggen met geïsoleerd draad. Dan is de onderzijde (soldeerzijde) aan de beurt. Ook hier moeten twee banen worden onderbroken en moeten er diverse doorverbindingen worden gelegd. Alleen de in de stuklijst genoemde komponenten moeten worden gemonteerd. Tot zover is dat voldoende voor de bouw van de 2-MHz-Octopus.

Voorbereidingen voor de EC-65K

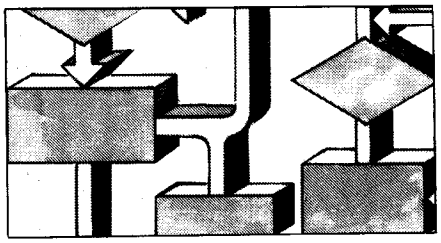
In figuur 3 vinden we de aansluitgegevens van het EC-65K-systeem. Er zijn weliswaar nieuwe signalen bijgekomen, maar die behoren geen hindernis te vormen bij eventuele uitbreiding op de bus. Mocht dit bij de een of andere Elektuur-kaart het geval zijn, dan kan er op de print zelf gemakkelijk een draadje worden gelegd naar een andere aansluiting.

Helaas zijn er toch een paar "roet-in-het-eten-gooiers", namelijk de aansluitingen 16a en 16c. Tot nu toe bevond zich hier altijd massa. De verbinding tussen deze pennen en de massabanen van de print kan zich bevinden aan de — goed bereikbare — soldeerzijde van de print maar ook aan de bovenkant, onder de buskonnektor! Een gewaarschuwd mens telt voor twee en dat betekent in dit geval dat eenieder die nu of in de toekomst de EC-65K wil bouwen, het beste de bewuste aansluitingen 16a en 16c meteen van de massalijn en van elkaar scheidt. Een buskonnektor desolderen zonder beschadiging van de rest van de printplaat, is geen eenvoudige opgave! "Nieuwbouwers" kunnen hun printen direkt behandelen. Let er wel op dat de massalijnen op de print niet onderbroken raken (eventueel met draadbruggen herstellen).
Als voorbeeld nemen we de "grafische kleurenkaart". Bij de basiskaart (EPS 85080-1) moet aan de bovenzijde de verbinding tussen 16a en 16c én de verbinding tussen 16a en de rest worden onderbroken. Men hoeft geen extra draadbruggen te leggen. Deze ingreep is dus eenvoudig. Bij de kleurgeheugenkaart (EPS 85080-2) moet men aan de bovenzijde de verbinding tussen 16a en de "passerende" massalijn verbreken en tevens de verbinding tusses 16a en 16c (aan de onderzijde) doorkrassen.
Degenen die de "universele geheugenkaart" reeds hebben opgebouwd, hoeven deze klus niet uit te voeren: in de volgende uitgave van Elektuur Computing zullen we de bewuste truuk uit de doeken doen. Laat de soldeerbout, desoldeerpomp, zijkniptang en ander grof materiaal dus nog maar even rusten: geduld is het devies!

Figuur 3. De aansluitgegevens van de EC-65K-bus. Het belangrijkste verschil met de vorige reeks Elektuur-bussen zit in pen 16a en 16c. Deze zaten gewoonlijk aan massa maar nu zijn ze "bevorderd" tot extra adreslijnen van de 65SC816.



PROGRAMMA'S



grafische programma's

Het grafische kleurenkaart-projekt, dat enige tijd geleden in Elektuur is beschreven, kan eigenlijk niet zonder enkele demonstratieprogramma's die laten zien welke mogelijkheden deze kaart bezit. Dat dacht een van onze kollega's van de Franstalige Elektuur-redaktie, en hij voegde meteen de daad bij het woord. Het resultaat: het ene fraaie programma na het andere...

Met de in EC2 en in deze uitgave gepubliceerde programma's is de voorraad overigens nog lang niet uitgeput. Daarbij is het demonstratie-effekt een beetje op de achtergrond geschoven en worden de grafische mogelijkheden momenteel voor meer serieuze zaken gebruikt. Intussen hebben de ontwerpers ook weer lustig geknutseld aan de grafische kleurenkaart: aansluiting op een 2-MHz-Octopus is reeds mogelijk, kleuren kunnen knipperen en geïnverteerd worden, enzovoorts. Maar dat zijn dingen voor de toekomst, laten we nu maar eens gaan kijken naar de hier afgedrukte programma's. De bij de programma's behorende kleurenplaatjes kunt u vinden op pagina 3 en 4 in deze Elektuur Computing. Hierbij is soms gewerkt met een intensiteitsbit. Daarvoor is wel een kleine ingreep in de monitor noodzakelijk, als deze van huis uit nog geen intensiteitsingang bezit. In totaal vindt u hier tien programma's, waarbij de nummering van de programma's overeenkomt met de nummers van de foto's op pagina 3 en 4.

Tenslotte nog een opmerking over de automatische demonstratiediskette die vele lezers hebben zien draaien op de HCC-dagen die eind 1984 in Utrecht werden gehouden. Menigeen heeft zich afgevraagd hoe die demonstratiediskette werkte, dus hoe de verschillende demoprogramma's automatisch werden doorlopen. Nou, dat is vlug verklaard. Op een geschikte plaats van elk grafisch programma — meestal de voorlaatste programmaregel — wordt het kommando

RUN"NN

gezet (NN = naam van het grafische programma dat na het momentele programma moet starten). De Octopus-bezitters weten onderhand wel dat met

RUN"file-naam"

een BASIC-programma van de diskette kan worden opgehaald en meteen wordt gestart. Bij een dubbelzijdige drive moet natuurlijk nog

DISK!"SE X":

voor het RUN"NN"-kommando worden geplaatst, zodat de Octopus weet op welke kant van de zwarte schijf hij het bewuste grafische programma moet zoeken.

Nadat men dit heeft gedaan, worden alle zo aan elkaar geknoopte programma's achter elkaar doorlopen. Misschien iets voor een etalage of een disko? Wie weet waar we de Octopus met grafische kaart straks nog zullen tegenkomen!

1. SYMMET

```
10 DISK!"GO C000":DISK!"IO ,04"
20 PRINTCHR$(2)CHR$(18)
30 PRINT"M255,125,I,B3,C4
40 FORF=.017453TO6.35STEP.017453/2
50 X=INT(COS(69*F)*SIN(14*F)*255+255)
60 Y=INT(COS(69*F)*COS(14*F)*127+127)
70 PRINT"D"X","Y:NEXT:PRINTCHR$(2)
80 FORZ=1TO10: PRINTCHR$(1):NEXT
90 DISK!"IO ,01"
```

2. DODEKA

```
10 DISK!"GO C000":DISK!"IO ,04":PRINTCHR$(2)CHR$(18)
20 PRINT"B3,M255,,":DIMX(13),Y(13)
30 X(1)=255: X(7)=X(1): Y(7)=X(1): X(2)=135
40 Y(2)=20: X(3)=42: Y(3)=60: Y(4)=125
50 X(5)=X(3): Y(5)=195: X(6)=X(2): Y(6)=235
60 X(8)=377: Y(8)=Y(6): X(9)=470: Y(9)=Y(5)
70 X(10)=510: Y(10)=Y(4): X(11)=X(9): Y(11)=Y(3)
80 X(12)=X(8): Y(12)=Y(2): X(13)=X(1)
90 FOR N=1TO70:FORD=1TO12:X=X(D+1):Y=Y(D+1)
100 PRINT"C"INT(D/2AND6)",D"X","Y
110 X(D)=X(D)+INT((X(D+1)-X(D))/9+1.6)
120 Y(D)=Y(D)+INT((Y(D+1)-Y(D))/9+.8)
130 NEXT:X(13)=X(1):Y(13)=Y(1):NEXT
140 DISK!"IO ,01
```

3. COLLAG

```
10 DISK!"IO ,04":DISK!"GO C000"
20 PRINTCHR$(18)"B6,C0":GOSUB570
30 PRINT"L0,C4,M255,,I,0255,100,100
40 PRINT"L3,C3,0255,90,90,C4
50 PRINT"H,G-1,-40,70,G-1,40,70
60 PRINT"M255,124,I,0255,190,190,C2,M255,194,I,0255,190,190,C3
70 PRINT"L2,M255,124,I,0255,190,190,C1,M255,194,I,0255,190,190
80 REM
90 PRINT"L0,C12,M53,176,I,060,95,95
100 PRINT"M453,176,I,0195,95,95
110 REM
120 PRINT"C1,M3,160,I,G-1,506,32
130 PRINT"C3,M13,150,I,G-1,486,52
140 PRINT"C2,M23,140,I,G-1,466,72
150 PRINT"C5,M33,134,I,G-1,446,84
160 PRINT"C10,M53,124,I,G-1,406,104
170 PRINT"C6,M63,127,I,G1,386,98
180 PRINT"C1,M64,128,I,G-1,384,96
190 PRINT"C12,M67,130,I,G1,378,92
200 PRINT"C2,M72,150,I,G1,368,52
210 PRINT"C4,M73,151,I,G1,366,50
220 PRINT"C2,M74,152,I,G1,364,48
230 PRINT"C6,M75,153,I,G1,362,46
240 REM
250 PRINT"M256,176,I,C3,G-2,-196,78,-2,196,78
260 PRINT"C11,G-2,-196,72,-2,196,72
270 PRINT"C2,G-2,-196,67,C2,G2,-196,63
280 PRINT"C1,G2,-196,60,C2,G2,196,57
290 PRINT"C1,G2,-196,54,C2,G2,-196,51
300 REM
310 PRINT"G2,-196,-148,G2,196,-148
320 PRINT"C2,G-2,196,67,C2,G2,196,63
330 PRINT"C1,G2,196,54,C2,G2,196,51
340 PRINT"C1,G2,196,60,C2,G2,196,57
350 REM
360 PRINT"C11,G-2,-194,45
370 PRINT"G-2,-194,-146,G-2,194,-146
380 PRINT"G-2,194,46,G-2,194,-146
390 PRINT"C3,G-2,-190,40,G-2,190,40
400 PRINT"G-2,-190,-138,G-2,190,-138
410 PRINT"C4,G-2,-100,-120,G-2,100,-120
420 REM
430 PRINT"C1,R-85,,G-2,-84,19,G-2,85,19"
440 PRINT"C9,G-2,-80,-9,G-2,80,-9,"
450 PRINT"C1,R170,,G-2,-84,19,G-2,85,19"
460 PRINT"C9,G-2,-80,-9,G-2,80,-9,R-170,,I"
470 PRINT"C6,0240,16,14,C11,0240,12,12
480 PRINT"C3,0240,8,8,C6,0240,4,4,R170,2,I"
490 PRINT"C6,0240,16,4,C11,0240,12,12
500 PRINT"C3,0240,8,8,C6,0240,4,4
510 REM
520 PRINT"R-89,-90,I,C1,H,G-2,-20,99,G-2,20,99
530 PRINT"C0,H,G-2,-1,255,G-2,1,255
540 PRINT"C6,R,-4,G-2,-20,-23,G-2,20,-23
550 REM
560 DISK!"IO ,01":END
570 PRINT"L1":FORX=0TO512STEP16
580 PRINT"M"X",,I,J,255":NEXT
590 FORY=0TO255STEP8
600 PRINT"M,"Y",I,J511,,":NEXT
610 PRINT"L2
620 RETURN
```

4. TESTXX

```
10 DISK!"GO C000":DISK!"IO ,04":PRINTCHR$(18)"C8,A";
20 FORL=0TO31:FORT=1TO4:PRINT"TEXT";:IFL=7ORL=15ORL=23ORL=31THEN80
30 IFL=6ANDT=1ORL=6ANDT=4THENPRINTCHR$(18)"C"L+8",S4,7,ATEST";:GOTO110
40 IFL=30ANDT=1ORL=30ANDT=4THENPRINTCHR$(18)"C12,S4,7,ATEST";:GOTO110
50 IFL=22ANDT=2THEN70
60 GOTO100
70 T=T+1:PRINTCHR$(18)"C7,S9,14,G-1,214,120,C11,ATEST";:GOTO110
80 PRINTCHR$(18)"R,1,I,H":FORF=1TO4:PRINT"C"F+L",G-1,24,6,R24,,I,H
90 NEXT:PRINT"R,-1,I,H":GOTO110
100 PRINTCHR$(18)"C"L",S4,1,ATEST";
110 PRINTCHR$(18)"C8,S1,1,A";
120 NEXT:IFL<31THENPRINTL:NEXT:GOTO140
130 PRINTL;:NEXT
140 DISK!"IO ,01
```

5. INUIT

```
10 DISK!"GO C000":DISK!"IO ,04":PRINTCHR$(18)"B2
20 PRINT"M255,110,I,L2,C9,0255,160,160
30 PRINT"M255,172,I,C3,G-2,-106,85,G-2,106,85
40 PRINT"M255,173,I,L0,C12,G-2,-104,80,G-2,104,80,L1
50 PRINT"M255,174,I,C3,G-2,-102,75,G-2,102,75
60 PRINT"M255,175,I,C6,G-2,-100,70,G-2,100,70
70 PRINT"C0,L0,R,-1,I,G-1,-25,-19,G-1,25,-19,L1
80 PRINT"C14,M255,138,I,G-1,-122,24,G-1,147,24,C4
90 PRINT"M255,140,I,G-1,-120,21,G-1,120,21
100 PRINT"C14,G-1,-62,-75,G-1,62,-75,C4
110 PRINT"G-1,-60,-67,G-1,60,-67
120 PRINT"M235,65,I,G-1,-40,14,M235,79,I,G-2,40,-15
130 PRINT"M275,65,I,G-1,40,14,M275,79,I,G-2,-40,-15
140 PRINT"C14,M316,139,I,G-2,17,-7,M194,139,I
150 PRINT"G-2,-17,-7,M229,162,I,G-2,-52,4
160 PRINT"M281,162,I,G-2,52,4,C4,M316,139,I
170 PRINT"G-2,15,-5,M194,139,I,G-2,-15,-5
180 PRINT"M229,162,I,G-2,-50,2,M281,162,I,G-2,50,2
190 PRINT"C2,M135,161,I,G-2,15,-5,C11,L0
200 PRINT"M235,64,I,G-1,-40,-60,M275,64,I,G-1,40,-60
210 PRINT"C11,M235,,I,G-1,-60,15,M275,,I,G-1,60,15
220 PRINT"M174,,I,G-2,-20,15,M336,,I,G-2,20,15,C4
230 PRINT"M235,,I,G-2,-60,40,M275,,I,G-2,60,40
240 PRINT"C11,M226,,I,G-1,-37,,M284,,I,G-1,37,,L1
250 PRINT"C14,M366,140,I,G-1,36,82,M400,220,I,G-1,-34,-7,C4
260 PRINT"M368,140,I,G-1,32,80,M400,220,I,C11,G-1,-32,-5
270 PRINT"C0,L0,M360,220,I,G-2,40,30
280 PRINT"M360,221,I,G-1,-8,3,R,5,I,G-1,-8,3,R,5,I,G-1,-8,3
290 PRINT"R,5,I,G-1,-8,3,R,5,I,G-1,-8,3,R,5,I,G-1,-8,3,L1
300 PRINT"C14,M133,139,I,G-1,36,-77,M135,64,I,G-1,34,7,C4
310 PRINT"M135,139,I,G-1,32,-75,C11,M135,64,I,G-1,32,5
320 PRINT"C0,L0,M175,64,I,G-2,-40,-30
330 PRINT"R,-1,I,G-1,8,-3,R,-5,I,G-1,8,-3,R,-5,I,G-1,8,-3
340 PRINT"R,-5,I,G-1,8,-3,R,-5,I,G-1,8,-3,R,-5,I,G-1,8,-3
350 PRINT"M255,165,I,0240,25,25,C14,0240,28,2
360 PRINT"C12,0240,32,2,C3,M150,145,I":FORI=1TO20
```

```
370 PRINT"G-2,-6,-3,G-2,-6,3,G-2,6,-3,G-2,6,3,R12,,I":NEXT
380 PRINT"M144,140,I":FORI=1TO11
390 PRINT"G-2,6,-3,G-2,-6,-3,G-2,6,3,G-2,-6,3,R,-6,I":NEXT
400 PRINT"M387,149,I":FORI=1TO11
410 PRINT"G-2,6,-3,G-2,-6,-3,G-2,6,3,G-2,-6,3,R,6,I":NEXT
420 PRINT"C9,M255,214,I,G-2,-46,-46
430 PRINT"G-2,46,-46,G-2,-45,15,G-2,45,15,C1
440 PRINT"M255,214,I,G-2,-44,-44,G-2,44,-44
450 PRINT"G-2,-43,13,G-2,43,13,C9,M225,213,I
460 PRINT"G-2,-11,-4,G-2,26,-4,G-2,-13,2
470 PRINT"G-2,23,2,M285,213,I,G-2,11,-4
480 PRINT"G-2,-26,-4,G-2,13,2,G-2,-23,2
490 PRINT"C7,M213,213,I,D297,213,C2
500 PRINT"M230,213,I,D240,5,5,M280,213,I,D240,5,5
510 PRINT"C3,M230,213,I,D240,3,3,M280,213,I,D240,3,3
520 PRINT"C4,M255,190,I,G-2,-3,25,G-2,3,38
530 PRINT"C6,M255,180,I,G-2,-9,3,G-2,-9,-3,G-2,9,3,G-2,9,-3
540 PRINT"C3,L3,M255,110,I,D255,50,50":DISK!"IO ,01":END
```

6. OKTA

```
10 DISK!"GO C000":DISK!"IO ,04"
20 PRINTCHR$(18)CHR$(2)"M150,,I"
30 DIMX(9),Y(9):X(1)=150
40 Y(2)=66: Y(3)=189: X(4)=X(1)
50 Y(4)=255: X(5)=355: Y(5)=Y(4)
60 X(6)=510: Y(6)=189:X(7)=X(6)
65 Y(7)=66:X(8)=X(5): X(4)=X(1)
70 Y(7)=66:X(8)=X(5): X(4)=X(1)
80 X(9)=X(1)
90 FOR O=1TO105:FORD=1TO8
100 X=X(D+1): Y=Y(D+1)
110 PRINT"C"DAND5",D"X","Y
120 X(D)=X(D)+INT((X(D+1)-X(D))/22+.5)
130 Y(D)=Y(D)+INT((Y(D+1)-Y(D))/22+2)
140 NEXT:X(9)=X(1):Y(9)=Y(1):NEXT
150 PRINTCHR$(2):FORK=1TO10
160 PRINTCHR$(1)"B"K:NEXT
170 DISK!"IO ,01
```

7. CON

```
10 DISK!"GO C000":DISK!"IO ,04
20 PRINTCHR$(18)"M255,127,I,W1,B0,"CHR$(2)
30 FORI=0TO249
40 X=INT(255%(1+.45%SIN(I%6.2832/250)))
50 Y=INT(127%(1-.6%COS(I%6.2832/250)))
60 PRINT"H,D"X","Y",C"I:NEXT:PRINTCHR$(2)CHR$(1)
70 FORI=1TO60
80 X=INT(RND(1)%511):Y=INT(RND(1)%255)
90 PRINT"M"X","Y",I":PRINTCHR$(1)CHR$(1):NEXT
100 DISK!"IO ,01":END
```

8. VASA

```
10 DISK!"GO C000":DISK!"IO ,04":PRINTCHR$(18)
20 PRINT"C9,M255,153,I,D15,100,100,H,C1,D15,80,80,H
30 PRINT"C9,D15,60,60,H,C1,D15,45,45,H
40 PRINT"C2,D15,35,35,C0,H,D15,20,20,C8,H,D15,12,12
50 Y=-47:CT=3:CR=9:X1=50:Y1=45:OX=4:OY=8:OC=8:X=256
60 Y=Y+Y1+1:Y1=Y1-OY:X2=X1-OX
70 PRINT"M"X","Y+1",I,C6,J255,,H":Y=Y+1
80 FORT=0TO12:IFX>511THEN110
90 PRINT"M"X","Y+1",I,C"CR",G-1,"X1","Y1",C"CT
100 PRINT"G-2,"(T%(X1-X2))","Y1
110 X=X+X1:CR=CR+OC:OC=OC%(-1):PRINT"C"CR:NEXT
120 X1=X2:OC=OC%(-1):PRINT"C"CR:X=255:OY=OY-1
130 IFY>100THENOY=OY+2
140 IFY<138THEN60
150 OC=8:Y=-47:CT=3:CR=9:X1=50:Y1=45:OX=4:OY=8:X=256
160 Y=Y+1+Y1:Y1=Y1-OY:X2=X1-OX:PRINT"M"X","Y+1",I,C6,J-255,,H
170 Y=Y+1
180 FORT=0TO12:IFX<0THEN210
190 PRINT"M"X","Y+1",I,C"CR",G-1,"(-X1)","Y1",C"CT
200 PRINT"G-2,"(-T%(X1-X2))","Y1
210 X=X-X1:CR=CR+OC:OC=OC%(-1):PRINT"C"CR:NEXT
220 X1=X2:OC=OC%(-1):PRINT"C"CR:X=255:OY=OY-1
230 IFY>100THENOY=OY+2
240 IFY<138THEN160
250 Y=255:CT=9:CR=3:X1=50:Y1=25:OC=8:OX=4:OY=13:X=256:GOTO270
260 Y=Y-Y1-1:Y1=Y1-OY
270 X2=X1-OX:PRINT"M"X","(Y)",I,C6,J255,,H
280 FORT=0TO12:IFX>511THEN310
290 PRINT"M"X","Y-1",I,C"CR",G-1,"X1","(-Y1)",C"CT
300 K=(T%(X1-X2)):PRINT"G-2,"K%SGN(K)","(-Y1)
310 X=X+X1:CR=CR+OC:OC=OC%(-1):PRINT"C"CR:NEXT
320 X1=X2:OC=OC%(-1):PRINT"C"CR:X=255:OY=OY-3
330 IFY<200THENOY=OY+2:OX=OX-1
340 IFY>200THEN260
350 Y=255:CT=9:CR=3:X1=50:Y1=25:O=8:OX=4:OY=13:X=256:GOTO370
360 Y=Y-Y1-1:Y1=Y1-OY
370 X2=X1-OX:PRINT"M"X","(Y)",I,C6,J-255,,H
380 FORT=0TO12:IFX<0THEN410
390 PRINT"M"(-X)","Y-1",I,C"CR",G-1,"(-X1)","(-Y1)",C"CT
400 K=T%(X1-X2):PRINT"G-2,"(-K%SGN(K))","(-Y1)
410 X=X-X1:CR=CR+OC:OC=OC%(-1):PRINT"C"CR:NEXT
420 X1=X2:OC=OC%(-1):PRINT"C"CR:X=255:OY=OY-3
430 IFY<200THENOY=OY+2:OX=OX-1
440 IFY>200THEN360
450 PRINT"M256,153,I":A=1
460 PRINT"G-1,"A","A",R"A+1",,I,C2,G-1,"A","A",R"A+1",,I,C1
470 IFA<15THENA=A+1:GOTO460
480 PRINT"M254,153,I":A=1
490 PRINT"G-1,"(-A)","A",R"(-A-1)",,I,C2,G-1,"(-A)","A"
500 PRINT"R"(-A-1)",,I,C1
510 IFA<15THENA=A+1:GOTO490
520 DISK!"IO ,01
```

9. COVER

```
10 DISK!"GO C000":DISK!"IO ,04":PRINTCHR$(18)"M,,I"
20 PRINT"C0,M20,5,I,G-1,490,250,C2,M20,30,I,G-1,490,200
30 PRINT"M40,40,I,G-1,465,182,C3,G-1,460,180
40 PRINT"C12,M39,39,I,G1,462,181
50 PRINT"M30,50,I,C7,G-1,24,162,C0,G-1,20,160
60 PRINT"C11,M48,52,I,Q2,S3,2,P ELEKTOR
70 PRINT"M29,49,I,C8,G1,21,161,S1,1
80 PRINT"C7,M45,45,I,O255,4,4
90 PRINT"M45,215,I,O255,4,4
100 PRINT"O255,3,1,C9
110 FORB=205TO55STEP-5:PRINT"M50,"B",I,J5,,":NEXT
120 REM IC 4, 1, 3
130 B=20:PRINT"M70,45,I":GOSUB970:PRINT"R5,25,P4
140 PRINT"M110,45,I":GOSUB970:PRINT"R5,25,P6
150 PRINT"M70,110,I":GOSUB970:PRINT"R5,25,P1
160 B=16:PRINT"M70,175,I":GOSUB970:PRINT"R5,23,P3
170 REM IC 2, 13, 11 ...
180 Y=175
190 PRINT"H,M110,"Y",I":GOSUB970:PRINT"R5,23,P2
200 PRINT"H,M150,"Y",I":GOSUB970:PRINT"R5,23,P13
210 PRINT"H,M190,"Y",I":GOSUB970:PRINT"R5,23,P11
220 PRINT"M230,45,I":GOSUB970:PRINT"R5,23,P8
230 PRINT"M270,45,I":GOSUB970:PRINT"R5,23,P9
240 PRINT"M310,45,I":GOSUB970:PRINT"R5,23,P16
250 PRINT"M350,45,I":GOSUB970:PRINT"R5,23,P30
260 PRINT"M390,45,I":GOSUB970:PRINT"R5,23,P29
270 B=14:PRINT"M430,45,I":GOSUB970:PRINT"R5,22,P28
280 REM RAM
290 B=16:Y=175
300 PRINT"M350,"Y",I":GOSUB970:PRINT"R5,23,P24
310 PRINT"M390,"Y",I":GOSUB970:PRINT"R5,23,P23
320 PRINT"M430,"Y",I":GOSUB970:PRINT"R5,23,P22
330 PRINT"M470,"Y",I":GOSUB970:PRINT"R5,23,P21
340 Y=123
350 PRINT"M470,"Y",I":GOSUB970:PRINT"R5,23,P17
360 PRINT"M430,"Y",I":GOSUB970:PRINT"R5,23,P18
370 PRINT"M390,"Y",I":GOSUB970:PRINT"R5,23,P19
380 PRINT"M350,"Y",I":GOSUB970:PRINT"R5,23,P20
390 REM IC12, 27 ,10, 7
400 B=14
410 PRINT"M230,180,I":GOSUB970:PRINT"R5,22,P12
420 PRINT"M270,180,I":GOSUB970:PRINT"R5,22,P27
430 B=16:PRINT"M270,123,I":GOSUB970:PRINT"R5,22,P10
440 B=20:PRINT"M230,95,I":GOSUB970:PRINT"R5,22,P7
450 PRINT"M316,124,I,C7,G1,10,83,M318,120,I,C14
470 FORK=193TO80STEP-12
480 PRINT"H,R,5,I,G1,2,1,R5,,G1,2,1":NEXT
490 PRINT"C2,Q1,R-5,3,PK1
500 PRINT"R-18,-7,P1
510 PRINT"R12,,P2
520 PRINT"R-31,-40,P17
530 PRINT"R11,,P18
540 PRINT"R-36,-40,P33
550 PRINT"R11,,P34
560 REM GDP
570 PRINT"M170,45,I,C7,R2,,G-1,40,100,C12,R-2,1
580 FORK=1TO20:PRINT"G1,2,2,R44,,G1,2,2,R-44,5":NEXT
590 PRINT"C1,S2,1,M195,55,I,Q2,P GDP
600 PRINT"S1,1,PIC5
610 PRINT"C3,M193,45,I,O15,3,1
620 REM IC 26,15,14...
630 PRINT"Q1
640 B=14:PRINT"M430,100,I":GOSUB1030:PRINT"R-20,-4,P26
650 B=16:PRINT"M350,100,I":GOSUB1030:PRINT"R-20,-4,P15
660 B=16:PRINT"M270,100,I":GOSUB1030:PRINT"R-20,-4,P14
670 B=14:PRINT"M160,152,I":GOSUB1030:PRINT"R-20,-4,P25
680 PRINT"C7,H,O60,4,2,M160,160,I,C3,O195,4,2
690 REM DIL
700 PRINT"M109,159,I,C6,G-1,30,-40,H,C0,R8,1
710 FOR K=1TO8
720 PRINT"R,-5,G-1,7,3,R9,,C7,G-1,7,3,R-9,,C0":NEXT
730 PRINT"M73,167,I,C9":GOSUB1080:REM C5
740 PRINT"M73,104,I":GOSUB1080:REM C2
750 PRINT"M112,110,I":GOSUB1080:REMC4
760 PRINT"M273,171,I":GOSUB1080:REMC7
770 PRINT"M433,86,I":GOSUB1080:REMC9
780 PRINT"M151,168,I
790 FORK=1TO8:PRINT"R-1,-4,I,O,3,1":NEXT
800 PRINT"C7,R-1,-3,J,39,M151,93,I,C9
810 FORK=1TO8:PRINT"R-1,-4,I,O,3,1":NEXT
820 PRINT"C7,R-1,-3,J,39,M243,88,I,C9
830 FORK=1TO8:PRINT"H,R8,,I,O,3,1,R-1,,C7,J,4,C9":NEXT
840 PRINT"M350,91,I,C9":FORK=1TO8:PRINT"H,R8,,I,O,3,1":NEXT
850 PRINT"M350,91,I,C7,J64,,I,R30,2,I,C9,O,4,1,R10,1,I,O,4,1,R10,1,I
860 PRINT"O255,4,1,M230,165,I":GOSUB1100
870 PRINT"M465,70,I":GOSUB1100
880 PRINT"M478,82,I":GOSUB1100
890 PRINT"M490,82,I,O,4,1,M490,75,I,O,4,1,M490,68,I,O,4,1
900 PRINT"C14,M493,115,I,O,3,1,M493,107,I,O,3,1,M493,99,I,O,3,1
910 PRINT"M493,91,I,O,3,1
920 REM QUARTZ
930 PRINT"C12,M482,61,I
940 PRINT"O,3,1,R-1,,J,-5,R-6,,J12,,R3,-2,G-1,-18,-2
950 PRINT"R-3,-4,J-12,,R6,,J,-5,R-1,,I,O,3,1
960 DISK!"IO ,01":END
970 PRINT"C7,H,R2,,G-1,20,"5%(B/2)
990 PRINT"H,R-1,1,C4":FORK=1TOB/2
1000 PRINT"G1,2,2,R24,,G1,2,2,R-24,5":NEXT
1010 PRINT"C1,H,R16,3,P IC
1020 PRINT"C3,H,R12,,I,O15,3,1,C1":RETURN
1030 PRINT"C7,H,R,2,G-1,"4%B",12
1040 PRINT"H,R4,-1,C4":FORK=1TOB/2
1050 PRINT"G1,2,2,R,15,G1,2,2,R8,-15":NEXT
1060 PRINT"C1,H,R9,4,P IC
1070 PRINT"C3,H,R"4%B",8,I,O60,4,2,C1":RETURN
1080 PRINT"O255,3,1,R3,1,J3,,R,3,J,-6,R3,,J,6,R,-3,J3,,R3,,I,O255,3,1
1090 RETURN
1100 PRINT"O255,3,1,R-1,-1,J,-3,R-3,,J6,,R,-3,J-6,,R3,,J,-3,R,-2,I,
1110 PRINT"O255,3,1":RETURN
```

10. OCTANT

```
10 DISK!"GO C000":DISK!"IO ,04":PRINTCHR$(18)"B6,C7,W1"
20 FORI=0TO100:GOSUB30:NEXT:DISK!"IO ,01":END
30 T=6.28%I/201:X=INT(255%(1-.8%COS(T)))
40 Y=INT(127%(1-.8%SIN(T)))
50 C=C+11:PRINT"M"X","Y",I,C"C",O255,40,40":RETURN
```