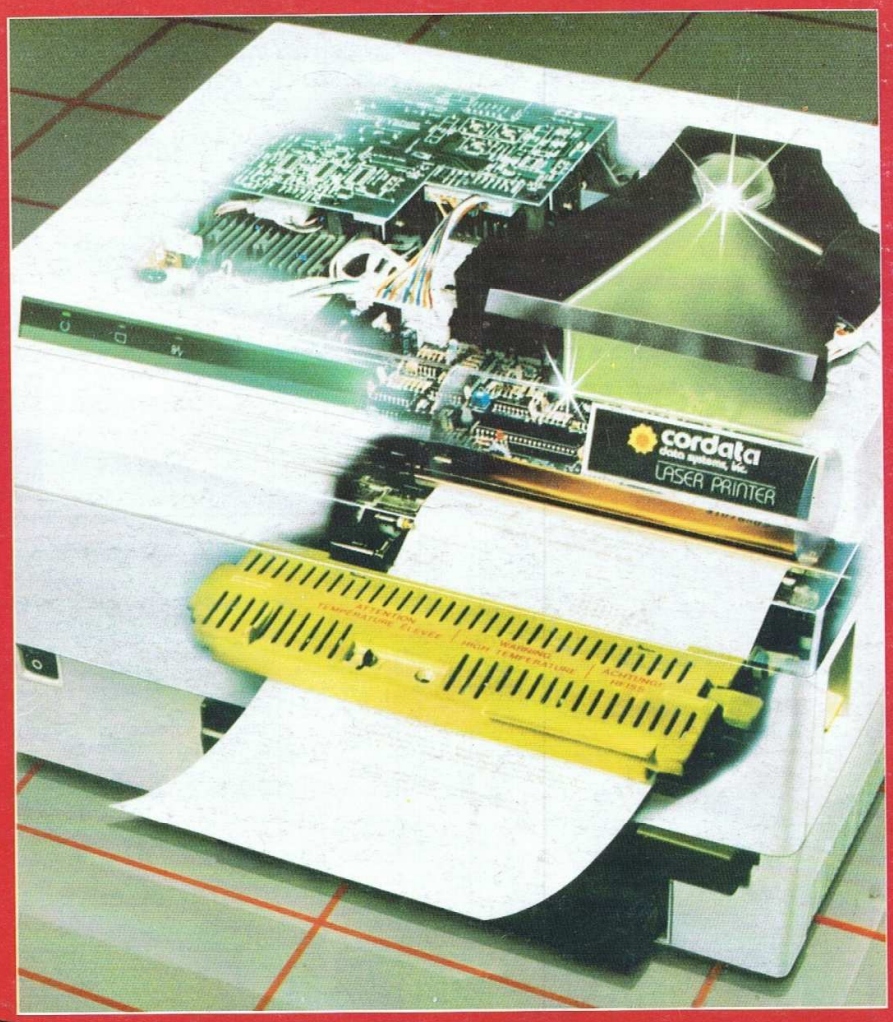
elektuur extra

f 18,75/Bfrs. 370

COMPUTING



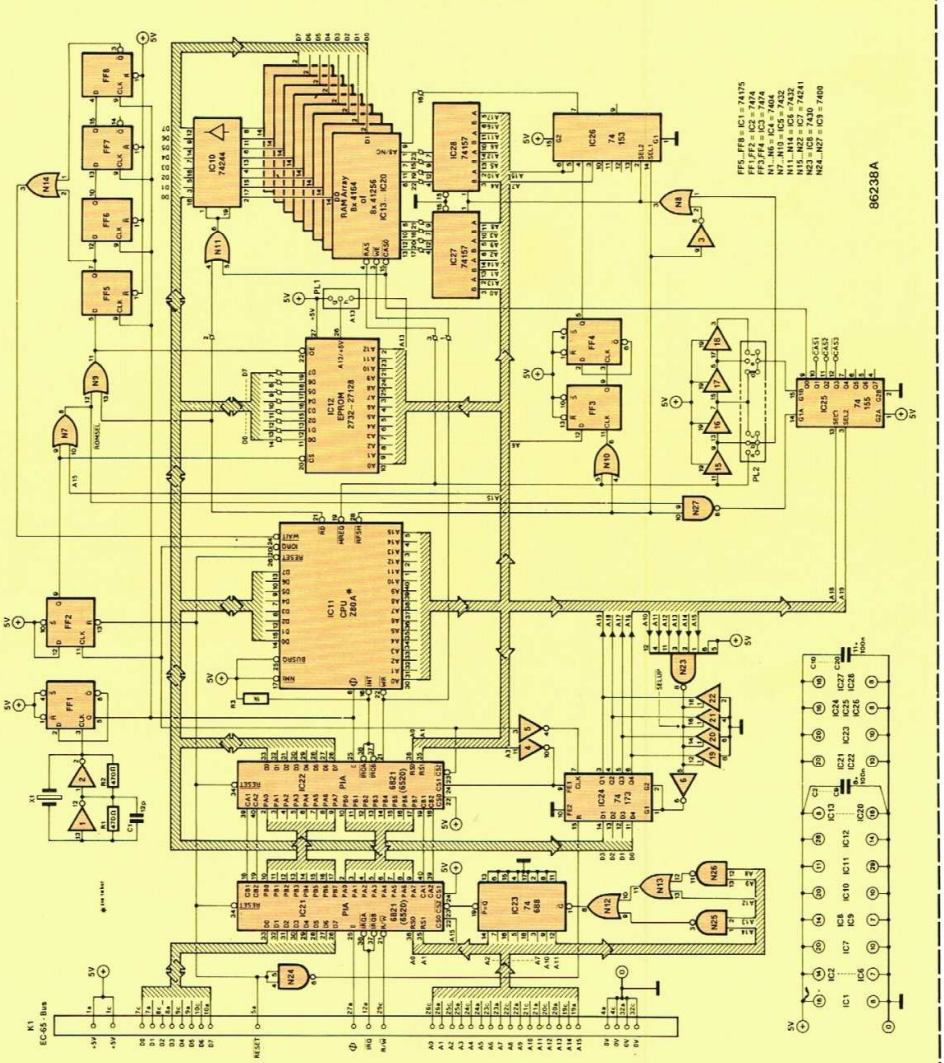
5e speciale computer-uitgave

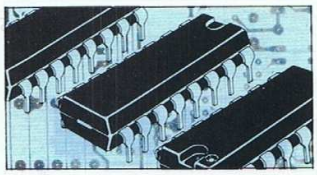
laser-printers • 8 MHz Z80-kaart

256 KB SRAM-kaart • UCSD-p-systeem

muis-interfaces • CP/M op de Octopus

EC-65 (K): Z80-kaart





CP/M op de Octopus 65 (deel 1)

een Z80-coprocessorkaart voor een 6502-computer

Wat kunt u als bezitter van een Octopus 65 met een tweede processorkaart doen? Deze vraag hebben we in de vorige special al opgehelderd: U kunt er CP/M-programma's mee laten lopen, maar met een beetje fantasie zult u er snel achter komen dat dit nog lang niet alles is. Zo kan men bij gekompliceerde rekenprogramma's beide CPU's gelijktijdig laten werken. Ook als men niet met CP/M werkt, hoeft de op de Z80-kaart aanwezige geheugenkapaciteit niet onbenut te blijven: deze kan als printer-spooler, als RAM-disk of als tijdelijk geheugen voor allerlei gegevens gebruikt worden.

Natuurlijk heeft men voor al deze toepassingen ook de bijbehorende software nodig, maar die zal men dan gedeeltelijk moeten schrijven. In deze Elektuur Computing vindt u wel alle programma's voor het werken met CP/M. Ook laten we in deze uitgave zien hoe de Z80-computer kan worden gebruikt voor tussentijdse data-opslag voor onze grafische kleurenkaart (er zijn dan minder repeterende berekeningen nodig, wat een wezenlijk hogere snelheid bij het weergeven van grafische beelden tot gevolg heeft).
Het kreëren van een RAM-disk is nogal ingewikkeld. U zult daarvoor het OHIO-DOS moeten aanpassen, omdat dit DOS geen enkele funktie kent voor het werken met een RAM-disk. We beperken ons hier dus tot CP/M.

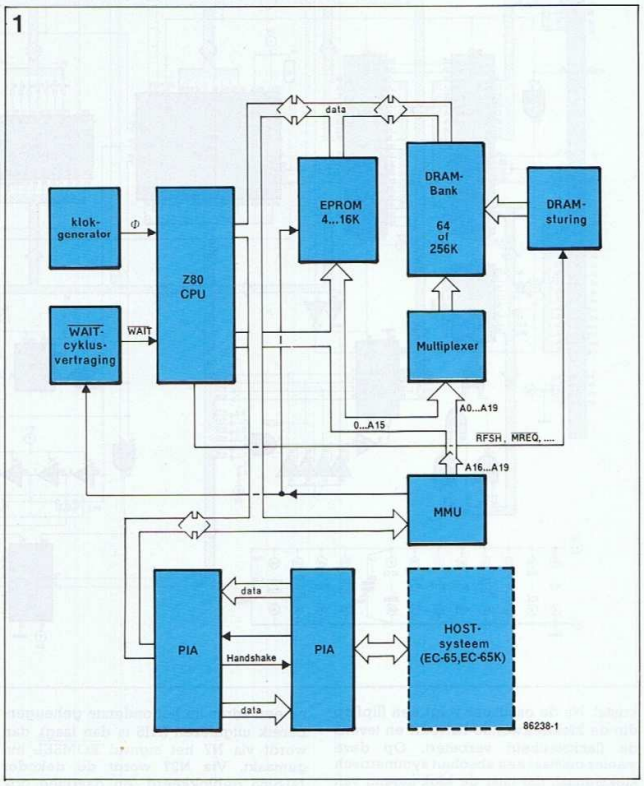
Het koncept

Laten we nu maar eens gaan kijken naar de hardware. De Z80-CPU-print is in de eerste plaats met het oog op de volgende punten ontwikkeld:

— De hele schakeling moest op een eurokaart passen.
— Voor een snelle data-verwerking moest de mogelijkheid bestaan om een snelle CPU te kunnen toepassen.
— Om dezelfde reden moest een grote geheugenkapaciteit ter beschikking staan, die men zonodig op een tweede print uitbreiden kan.
— Uit plaatsgebrek en prijsoverwegingen moest de kaart zo kompleet mogelijk zijn. Er komen namelijk geen verdere uitbreidingen voor het Z80-systeem.

Wat wij hiervan verwezenlijkt hebben, kunt u zien in het kader met technische gegevens aan het einde van dit artikel. Daarin zijn de belangrijkste gegevens opgesomd. U kunt zien dat we niet alleen de schakeling, die oorspronkelijk over drie eurokaarten verdeeld was, sterk hebben gekomprimeerd, maar daarnaast ook nog voor een hoge snelheid gezorgd hebben. Hiermee kunt u de Octopus 65 uitbouwen tot een van de snelste CP/M-computers die momenteel bestaan. Dat geldt vooral wanneer u het geheugen later tot 512 of 1024 Kbyte wilt uitbreiden. Dit extra geheugen kan bijvoorbeeld als disk-buffer worden gebruikt, waardoor de handelingen met de drives tot een absoluut minimum worden teruggebracht. Helaas vervallen dan wel de vele koffiepauzes die nu nog ontstaan bij het laden van de disk-files in RAM.

1

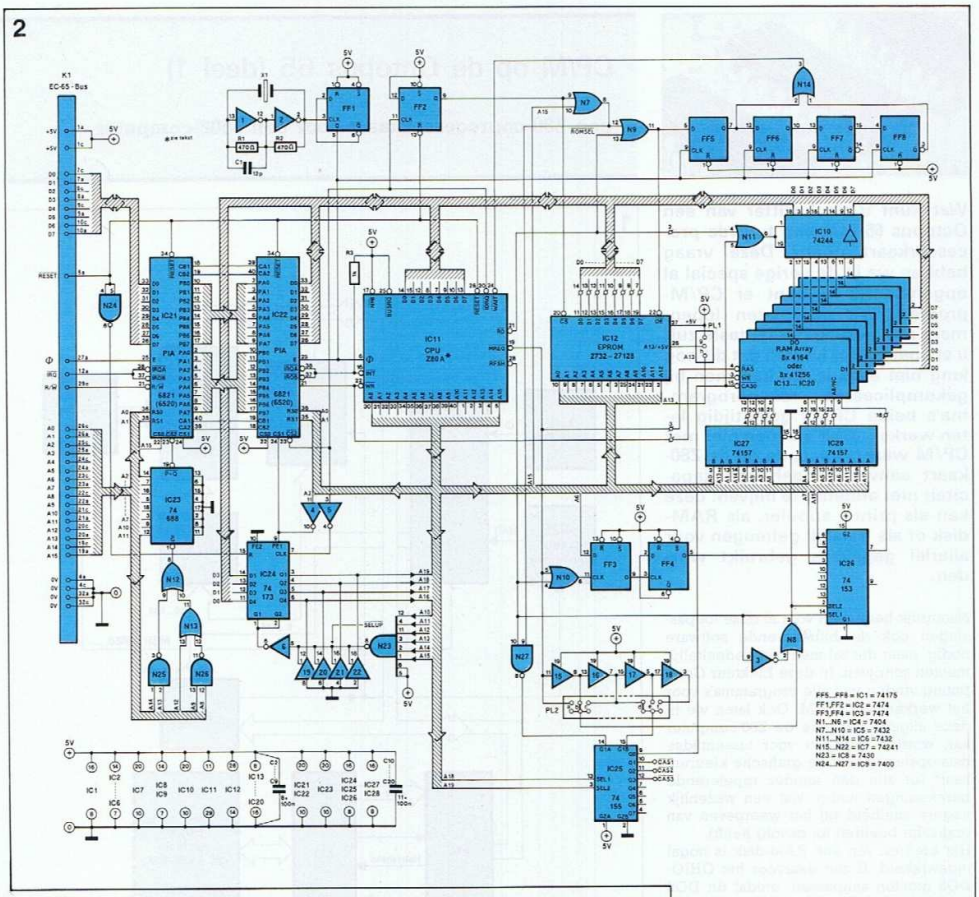


Figuur 1. Het blokschema van de Z80-print. De kommunikatie tussen Octopus en Z80-print loopt via twee PIA's.

De schakeling

Laten we eens een blik op het blokschema (figuur 1) werpen. Daarin zijn voor de overzichtelijkheid alleen de belangrijkste verbindingen getekend. Bovenin bevindt zich de komplete Z80-computer met CPU, RAM, ROM en I/O. De met twee PIA's opgebouwde FIFO verzorgt de verbinding tussen het Z80-systeem de Octopus.

De moeite van het vermelden waard is nog het blokje MMU in het rechter gedeelte van het blokschema. Daar worden de adressen A16...A19 gegenereerd, die nodig zijn om het systeem uit te kunnen bouwen tot één Megabyte. Verder schakelt de MMU de EPROM na het kopiëren in het RAM-bereik af, zodat altijd komplete blokken van 64 Kbyte RAM geadresseerd kunnen worden.
Dan komen we nu bij het eigenlijke schema (figuur 2). Linksboven bevindt zich de klok van het systeem. Deze is opgebouwd uit de poorten N1 en N2, alsmede twee weerstanden, een kondensator en een



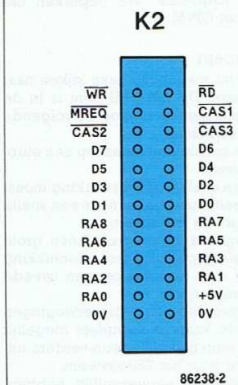
kristal. Na de oscillator volgt een flipflop die de klokfrekwentie halveert en tevens de flanksteilheid verbetert. Op deze manier ontstaat een absoluut symmetrisch kloksignaal, dat naar de klok-ingang van de CPU wordt gevoerd.

De beide ingangen $\overline{\text{NMI}}$ en $\overline{\text{BUSRQ}}$ van de CPU worden niet gebruikt en liggen dan ook kontinu aan +5 V. Het RESET-signaal wordt direkt van de Octopus-bus afgenomen. De flipflops FF5...FF8 zorgen ervoor dat bij iedere EPROM-adressering drie wachtcycli worden tussengevoegd. Daardoor kan men zelfs in een 8-MHz systeem met goedkope 350-ns- of 400-ns-EPROM's werken. Deze vertraging van het EPROM-geheugenbereik heeft geen nadelige invloed op de CPU-snelheid. Tenslotte wordt de EPROM immers na een RESET gekopieerd en dan uitgeschakeld.

RAM — ROM

Flipflop FF2 vormt het eerste deel van de MMU (Memory Managing Unit = geheugenbesturingseenheid). Na een RESET (dus bijv. na het inschakelen) wordt deze altijd op nul gezet, zodat de $\overline{\text{CS}}$-ingang van de EPROM geaktiveerd wordt. Wordt nu een adres uit het onderste geheugenbereik uitgelezen (A15 is dan laag), dan wordt via N7 het signaal $\overline{\text{ROMSEL}}$ nul gemaakt. Via N27 wordt de dekoder 74LS155 geblokkeerd, en daarmee ook het RAM-bereik. Wordt nu $\overline{\text{RD}}$ geaktiveerd, dan verandert het uitgangsnivo van N9 en wordt de EPROM geselekteerd.

Wil de CPU zich toegang verschaffen tot de I/O, dan wordt het $\overline{\text{IORQ}}$-signaal even nul gemaakt, zodat FF2 getriggerd wordt. De uitgang van deze flipflop wordt "hoog" tot de volgende RESET. Daarmee is iedere toegang tot de EPROM onmogelijk gemaakt. Tegelijkertijd wordt de EPROM via de $\overline{\text{CS}}$-ingang in de stroomsparende standby-mode geschakeld. Op deze manier wordt in het onderste adresbereik in plaats van het ROM- altijd het RAM-geheugen geaktiveerd. In het bovenste geheugenbereik is dat altijd al het geval. Op deze manier kan men met behulp van een I/O-signaal na een RESET de EPROM uitschakelen en dan heeft men een computer met 64 Kbyte RAM. Daarmee is dan tegelijk de opzet rond de EPROM verklaard. Rest ons nog te vertellen dat men met PL1 kan kiezen tussen de EPROM-typen 2732,



Figuur 2. De totale schakeling met CPU, ROM, RAM, MMU en I/O past op een enkele eurokaart.

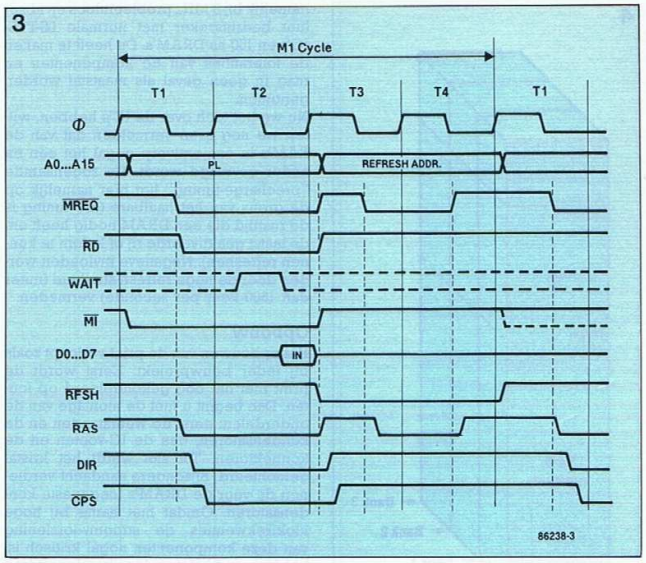
2764 en 27128 (zie tabel 1). De 2732 wordt zodanig in de (voor hem te grote) voet geplaatst dat de aansluitpunten 1, 2, 27 en 28 vrij blijven.

DRAM

Laten we nu eens kijken naar de besturing voor de dynamische RAM's. Naar keuze kunnen 64-K- of 256-K-typen gebruikt worden. Helaas vraagt het tweede type wat meer vermogen van de refreshlogika, maar hierover later meer.
Het $\overline{RAS}$-signaal voor het geheugen wordt direkt van $\overline{MREQ}$ afgeleid. Het signaleert de eerste (de onderste) helft van de geheugenadressen. Even later worden de multiplexers IC26...IC28 omgeschakeld, waarna het tweede gedeelte van het adres wordt aangeboden aan de RAM's. De vertragingstijd kan men instellen met PL2. Daarna wordt ingang G1B van de 74LS155 doorgeschakeld en een van de signalen $\overline{CAS0}$ tot $\overline{CAS3}$ geaktiveerd. Dit signaal deelt de geselekteerde geheugenbank mee dat nu de tweede adreshelft stabiel is. De signalen op de ingangen SEL1 en SEL2 van de 74LS155 bepalen welke van de geheugenbanken 0 tot 3 is gekozen. Deze signalen zijn afkomstig van de adreslijnen A18 en A19. Elke geheugenbank kan maximaal 256 Kbyte groot zijn. Op deze manier krijgen we een adresbereik van maximaal 1 Mbyte. In verband met de beperkte ruimte kunnen op de print slechts acht geheugen-IC's ondergebracht worden, wat neerkomt op een kapaciteit van 64 of 256 Kbyte. Het is overigens de moeite waard om meteen 256-K-RAM's toe te passen. Ten eerste zijn deze zo langzamerhand tegen beschaafde prijzen te koop en ten tweede is deze investering direkt vanaf het begin lonend (niet alleen bij het gebruik van CP/M 3.0). Hierover meer in het software-gedeelte van dit artikel.

Refresh

Het hele refresh-gebeuren voor de DRAM's is tamelijk gekompliceerd, hoewel de CPU daarbij stevig meehelpt, zoals al eerder is opgemerkt. Na elke opcodefetch, dus altijd wanneer de CPU met de dekodering van een pas geladen kommando bezig is, stuurt de Z80 een refreshadres via de onderste helft van de adresbus. Daarbij worden $\overline{RFSH}$ en $\overline{MREQ}$ geaktiveerd. Met $\overline{MREQ}$ nemen de RAM's het adres over en voeren dan de refresh uit. In de CPU wordt het laatste refresh-adres in een register opgeslagen en de volgende keer automatisch verhoogd. Helaas is het door de CPU geleverde adres maar zeven bits breed. Het achtste bit is altijd nul. Bij de meeste 64-K-chips is dat geen probleem, maar een 256-K-DRAM heeft beslist een acht-bits refresh nodig. Daarom moet de interne CPU-teller extern met een bit uitgebreid worden. Dat gebeurt met FF3 en FF4. FF3 neemt aan het eind van elke $\overline{RFSH}$ de waarde van het zevende bit over. De flipflop vormt zo een tussengeheugen voor deze stand van de refresh-teller. FF4 is als deler geschakeld. Op zijn uitgang staat het gewenste achtste bit ter beschikking. De dekoder zet dit bit op ingang A7 van de RAM's zodra $\overline{RFSH}$ aktief is en de onderste helft van de adresbus aangeboden wordt.



Figuur 3. Het tijdvolgordediagram voor de Z80-print.

De MMU

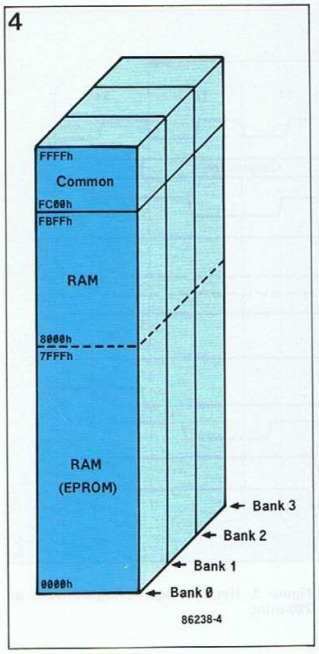
We hebben hierboven al gezien dat de CPU in onze schakeling maximaal 1 Mbyte aan geheugen kan adresseren. Daarvoor zijn 20 adreslijnen nodig. De Z80 zelf heeft er maar 16. De resterende vier worden op de kaart gemaakt, waardoor het mogelijk is om in totaal 16 geheugenbanken van 64 Kbyte per stuk (of beter: 63 Kbyte, zie onder) te adresseren. Laten we nu even kijken waar de toegevoegde adreslijnen vandaan komen.
Links onder in het schema (figuur 2) bevindt zich een 74LS173. Dat is een viervoudige D-flipflop met tri-state-uitgangen. Op de ingangen zijn de datalijnen D0...D3 aangesloten en op de uitgangen de nieuwe adreslijnen A16...A19. De flipflops nemen de data aan de ingangen over op het moment dat ze een positieve flank aan de klokingang ontvangen en de beide ingangen FE1 en FE2 bovendien nul zijn. FE2 is dat altijd, FE1 alleen als A3 "1" is. Men kan dus data wegschrijven in deze flipflop door deze data bijvoorbeeld naar I/O-kanaal 8 te sturen. Op deze manier kan men (software-matig) de vier hoogste adreslijnen instellen.
Om te voorkomen dat de computer zichzelf "ophangt" als de geheugenbank plotseling omgeschakeld wordt, is er een zogenaamd COMMON-bereik aanwezig. Dit is een geheugenbereik dat niet door de hoogste adreslijnen omgeschakeld kan worden. Hier is dit bereik 1 Kbyte groot en het bevindt zich op $FC00...$FFFF. Het COMMON-bereik kan gebruikt worden om data van de ene 64-K-geheugenbank naar de andere te transporteren. Dit COMMON-bereik is gerealiseerd met behulp van N23, N6 en N19...N22. Wordt een van de adressen uit het COMMON-bereik op de adresbus aangeboden, dan zijn de adreslijnen A10...A15 logisch één. N23 wordt omgeschakeld en via N6 worden de uitgangen van de adresflipflops in de hoogohmige toestand geschakeld. Tegelijkertijd worden N19...N22 weer teruggeschakeld tot hun normale driver-funktie. Hun ingangen liggen aan massa, zodat ook de adreslijnen A16...A19 "0" worden, onafhankelijk van de toestand van de flip-flops in de 74LS173.
Figuur 4 verduidelijkt de betekenis van het COMMON-bereik.

PIA

Tenslotte blijft alleen nog het linker gedeelte van de schakeling over, dat de interface tussen de Octopus 65 en Z80 bevat. Veel valt daarover niet te vertellen. Het gaat hier om twee PIA's waarvan telkens één poort als ingang en de andere als uitgang geprogrammeerd wordt. Het schema toont de (geprogrammeerde) datarichting. Bovendien worden de handshake-lijnen gebruikt om de CPU's te laten weten dat aan de ingangen nieuwe data beschikbaar zijn en wanneer de data aan de uitgang overgenomen zijn. De als adresdekoder geschakelde poort N4 zorgt er voor dat de PIA alleen als uitgangskanaal kan worden geschakeld als de 74LS173 niet geselekteerd is. De PIA die het verkeer naar de Z80 verzorgt, bezet de I/O-kanalen 0...3. De PIA naar de 6502-kant gebruikt de adressen $E300...$E303. Hiervoor worden de poorten N25, N26, N12 en N13, alsmede de komparator 74LS688 als adresdekoder gebruikt.

4, 6 of 8 MHz?

Deze vraag moet men stellen voordat met het opbouwen van de print wordt begonnen. Standaard is een Z80 met een 4-MHz-klok. 6 MHz halen momenteel maar enkele machines en met 8 Mhz zit u zo onge-



Figuur 4. Zo ziet de geheugen-indeling er uit wanneer de print is uitgerust met 256 Kbyte RAM. In het bovenste gedeelte bevindt zich het COMMON-geheugen. In dit moet zich de CPU bevinden wanneer omgeschakeld wordt naar een andere geheugenbank. De EPROM wordt in alle geheugenbanken gedupliceerd. Direkt na een RESET wordt de EPROM helemaal uitgeschakeld. De CPU werkt dan nog alleen maar met RAM.

veer het maximum van wat u op dit moment met een Z80 kunt bereiken. We zullen een voorzichtige vergelijking geven: MBASIC onder CP/M is op een Z80-CPU met 8 MHz bij rekenkundige routines (sin, cos. log, exp) meer dan twee keer zo snel als de Microsoft-BASIC van de Octopus met OSI-DOS en een 6502-klok van 2MHz!

Tabel twee laat zien welke komponenten men voor welke klokfrekwentie moet gebruiken. De CPU's en de DRAM's zijn zeer kritisch wat de timing betreft. Het is aan te bevelen om in ieder geval de aanbevelingen in de tabel op te volgen. Minder kritisch zijn de TTL-IC's, maar ook hier geldt: zeker is zeker.

Mocht u ondanks alles toch nog moeilijkheden bij zo'n hoge klokfrekwentie hebben, dan kan het ook liggen aan de flanksteilheid van het kloksignaal. In dit geval is het nuttig om FF1 (IC2) eenste vervangen. In geval van nood kan men het LS-exemplaar ook vervangen door een F-type.

Indien u uw Octopus 65 met een klok van 2 MHz uitrust, dient u IC21 te vervangen door een 68B21 of door een 6520A.

Dat het hierbij enkel om "aanbevelingen" gaat, bewijst de Z80-print die voor dit artikel werd opgebouwd. De print werkt namelijk bij 8 MHz probleemloos en absoluut bedrijfszeker met normale LS-TTL-IC's en 150-ns-DRAM's. Dit heeft te maken de toleranties van de komponenten en mag in geen geval als maatstaf worden genomen.

Nu we het toch over de IC's hebben, willen we nog even vermelden dat van de RAM's in ons ontwerp nogal het een en ander gevraagd wordt. De zogenaamde "precharge-timing" ligt hier namelijk op de grens van het haalbare (die timing is de rusttijd die een DRAM nodig heeft om de laatst geaktiveerde rij of kolom te kunnen refreshen). Negatieve invloeden worden door de hoge refresh-snelheid (meer dan 1500 keer per seconde) vermeden.

Opbouw

Het opbouwen van de print verloopt zoals bij ieder bouwprojekt: Eerst wordt de print met het oog gekontroleerd op fouten. Dan begint u met de montage van de onderdelen: eerst de weerstanden en de kondensatoren, dan de IC-voeten en de konnektoren. Tot slot wordt het kristal gemonteerd. Bijzondere aandacht verdienen de voor de DRAM's toegepaste kondensatoren. Omdat met name bij hoge klokfrekwenties de stroomvoorziening van deze komponenten nogal kritisch is, hebben we besloten deze kondensatoren direkt op de voedingsaansluitingen van de IC's te solderen. Vanuit technische oogpunt is dit de beste oplossing, die de bedrijfszekerheid van de print zeker ten goede komt. Natuurlijk vergt deze methode wat meer werk bij het opbouwen, maar we dachten dat dit geen al te grote problemen mag geven voor iemand die al eerder een totale Octopus in elkaar heeft gezet.

De voedingsaansluitingen zitten bij de DRAM's op de pennen 8 en 16. Wie het zekere voor het onzekere wil nemen, kan tantaal-elko's gebruiken, maar moet dan wel op de plus- en min-aansluiting letten: massa zit op pen 16 en pen 8 ligt aan +5V. Dus precies omgekeerd ten opzichte van TTL-IC's.

De kondensatoren worden direkt aan de onderzijde van de print gemonteerd. Het verdient aanbeveling om de aansluitdraden te isoleren met korte stukjes krimpkous of de kunststof isolatie van gewone koperdraad. Wie deze methoden niet professioneel genoeg vindt, kan ook voetjes aanschaffen waarin de kondensator al geïntegreerd zijn. In de industrie gebruikt men deze reeds lang.

Voor de eerste test moeten nog drie kortsluitstekertjes op de konnektoren PL1 en PL2 gezet worden. Over PL1 vindt u meer informatie in tabel 1. Op PL2 worden de jumpers voorlopig geplaatst zoals in het schema is aangegeven.

Omdat er maar weinig plaats is op onze print, en de printbanen nogal dun zijn, wordt aanbevolen om met een fijne soldeerbout met een niet te groot vermogen te werken; de dikke pook waarmee u gisteren nog de dakgoot heeft gerepareerd, kunt u beter in de kelder laten liggen. Bent u zover klaar met bouwen, werp dan eens een kritische blik op uw werkstuk. Als u nu een te grote soldeerklodder of een slecht soldeerpunt aantreft, kan u dat later een hoop ergernis besparen.

Test, test, test

Nu is het zo ver dat de print in de Octopus kan worden gestoken (nog zonder IC's). U kunt de computer daarna inschakelen en een of ander programma laten lopen. Gedraagt de computer zich niet normaal, dan heeft u wellicht toch ergens een kortsluiting over het hoofd gezien. Het beste is dan om zo snel mogelijk deze kortsluiting op te sporen.

Daarna wordt de spanning op de verschillende IC-voeten gekontroleerd. Is dit allemaal in orde, dan kan men IC21, IC23, IC6 en IC9 in hun voetjes plaatsen. Als u de print nu in de Octopus schuift, moet het mogelijk zijn om de PIA via de geheugenplaatsen $E300...$E303 te programmeren. Vervolgens worden IC2 en IC4 geplaatst. Wie in het bezit van een frekwentiemeter of een oscilloskoop is, kan testen of de klokgenerator goed werkt.

Hierna kan de print verder opgebouwd worden door het plaatsen van de resterende IC's. Na het inschakelen gaan we uitgang Q van FF2 bekijken (met behulp van een logic probe of een voltmeter, een skoop is hiervoor niet noodzakelijk). Na een RESET moet dit punt konstant "1" zijn. Dit is naar alle waarschijnlijkheid het teken dat het monitorprogramma van EPROM naar RAM gekopieerd is en daar braaf zijn rondjes draait.

Technische gegevens

* eurokaart voor de Octopus 65(of EC-65K)
* bezet vier adresplaatsen ($E300...$E303)
* Z80-CPU met 4, 6 of 8 MHz
* maximaal te adresseren geheugenbereik: 1 Mbyte
* DRAM op de print: 64 K of 256 K
* bruikbare EPROM's: 2732, 2764 of 27128 (4K, 8K of 16K)
* EPROM wordt na het opstarten uitgeschakeld
* WAIT-logika maakt het mogelijk om langzame EPROM's toe te passen, zelfs bij 8 MHz
* data-transport tussen de twee CPU's verloopt zeer snel door toepassing van twee PIA's als FIFO

Tabel 1. Met behulp van de instelling op konnektor PL1 wordt het gebruikte type EPROM aan de kaart kenbaar gemaakt.

Tabel 1.

EPROM	verbinding
2732	g
2764	geen
27128	h

Tabel 2. Hier ziet u welke onderdelen bij een bepaalde klokfrekwentie moeten worden gebruikt. Bij de TTL-IC's kan men eventueel nog volstaan met LS-typen, maar bij de overige komponenten moet men zich exakt houden aan het voorgeschreven type.

Tabel 2

komponent	4 MHz	6 MHz	8 MHz
IC5, IC6	74LS32	74LS32	74F32
IC11	Z80A	Z80B	Z80H
IC13...20 DRAM's	200 ns	150 ns	120 ns
IC22 of	6821 6520	68A21 6520A	68B21 6520A
IC25	74LS155	74LS155	74F155
IC26	74LS153	74F153	74F153
IC27, IC28	74LS157	74F157	74F157
X1, kristal	8 MHz	12 MHz	16 MHz

Figuur 5. Dit programma is nodig bij het testen van de schakeling. Het programma zorgt er voor dat de inhoud van de monitor-EPROM van de Octopus in het geheugen van de Z80-kaart geladen wordt. Daarna wordt de hele RAM-inhoud vergeleken met de EPROM.

Werkt dit ondanks een foutloze opbouw niet, dan moet u de instelling van PL2 veranderen, zodat de timing van de DRAM aangepast wordt.

Software-hulp

Is alles tot zo ver goed gegaan, dan kunt u met een laatste test zekerheid krijgen of alles helemaal in orde is. Daarvoor typt u het kleine BASIC-programma van figuur 5 in. Voordat u het programma start, is het verstandig om het eerst op een diskette op te slaan. Is dit gebeurd, dan kunt het programma laten draaien op de Octopus en afwachten wat vervolgens op het scherm verschijnt. Na een kleine melding zal het beeldscherm een heleboel punten laten zien: één voor elk byte dat naar de Z80 gestuurd wordt. Is dit niet het geval, dan is er iets niet in orde met de handshaking van de machine, of u heeft een fout gemaakt bij het intypen van het programma. Daarna verschijnt nog een keer een korte boodschap en probeert de Octopus de data weer terug te lezen. Dat wat de computer leest, wordt met het origineel vergeleken en als alles in orde is, wordt per byte een punt op het scherm geschreven. Komen de verstuurde en ontvangen data niet overeen, dan verschijnt een uitroepteken in plaats van een punt op het scherm. Tot slot wordt nog het aantal bytes aangegeven dat niet overeenkomt. Verschijnt het getal nul, dan kunt u verder gaan met het software-gedeelte in dit nummer.

5

```
1 PRINT CHR$(27);"1";"    TESTING EC-65 Z8Ø-BOARD": PRINT
2 GOTO 1ØØ: REM skip header
3 :
1Ø ***********************************************
2Ø *                                             *
3Ø *  Test Program FOR EC-65 Z8Ø Board at $E3ØØ  *
4Ø *                                             *
5Ø *       Written 1986  by  Frank Schmidt       *
6Ø *                                             *
7Ø ***********************************************
8Ø :
1ØØ REM === Initialisations ===
11Ø :
115 PRINT "... Initializing"
12Ø REM PIA Addresses
13Ø AD=58112: AC=AD+1: BD=AD+2: BC=AD+3
14Ø EPROM=6144Ø: REM start of EPROM
15Ø GOSUB 92ØØ: REM init PIA
16Ø POKE 23,79: REM Set terminal width for 8Ø columns
17Ø Z=Ø: REM counter for compare fails
18Ø :
19Ø :
2ØØ REM === Move EPROM data to Z8Ø ===
21Ø :
215 PRINT "... Moving Data"
22Ø A=1:   GOSUB 9ØØØ: REM Command: get 65Ø2-data
23Ø A=16:  GOSUB 9ØØØ: REM Store data from $1ØØØ on
24Ø A=Ø:   GOSUB 9ØØØ
25Ø A=INT(4Ø96/256): GOSUB 9ØØØ: REM Handle 4 KBytes
26Ø A=4Ø96-256*INT(4Ø96/256) :GOSUB 9ØØØ
27Ø FOR I = EPROM TO EPROM+4Ø95
275 :  PRINT ".";
28Ø :  A=PEEK(I): GOSUB 9ØØØ
29Ø NEXT: PRINT: PRINT
3ØØ :
31Ø :
5ØØ REM === Return data from Z8Ø and compare ===
51Ø :
515 PRINT "... Comparing Data ('!' means 'no match')
52Ø A=3:   GOSUB 9ØØØ: REM Command: hand data to 65Ø2
53Ø A=16:  GOSUB 9ØØØ: REM Hand data from $1ØØØ on
54Ø A=Ø:   GOSUB 9ØØØ
55Ø A=INT(4Ø96/256): GOSUB 9ØØØ: REM Handle 4 KBytes
56Ø A=4Ø96-256*INT(4Ø96/256) :GOSUB 9ØØØ
57Ø FOR I = EPROM TO EPROM+4Ø95
58Ø :  GOSUB 91ØØ: REM get byte from Z8Ø
```

Onderdelenlijst

Weerstanden:
R1,R2 = 470 Ω
R3 = 1 k

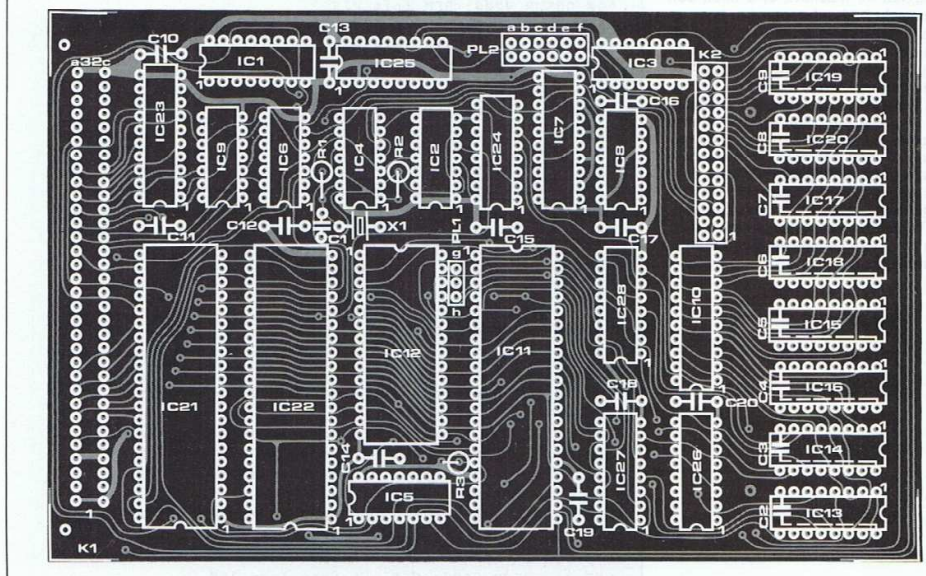
Kondensatoren:
C1 = 12 p
C2...C20 = 100 n

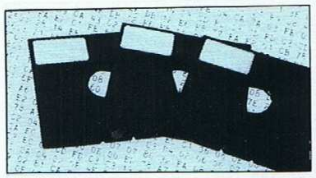
Halfgeleiders:
IC1 = 74LS175
IC2,IC3 = 74LS74 (+)
IC4 = 74LS04
IC5,IC6 = 74LS32 (*)
IC7 = 74LS241
IC8 = 74LS30
IC9 = 74LS00
IC10 = 74LS244
IC11 = Z80A (*)
IC12 = EPROM, 2732, 2764, 27128, max. 400 ns
IC13...IC20 = 4164 of 41256, 200 ns (*) (+)
IC21,IC22 = 6821 of 6520 (*) (+)
IC23 = 74LS688
IC24 = 74LS173
IC25 = 74LS155 (*)
IC26 = 74LS153 (*)
IC27,IC28 = 74LS157 (*)

Diversen:
X1 = kristal 8 MHz (*)
K1 = 64-polige konnektor, male, volgens DIN 41612
K2 = 26-pens konnektorstrip, twee rijen
PL1 = 3-pens konnektorstrip, enkele rij pennen
PL2 = 12-pens konnektorstrip, twee rijen
3 kortsluitstekert (jumpers)
print EPS 86238
(*) geldt voor 4-MHz-versie, zie ook tabel 2
(+) zie tekst

```
59Ø :  IF A=PEEK(I) THEN PRINT ".";: GOTO 61Ø
6ØØ :      PRINT "!";: Z=Z+1
61Ø NEXT
62Ø A=255: GOSUB 9ØØØ
63Ø PRINT
64Ø PRINT "... End of Test: ";Z;" compare fails
65Ø :
66Ø :
67Ø END
68Ø :
69Ø :
9ØØØ REM === Output (A) to Z8Ø Port ===
9Ø1Ø :
9Ø2Ø REM Wait until last byte accepted
9Ø3Ø IF PEEK(BC)<128 GOTO 9Ø3Ø
9Ø4Ø B=PEEK(BD): REM reset PBC flag
9Ø5Ø POKE BD,A : REM store data
9Ø6Ø RETURN
9Ø7Ø :
9Ø8Ø :
91ØØ REM === Read byte from Z8Ø Port to A ==:
911Ø :
912Ø REM Wait until PIA contains new data
913Ø IF PEEK(AC)<128 GOTO 913Ø
914Ø A=PEEK(AD): REM store PIA data in A
915Ø RETURN
916Ø :
917Ø :
92ØØ REM === Init PIA on Z8Ø Board ===
921Ø :
922Ø POKE AC,4Ø: POKE AD,Ø:   POKE AC,44
924Ø POKE BC,4Ø: POKE BD,255: POKE BC,44
925Ø POKE BD,255
926Ø FOR I=1 TO 2Ø
927Ø :  A=PEEK(AD): NEXT
928Ø RETURN
```

Figuur 6. Onderdelenopstelling voor de Z80-kaart.





CP/M op de Octopus 65 — deel 2

de systeem-software voor de coprocessor-kaart

Zonder software op disk kun je niet veel beginnen bij de Octopus 65. Vooral, wanneer hij moet kunnen kommuniceren met de in deze uitgave gepresenteerde Z80-kaart. Natuurlijk hebben we ook in dit geval gezorgd voor die noodzakelijke software. In dit artikel wordt daar het nodige over verteld.

Bij de meeste grotere computersystemen is bij het inschakelen van de computer nog geen operating-system in de machine aanwezig. Dat geldt vooral voor systemen die één of meer drives bezitten en geen vast ingebouwde hogere programmeertaal zoals BASIC bevatten. Na het inschakelen komt men in dat geval in een monitorprogramma terecht; van daaruit kunnen dan alle andere zaken vanaf schijf in het geheugen worden geladen: operating-system, hogere programmeertaal, etcetera. Zo werkte dat al bij onze oude junior computer en ook nog steeds bij de nieuwere systemen, zoals de Octopus 65, de EC-68 en de Big-Board II. In de EPROM op de Z80-kaart voor de Octopus bevindt zich ook een monitorprogramma, maar toch kan men hiermee niet werken zonder de nodige diskette-software. De reden daarvoor ligt eigenlijk voor de hand. De computer moet software krijgen om zowel met de 6502 als de Z80 te kunnen werken. Na het inschakelen "weet" de Octopus helemaal niet dat een Z80-kaart op zijn bus aanwezig is. Dat moet hem sowieso via software worden bijgebracht. En als dat toch moet gebeuren, dan kunnen we ook meteen een kompleet operating-system laden! Tussen dit operating-system en het monitorprogramma kan men op elk willekeurig moment heen en weer springen. Meer daarover in het artikel "CP/M-software voor de Octopus 65" in deze Special. En aan welk operating-system denkt u als de naam Z80 valt? Juist, CP/M!

De rechten die we voor het gebruik van CP/M op de Octopus zouden moeten betalen, zijn echter enorm. Daarom hebben we hiervoor een andere oplossing gezocht en gevonden. Uit de Verenigde Staten komt het operating-system ZCPR2. Dit is, zoals Big-Board-fans al lang weten, een CP/M-kompatibel operating-system dat niet alleen komfortabeler in het gebruik is dan CP/M 2.2, maar ook nog "public domain" is. Dit betekent dat men het systeem voor privé-gebruik mag toepassen zonder dat hiervoor iets hoeft te worden betaald. Dit fraaie operating-system is nu ook geïmplementeerd op de Octopus 65 met Z80-kaart. U hebt daarmee in principe toegang tot een reusachtige hoeveelheid CP/M-software die over de hele wereld praktisch gratis verkrijgbaar is.

Booten

De ZCPR2-systeemdiskette is verkrijgbaar in een 40- en een 80-track-uitvoering (waarover ook meer verteld wordt in het artikel "CP/M-software voor de Octopus 65"). Het is aan te bevelen dat u van deze diskette meteen een kopie maakt, zodat het origineel op een veilige plaats kan worden opgeborgen. Het kopiëren doet u, zoals gewoonlijk bij de Octopus, met punt 8 van het BEXEC*menu (single or dual disk drive copier). Daarvoor gebruiken we dus het Ohio-operating-system. Wie dat niet heeft en de Octopus alleen als CP/M-machine wil gebruiken (wat ook mogelijk is), die moet nog even wachten met kopiëren en eerst dit artikel verder doorlezen voordat hij aan de slag gaat.

Het booten van de nieuwe systeemschijf gebeurt net zoals bij een normale OSI-systeemschijf: computer inschakelen (of op RESET drukken), systeemdiskette in drive A steken en ".b" intypen. Nu moet op het beeldscherm hetzelfde verschijnen als in figuur 1 te zien is. We moeten hierbij nog even opmerken dat het systeem zich meldt met "EC 65" en niet met "Octopus", zoals we dat gewend zijn. De ZCPR2-software is namelijk van Duitse afkomst, en bij onze oosterburen heeft ons systeem een andere naam meegekregen. Komt u voortaan de naam EC-65 tegen, dan weet u dus dat daarmee de Octopus wordt bedoeld. Verschijnt er iets anders op het scherm (ofschoon de Z80-kaart is gekontroleerd met behulp van het testprogramma en de Octopus verder ook probleemloos funktioneert), dan ligt dat waarschijnlijk aan een onjuiste interrrupt-verwerking van uw systeem. De normale software voor de Octopus werkt namelijk volledig zonder interrupts (met uitzondering van de tracer, maar die werkt met een niet-maskeerbare interrupt NMI) en dan kan het voorkomen dat op de IRQ-lijn van uw computer vreemde dingen gebeuren die gewoonlijk geen invloed hebben op de goede werking van het systeem. De CP/M-software maakt echter veelvuldig gebruik van de interrupt-mogelijkheden van de Octopus. Een tot nu toe niet ontdekte fout heeft dan alsnog gevolgen.

Werkt uw systeem niet goed, kontroleer na het booten dan eerst eens de IRQ-lijn. Ze zit op de bus op aansluiting 12a. Als deze lijn voortdurend "0" is, dan ligt hier hoogstwaarschijnlijk de fout. De software geeft alleen VIA A (zie het schema van de Octopus) de mogelijkheid om interrupts op te wekken. Waarschijnlijk zit ergens in het systeem nog een interrupt-bron die een ongewenst steentje bijdraagt.

U kunt de dader het gemakkelijkste opsporen door de IC's stuk voor stuk te verwijderen uit hun voetje. Na deze grove lokalisering van de fout kunt u dan gaan kijken waar ze zich exakt bevindt en hoe deze kan worden opgelost.

Het is ook nog mogelijk dat uw machine zich ophangt zonder dat een extra interrupt-bron aanwezig is. Dat gebeurde ons namelijk op de redaktie met een exemplaar. De fout zat daar in de monitor-EPROM. Om de een of andere reden was het laatste byte in de EPROM niet geprogrammeerd (dat is ook vaak moeilijk: moet je nu bij het programmeren het laatste adres instellen of het laatste adres + 1?). Op die plaats in de EPROM zit juist de vektor die naar de ISR (Interrupt Service Routine) voert. Als de Octopus die vektor niet vindt, dan laat hij het bij de eerste IRQ ook meteen afweten.

Taktisch

Misschien denkt u dat de melding "6502 speed: 2 MHz" alleen voor het oog op het scherm staat. Nou, dat is beslist niet waar! Tijdens het booten wordt namelijk de klokfrekwentie van de processor "gemeten". Afhankelijk daarvan verschijnt op het scherm de mededeling "1 MHz" of "2 MHz" of "No standard 6502 speed!", als het systeem op een andere frekwentie draait. De zin van deze meting wordt u misschien duidelijk als uw Octopus is "opgevoerd" tot 2 MHz. Bij de besturing van de drives moet een vaste timing in acht worden genomen. Wil men de drivebesturing goed laten verlopen bij twee verschillende klokfrekwenties, dan heeft men daarvoor dus twee iets verschillende operating-systems nodig.

Bij onze CP/M-implementatie hoeft dat

1

```
  == EC 65  CP/M system ==

6502 speed: 2 MHz

CBIOS65 vers 3.3
CBIOS80 vers 3.4
  (C) Frank Schmidt

59k ZCPR2.3
  (C) Richard Conn

A>
```

Figuur 1. Zo meldt de computer zich na het booten van de ZCPR2-diskette. De klokfrekwentie van de 6502-CPU wordt daarbij automatisch "gemeten", waarna de timing voor de disk-routines hierop wordt aangepast. Eén enkel operating-system is dus voldoende voor twee verschillende klokfrekwenties.

overigens niet. Nadat de CPU-klok is gemeten, wordt de timing voor de drive-besturing daarop aangepast.

Verwisselen van diskettes

Voordat u zich nu gaat storten op het nieuwe operating-system, moet u het onderstaande stukje tekst beslist nog lezen, want dat kan u veel narigheid besparen.
Zoals alle dingen heeft ook ons CP/M-BIOS zijn schaduwkanten. In ons geval heeft dat betrekking op het verwisselen van diskettes. Bij de meeste CP/M-systemen geeft dat wel enige problemen, maar bij ons kan dat zelfs heel gevaarlijk zijn! Tenminste, als de onderstaande regels niet in acht worden genomen:

Voor en na elke diskette-wissel moet een warm-start worden gegeven (Ctrl-C)!

(Dat geldt niet als een programma u opdracht geeft om een diskette te wisselen!) Niet navolgen van deze regel kan destruktieve gevolgen voor de software tot gevolg hebben. De reden hiervoor zal in deel 3 worden gegeven. We willen hier alleen even vermelden dat de reden ligt in het gebruikte blocking/deblocking-algoritme, dat niet exakt volgens de aanbevelingen van Digital Research werkt, maar daarvoor het aantal disk-operaties wel aanzienlijk vermindert en zo de snelheid flink verhoogt. Er moeten nu eenmaal kompromissen gesloten worden.

Diskettes formatteren

Het kopiëren van CP/M-diskettes kunt u zonder problemen doen met behulp van het OHIO-DOS. Het initialiseren is echter niet meer mogelijk. Dat komt doordat de door OS65D voorbereide diskettes weliswaar beschrijfbaar zijn, maar nog niet gelezen kunnen worden (probeer het maar eens). Met de utility "INIDSK.COM" kunnen we nu verder werken. Figuur 2 toont hoe dat gaat. Eerst typt u in "INIDSK", dan drive-nummer en daarna het aantal tracks dat geïnitialiseerd moet worden. Momenteel zult u altijd punt 1 (40 tracks) of 3 (80 tracks) kiezen, later (als u met andere disk-formaten wilt werken) kunnen dat ook andere aantallen zijn. Pas hierbij wel op! Direkt op het moment dat het nummer voor het aantal tracks wordt ingetikt, start ook de initialisering. Werk daarom altijd op de volgende manier: Eerst INIDSK starten. Vervolgens een lege diskette in het gewenste loopwerk steken (er hoeft in dit geval geen Ctrl-C te worden gegeven) en dan het drive-nummer intypen. Heeft u een type-foutje gemaakt? Geen nood, geef een CR en probeer het dan nog eens. Is alles in orde, dan drukt u op de toets met het cijfer 3 (of 1). Nu gebeurt het volgende:
Op het beeldscherm verschijnt het nummer van de track die momenteel wordt bewerkt. Kan deze track om de een of andere reden niet worden geïnitialiseerd, dan verschijnt achter dit nummer de melding "Cannot init track". Het programma gaat daarna verder met het volgende track-nummer. Verloopt het initialiseren naar wens, dan wordt het oude track-nummer op het scherm vervangen door het nieuwe nummer. Nadat alle tracks zo

2

```
A>inidsk

Disk Initialisation Utility
Created 02-86 by F. Schmidt

Init disk on which drive (A..D) ? c
Init how many tracks ?
35 (0), 40 (1), 77 (2), 80 (3) or 160 (4)
Enter number (0..4): 3

The whole disk is now initialised

Init disk on which drive (A..D) ? <CR>

A>
```

86265-2

3

```
A>sysgen

CP/M - SYSGEN for JUNIOR
Version 1.1  -  05-01-86
Copyright (C) 1986 by Frank Schmidt


Enter source drive name (or RETURN to skip) a
Source on drive A, then press RETURN <CR>

Enter destination drive name (or RETURN to skip) c
Destination on drive C, then press RETURN <CR>


 - System generated -

Enter destination drive name (or RETURN to skip) <CR>

A>
```

zijn doorlopen, meldt het programma zich terug en kan een volgende schijf worden geïnitialiseerd. Het programma kan tussentijds worden gestopt met Ctrl-C of CR.

Figuur 2. Zo eenvoudig is het initialiseren van de diskettes (wat u moet intypen, is steeds onderstreept). Pas wel op! Zodra het cijfer voor het aantal tracks is ingedrukt, begint het programma direkt met initialiseren. Eerst dus de lege diskette in de drive stoppen!

Figuur 3. Zo kan men de systeem-tracks kopiëren. Het kopiëren kan ook geschieden op één enkele drive als na het lezen van de tracks een nieuwe schijf in de drive wordt gestopt.

Systeem- en data-diskettes

Het CP/M-operating-system bevindt zich op de buitenste tracks van de diskette (het beslaat in ons geval 5 tracks). Staat het systeem op een schijf, dan is dit altijd een systeem-diskette. Hierbij maakt het niet uit wat verder nog op de schijf staat. Alle andere schijven zijn data-diskettes.
Bij elke warm-start wordt een gedeelte van CP/M (BDOS en CCP) opnieuw in het geheugen geladen. Bij een standaard-CP/M-systeem **moet** in drive A altijd een systeemschijf aanwezig zijn. Bij ons systeem hoeft dat niet. Alleen tijdens het booten moet in drive A een systeem-diskette zitten. Tijdens dit proces wordt het operating-system dan eenmaal geladen en in het 6502-geheugen opgeslagen. Bij elke nieuwe warm-start wordt het operating-system gewoon vanuit het 6502-geheugen naar het Z80-geheugen gekopieerd. Dat bespaart een hoop tijd en de drive hoeft niet meer zo vaak te werken.
U zult ook wel eens een nieuwe systeem-diskette moeten maken. Dat gebeurt met behulp van de utility "SYSGEN.COM". Deze werd speciaal geschreven voor de Octopus met Z80-kaart, maar ze werkt precies hetzelfde als de originele versie van Digital Research. Figuur 3 laat aan de hand van een praktijkvoorbeeld zien hoe men met deze utility werkt. U kunt overigens ook de systeem-tracks kopiëren van drive A naar drive A, waarbij tusentijds dus van diskette wordt gewisseld. Ook in dit geval hoeft u bij het verwisselen geen Ctrl-C te geven.
Het is natuurlijk ook mogelijk om gewone files te kopiëren. Dat kunt u doen met de ZCPR2-utility "MCOPY.COM". Hoe deze utility exakt werkt, zou hier te ver voeren om te vertellen. Voorlopig is het voldoende om te weten dat u hiermee precies zo kunt werken als met de CP/M-utility "PIP.COM" (zie het artikel "CP/M in 90 minuten"). Verder kunt u met het kommando "MCOPY //" informatie over het programma opvragen.
Dat laatste geldt overigens voor alle ZCPR2-utilities. Als deze hulp-informatie bevatten, dan kan deze altijd met "$$" worden opgeroepen. Zo geeft het kommando "DIFF $$" bijvoorbeeld info over het programma "DIFF.COM".
Er is nog een derde mogelijkheid om

Figuur 4. Drive E is een RAM-floppy. Hiervoor wordt alle RAM-geheugen gebruikt dat behalve de standaard 64 K op de Z80-kaart aanwezig is. De uitdraai toont de disk-parameters voor een konfiguratie van 256 Kbyte. De software bepaalt deze zelf aan de hand van de aanwezige geheugenbanken. Bij een geheugenkapaciteit van "slechts" 64 Kbyte wordt een foutmelding gegeven bij het oproepen van drive E.

```
4


A>stat e:dsk:

    E: Drive Characteristics
 1512: 128 Byte Record Capacity
  189: Kilobyte Drive  Capacity
   64: 32  Byte Directory Entries
   64: Checked  Directory Entries
  128: Records/ Extent
    8: Records/ Block
   24: Sectors/ Track
    0: Reserved Tracks

A>
```

ZCPR2-informatie te krijgen. Die is namelijk te vinden in het "Software-handbook" voor de Big-Board II. Dit boekwerkje van zo'n 300 pagina's bevat zeer veel nuttige informatie. Ook voor andere CP/M-gebruikers is dit boek interessant. Het beschijft uitgebreid de mogelijkheden van het ZCPR2-systeem en de installatie hiervan. Ook de omgang met dit softwarepakket wordt behandeld. Men kan dit systeem echter alleen installeren bij een Z80-systeem en met een goede kennis van de Z80-machinetaal. Bovendien is een macro-assembler nodig. Dit even terzijde, nu weer terug naar ons eigen systeem.

Snelle schijven

Nu gaan we eens kijken naar een ander "feature" van onze CP/M-implementatie: de RAM-floppy.
CP/M kan standaard slechts 64 Kbyte RAM-geheugen beheren. De in deze uitgave beschreven CPU-kaart kan echter maximaal 1 megabyte adresseren!. Zelfs in de basisversie heeft de kaart een adresbereik van 256 Kbyte.
We raden elke Z80-kaart-bouwer aan om meteen 256-k-DRAM's op zijn kaart te zetten. De resterende drie geheugenbanken van 64 K kunnen dan als RAM-disk worden gebruikt. Ze worden door het systeem ook helemaal beschouwd als "disk". Voor de gebruiker zijn er echter twee belangrijke verschillen tussen een "echte" floppy en een RAM-disk:

— Na het uitschakelen van de computer is de RAM-disk "leeg".
— De schrijf- en leestijden van de RAM-disk zijn aanzienlijk korter dan bij een echte disk.

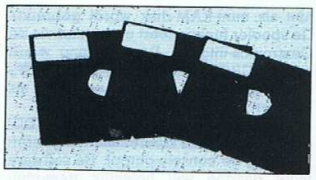
Voor ons is vooral het tweede punt van belang. Maar daarover later meer.
De software herkent automatisch de hoeveelheid geheugen die ter beschikking staat en beschouwt alle geheugen dat boven de noodzakelijke 64 KB uitkomt als RAM-disk. Figuur 4 toont een "STAT DSK:"-uitdraai van drive E — de RAM-disk — bij een RAM-konfiguratie van 256 Kbyte.
Als op de Z80-kaart slechts 64 Kbyte aanwezig is, verschijnt bij het oproepen van drive E een foutmelding. Maar zelfs in dit geval heeft men nog een soort RAM-disk tot zijn beschikking. Het geheugen van de 6502 bevat ongeveer 40 Kbytes die in de "CP/M-mode" niet worden benut. Die kunnen dus ook als een mini-RAM-disk worden gebruikt. De software hiervoor staat reeds op de systeem-schijf. Ze hoeft alleen maar te worden geaktiveerd. Gewoonlijk wordt dit geheugenbereik als printer-spooler gebruikt; deze funktie moet in dit geval dus worden uitgeschakeld. Ook dit zullen we straks nog nader toelichten. U kunt dus kiezen tussen een kleine RAM-disk en een printer-spooler.
Waarvoor gebruikt men nu zo'n RAM-disk?
De meeste, onder CP/M werkende compilers, assemblers en andere "omzetters" werken als volgt: ze lezen een file in, vertalen deze en leveren als output dan een tweede file (of zelfs meerdere). De files worden niet netjes na elkaar gelezen of geschreven, maar net zoals dit het programma het beste past: een sektor van file A lezen, dan naar file B schrijven, etcetera (CP/M-sektoren zijn altijd 128 bytes lang). Aangezien de files op verschillende plaatsen op de schijf staan, moet de lees/schrijf-kop steeds tussen die files heen en weer worden gestuurd. Dat kost relatief veel tijd en is ook niet zo best voor de diskettes en de drives.
Bij een RAM-disk maakt dat allemaal niets uit. Daar is geen kop die op en neer moet fietsen, er hoeven maar een paar adressen te worden gegeven en we zitten al op de juiste plaats in het geheugen.
De snelle schijf is dan ook vooral geschikt voor het "vertalen" of bewerken van programma's. Ook is het verstandig om daarin veelgebruikte programma's op te slaan.
En dan is er nog een derde toepassingsmogelijkheid: het kopieëren van data-files via een enkele drive. Dat is bij CP/M namelijk niet mogelijk! De oplossing is simpel: file eerst in RAM-disk kopiëren, Crtl-C, diskette omwisselen, Ctrl-C, file van RAM-disk naar andere diskette kopiëren. Klaar!
Tenslotte nog een paar getallen. Het assembleren van een bepaald programma duurt op een gewone diskette ongeveer 42 sekonden. In de "kleine" RAM-disk zijn hiervoor maar 11 sekonden nodig en in de "grote" RAM-disk slechts 9 sekonden.

Input/output

We hebben zojuist al verteld dat de software voor de Octopus/Z80 ook een programma bevat dat de 40 Kbyte van het 6502-geheugen als printer-spooler laat funktioneren (als dat geheugen tenminste niet als mini-RAM-disk wordt gebruikt). De spooler funktioneert uitsluitend via de Centronics-uitgang. Deze uitgang wordt vanuit CP/M geadresseerd als LST:-device. De RS-232-interface werkt hier als PUN: en RDR:.
Het is mogelijk om in plaats van de VDU-kaart de grafische kleurenkaart te gebruiken voor de console-output. De software moet daartoe wel lichtelijk gemodificeerd worden. Ook dit zal behandeld worden in het derde deel dat u in deze uitgave kunt vinden. Voor het booten van ZCPR2 moet dan wel eerst de graphics-software worden geladen. Verder moet er voor worden gezorgd dat deze software niet kan worden overschreven (bijvoorbeeld door de printer-spooler). Men kan natuurlijk een "patch"-programma bedenken dat dit allemaal automatisch regelt.
De console-input is gebufferd (256 bytes). U kunt dus gerust al een nieuw kommando intypen terwijl de computer nog bezig is met het uitvoeren van het vorige. Hierbij is één uitzondering: tijdens disk-access zijn interrupts verboden, zodat dan ook niet op het toetsenbord kan worden gewerkt.
In de software zit nog een aardige vondst verwerkt. De meeste beeldscherm-georiënteerde tekstverwerkers (zoals Wordstar, maar ook de editor van "Turbo Pascal") hebben een terminal-funktie nodig waarmee een direkte cursor-adressering mogelijk is. Dat is zoiets als "PRINT AT" in BASIC. We hebben die funktie meteen geïntegreerd. Zo wordt met $27,$11,x,y (x = 0...79, y = 0...23) de cursor in de x-de kolom en de y-de regel geplaatst.

Kompatibiliteit, een schone zaak

Helaas moeten we dit artikel beëindigen met een niet zo prettige mededeling: Er is momenteel nog geen enkel CP/M-diskformaat dat door de Octopus kan worden gelezen. Er zullen wel CP/M-programma's voor de Z80-Octopus in omloop komen, maar dat is niet die enorme stapel die de gewone CP/M-gebruiker ter beschikking staat. Er wordt door enkele enthousiaste hobbyisten echter gewerkt aan een tweede floppy-controller, die in de toekomst misschien nog eens zal worden gepubliceerd in een Elektuur-uitgave.
Mocht u toch graag sommige CP/M-programma's op uw computer willen laten draaien, dan bestaat daarvoor een uiterst simpele oplossing. Ga met uw Octopus naar een kennis die een CP/M-machine bezit en koppel de computers aan elkaar via de RS232-interface. U kunt de bewuste programma's dan overspelen via dit seriële kanaal en ze vervolgens op schijf opslaan. En mocht die kennis wat verder weg wonen: wat dacht u van een koppeling via een modem?
Na al deze theorie kunt u eerst eens gaan experimenteren met ZCPR2. Veel plezier daarmee!



CP/M op de Octopus 65 — deel 3

de software-know-how voor de Z80-kaart

In het tweede deel van deze artikelenreeks hebben we al iets verteld over de opzet van de software voor de Z80-kaart. In dit derde deel gaan we deze software wat gedetailleerder bekijken en vertellen wat over de realisatie ervan. Daarbij wordt vooral gekeken naar het 6502-gedeelte van het systeem.

Het basisgedeelte van de software voor de Z80-kaart bestaat grofweg uit zeven delen:

- Octopus-65-EPROM-software (I/O-besturing, boot-routine)
- BIOS65, deel 1: I/O-handler
- BIOS65, deel 2: disk-routines voor Ohio-kompatibele controllers
- BIOS65, deel 3: RAM-disk- en printer-spooler-routines (en eventuele software voor een tweede disk-controller)
- Z80-EPROM-software
- BIOS80
- operating-system (ZCPR2, CP/M,...)

Deze delen sluiten "naadloos" op elkaar aan. Daartoe een klein voorbeeld. Stel dat we ZCPR2 hebben gestart en de computer wil dan de prompt "A" op het beeldscherm laten zien. Eerst worden de karakters overgedragen aan BIOS80. Van daaruit gaat het via de Z80-EPROM-routines naar de I/O-handler van BIOS65. Dit laatste vertaalt het door de Z80 van de console ontvangen kommando "teken naar console sturen" en stuurt de ontvangen data weer verder, in dit geval naar de video-handler in de Octopus-65-EPROM. Dat levert dan het gewenste resultaat.

De I/O-handler in het BIOS65 is zo'n beetje de "afdelingschef" van het 6502-systeem. Hij ontvangt kommando's "van boven" en zorgt er vervolgens voor, dat de daarvoor benodigde werkzaamheden worden verdeeld. Hij heeft hier dan ook een vrij centrale, belangrijke funktie. Dat is de reden waarom we er straks nog wat meer over zullen vertellen.

De Z80-software valt eigenlijk buiten de opzet van dit verhaal; daar zal hier verder dan ook niet op worden ingegaan.

Software-schakelcentrale

In figuur 1 is de indeling gegeven van het 6502-geheugen. Zoals men kan zien, werd het BIOS in twee delen gesplitst. Dat is gedaan om het geheugenbereik optimaal te kunnen benutten. Hierbij komt een probleem om de hoek kijken. Pagina $23xx kan niet volledig worden gebruikt, omdat hierin de variabele AHOLD staat. Deze wordt door de diverse I/O-drivers gebruikt.

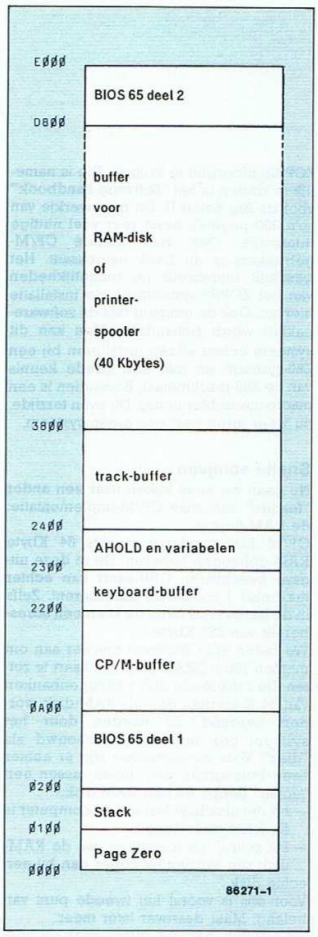
Helemaal aan het begin van het programma staan vektoren voor de verschillende I/O-routines. Deze vormen de "koppeling" wanneer men andere apparaten in het systeem wil opnemen, bijvoorbeeld een grafische kaart. Enige voorzichtigheid is geboden bij de vektoren CI en CO:

CO springt gewoonlijk direkt naar de video-handler van de Octopus. Deze sprong wordt in de loop van het boot-proces ge-"patched", dat wil zeggen overschreven. Dit gebeurt in regel 2160 en verder. Als de tweede track van de diskette wordt gelezen, dan staat een iets uitgebreidere VDU-driver ter beschikking, die ook een absolute cursor-adressering toestaat. Logischerwijs wordt de console-output in dat geval naar die driver omgeleid.

De sprong CI voert niet naar een ROM-routine. Hiermee wordt gewoon een karakter uit de keyboard-buffer gelezen. Staat er geen teken in de buffer, dan wordt gewacht tot eentje is binnengekomen. Deze wachttijd wordt tevens gebruikt om de printer-spooler te laten draaien (regel 6320 en verder). De keyboard-buffer wordt overigens via een interrupt-routine gevuld (regel 6550 e.v.).

Als men de CI-vektor verandert, dan moet meestal ook de routine worden veranderd die de console-status aan de Z80 doorgeeft (regel 4680 e.v.). Dat is dan ook de reden waarom men de CI-vektor zo weinig mogelijk moet veranderen.

Vanaf het label "START" begint het zaakje pas echt goed te werken. Eerst komen een hele reeks initialiseringen: stack-pointer, PIA, verschillende pointers en variabelen, timers en keyboard-poort. Hierna wordt de directory van de RAM-disk gewist. Dan verschijnt de eerste systeemmelding. De klokfrekwentie van de CPU wordt gemeten en de timing wordt, indien nodig, aangepast. Daarna verschijnt een tweede melding, de informatie over de klokfrekwentie van het systeem. Vervolgens wordt de tweede helft van BIOS65 van schijf geladen en de CO-vektor veranderd. De nu aanwezige spooler wordt geïnitialiseerd en tenslotte worden de tracks 2, 3 en 4 in de CP/M-buffer geladen (regel 2270). Tenslotte wordt een kommando naar de Z80 gestuurd: voer een "cold start" uit. Dit betekent dat het kommando wordt overgedragen aan de Z80. Als afscheidsgroet verschijnt nog het versienummer van BIOS65 op het scherm, waarna de 6502 in een eindloze lus springt bij label "MAINLP" (regel 4460). Deze processor mag nu alleen nog maar bijzaken doen, namelijk het uitvoeren van kommando's die door de Z80 worden gegeven. De meeste van deze kommando's kunnen vrij gemakkelijk worden uitgevoerd en zijn dan ook vrij eenvoudig van opzet. Het wordt eigenlijk alleen gekompliceerd bij de disk-besturing. Dat gaan we nu bekijken.



Figuur 1. De geheugenindeling van de Octopus 65 als we werken met CP/M. Het hele geheugenbereik wordt hierbij optimaal benut.

Disk-handling

Het grote probleem bij de besturing van de disk-drives is het feit dat CP/M het beschrijven en lezen van de floppies altijd uitvoert met "porties" van 128 bytes. Het is echter raadzaam om het aantal sektoren per track zo klein mogelijk te houden. Anders is vrij veel ruimte nodig voor formatteringsinformatie, waardoor de tota-

le, werkelijk bruikbare kapaciteit van de floppy sterk wordt verminderd. Dat is ook de reden waarom bij het Ohio-formaat de maximale sektorlengte 2 Kbyte bedraagt — sektor- en track-lengte zijn in dat geval identiek. We hebben dan ook een oplossing moeten bedenken om om de inkompatibiliteit wat betreft die sektorlengtes te kunnen overbruggen. Tegelijkertijd moest er op worden gelet dat het hierbij gebruikte omrekeningsalgoritme zoveel mogelijk bijdroeg tot een hoge verwerkingssnelheid bij disk-operaties.

Laten we eens kijken hoe het lezen van de schijf verloopt.

Voor het lezen van data van de zwarte schijf moet het operating-system drie parameters opgeven: het drive-nummer, de track en de (128-byte-)sektor. Het BIOS adresseert nu eerst de gekozen drive, positioneert de lees/schrijf-kop boven de gewenste track en laadt dan de hele track-inhoud in een buffer. Die buffer heeft een omvang van 5 Kbyte. Waarschijnlijk had u 2 Kbyte verwacht, maar die ruimte is wat groter gekozen in verband met eventuele afwijkende diskformaten in de toekomst.

Het sektor-nummer wordt nu als index gebruikt om uit de hoeveelheid ingelezen informatie een blok van 128 bytes te kiezen dat dan naar de Z80 wordt gestuurd.

Wil het operating-systeem hierna nog meer data van de schijf lezen, dan worden eerst de opgegeven drive- en tracknummers vergeleken met de vorige. Als ze niet hetzelfde zijn, wordt de hierboven beschreven procedure herhaald. Zijn ze wel identiek, dan loont het de moeite dat we de hele track hebben ingelezen, want in dat geval hoeft niet opnieuw van de schijf te worden gelezen en kan de gewenste sektor direkt vanuit de buffer naar de Z80 worden gestuurd. Deze procedure verhoogt de snelheid van de diskroutines aanzienlijk, als tenminste verschillende opeenvolgende sektoren moeten worden gelezen, zoals bijvoorbeeld bij het inlezen of wegschrijven van een programma. Bij het werken met compilers en assemblers ligt de zaak helaas niet zo eenvoudig. Daarvoor kunt u beter de RAM-disk gebruiken.

Er moet natuurlijk ook op de schijf geschreven kunnen worden. Dat gaat in principe op dezelfde wijze als het lezen. Eerst wordt de geselekteerde track in de buffer geladen, als dat nog tenminste nog niet is gebeurd. Uit het sektor-nummer wordt het start-adres van de sektor in de buffer berekend, waarna de Z80-data op de aangegeven plaats in die buffer worden weggeschreven.

We wilden de data echter op de schijf zetten, niet in de buffer. De gehele, veranderde buffer-inhoud zou nu naar de schijf kunnen worden gekopieerd. Dat heeft echter een nadeel: Wil men op deze wijze 16 opeenvolgende sektoren naar de disk schrijven, dan moet daarvoor de hele buffer-inhoud 16 keer naar de disk worden gekopieerd. Dat kost veel tijd. We laten de data daarom voorlopig in de buffer staan en onthouden dat de inhoud later nog naar de schijf moet worden gekopieerd.

De kopieer-procedure vindt plaats als een nieuwe track in de buffer moet worden geladen of als dat uitdrukkelijk wordt bevolen door het operating-system — dat gebeurt bijvoorbeeld bij een warm-start van het operating-system.

Nu kan ook de waarschuwing uit het tweede deel worden verklaard:

Stel dat het kommando "ERA **" wordt gegeven. Het operating-system zal dan eerst de track laden waarop de directory staat. Alle zich daarin bevindende namen worden gewist, waarna de track-inhoud weer naar de schijf wordt geschreven. Aangezien bij deze procedure steeds dezelfde track wordt opgeroepen, wordt de drive zelf na de eerste maal inlezen niet meer geaktiveerd. Na de uitvoering van het door ons gegeven wis-kommando staat de gewiste directory wel in de buffer, maar nog niet op schijf. Als u nu een andere schijf in de drive stopt en daarmee verder werkt, dan gaat het goed mis. Zodra het systeem met een andere track moet gaan werken of u een andere drive kiest, wordt de "oude" (lege) directory op de "nieuwe" schijf gekopieerd. De originele directory van de schijf gaat dan verloren en de schijf is zo goed als onbruikbaar geworden.

Een warm-start vóór het wisselen van floppy heeft tot gevolg dat de buffer-inhoud op de oorspronkelijke schijf wordt weggeschreven. Nadat u een andere floppy in de drive hebt gestopt, zorgt een tweede warm-start er voor dat de directory van de nieuwe floppy in de buffer wordt gelezen.

Details

Van de buffer-inhoud moet een flinke hoeveelheid gegevens worden bijgehouden: het adres van de buffer-inhoud op de disk (drive- en track-nummer), de informatie dat de buffer zinvolle data bevat (of gewoon leeg is), en tenslotte de informatie dat de buffer-inhoud is veranderd en dus later weer moet worden weggeschreven naar disk. De geheugenplaatsen die deze info bevatten, dragen de namen DRIVE, TRACK, TRFLAG en WRFLAG.

De hele verwerking van de track-buffers gebeurt door het subprogramma "SET" (regel 3000 e.v.). Dit programma ontvangt de drive-, track- en sektorkode van de Z80 en zet deze in een tussengeheugen. Daarna wordt gekontroleerd of de buffer data bevat die nog van belang zijn. Is dat niet het geval, dan wordt naar label "SETB" gesprongen (regel 3320). Daar wordt de nieuwe track dan in de buffer geladen en worden de desbetreffende flags gezet. Daarmee heeft deze routine zijn werk gedaan.

Misschien bevat de buffer nog een waardevolle track, die echter niet de gewenste is. In dat geval wordt naar label "SETA" in regel 3220 gesprongen. Daar wordt, voordat een nieuwe track wordt ingelezen, eerst gekontroleerd welke info de variabele "WRFLAG" bevat. Een "0" in deze geheugenplaats geeft aan dat de oude track eerst naar de schijf moet worden geschreven voordat de nieuwe kan worden binnengehaald.

CP/M verwerkt alle data in binaire vorm. Het Ohio-DOS werkt daarentegen met BCD-kodes bij de track-nummers. In regel 3140 e.v. wordt daarom gekeken of een Ohio-kompatibele drive is geselekteerd (een van de eerste vier). Is dat zo, dan wordt de track-kode vertaald in een BCD-kode.

Bij het verlaten van de routine "SET" staat altijd de gekozen track in de buffer. Hoewel... het kan natuurlijk wel eens gebeuren dat in de geselekteerde drive geen floppy aanwezig is, of misschien kan een track door een vuiltje of een kras niet meer worden ingelezen. In zo'n geval wordt het Z-bit gereset, waarna de routine "SET" wordt verlaten en het overkoepelende programma een foutmelding doorgeeft aan CP/M.

Het omrekenen van het sektor-nummer naar het effektieve adres waarop de sektor in de buffer begint, wordt uitgevoerd door de routine "CALCTR" (regel 3500 e.v.). "MOVSEC" en "RECSEC" zorgen voor het heen en weer verplaatsen van de bytes.

Het overige gedeelte van het programma is vrij normaal van opzet. Wie thuis is in de 6502-machinetaal en een woordje Engels verstaat (wie kent dat tegenwoordig niet?), die heeft verder beslist geen moeilijkheden bij het doorgronden van het programma dat hierna is afgedrukt.

```
10          ; ************************************
20          ; *                                  *
30          ; *  CP/M BIOS for EC 65  /  6502 part  *
40          ; *       File Nr. 0                 *
50          ; *                                  *
60          ; *  Main I/O handler for the Z80 CPU  *
70          ; *       1 / 2 MHz auto select      *
80          ; *       program version 3.4        *
90          ; *       last revision date: 22/03/86  *
100         ; *                                  *
110         ; *  Written 1985/86 by  Frank Schmidt  *
120         ; *                                  *
130         ; ************************************
140         ;
150         ;
160         ;
170         ;
180 0200    *       = $0200
190         ;
200         ;
210         ;  I/O Handling Addresses
220         ;
230 E101=   VAPAD   = $E101  ; Port A data (keyb.)
240 E104=   VATACL  = $E104  ; timer 1, counter low
250 E105=   VATACH  = $E105  ;   "     counter high
260 E10B=   VAACR   = $E10B  ; auxiliary control register
270 E10C=   VAPCR   = $E10C  ; peripheral control register
280 E10D=   VAIFR   = $E10D  ; Int. Flag Reg.
290 E10E=   VAIER   = $E10E  ; Int. enable Reg.
300         ;
```

```
310             ;
320             ;  Floppy Interface Addresses
330             ;
340 E002=       FDRB   = $E002 ; data/data dir reg B
350             ;
360             ;
370             ;  DOS Locations
380             ;
390 2363=       AHOLD  = $2363 ; buffer for character
400 235E=       SECTOR = $235E ; sector number
410 235C=       DRIVE  = $235C ; drive number
420 2370=       VECLO  = $2370 ; buffer for page 0 load vector
430 2371=       VECHI  = $2371
440 235D=       TRACK  = $235D ; track number
450             ;
460 0A00=       CPMA   = ++$0800 ; buffer for CP/M (6Kbytes)
470 2200=       KEYBUF = ++$2000 ; buffer for keyboard - input
480 2400=       BLOCK  = ++$2200 ; buffer for actual track (5Kbytes
490 3800=       RAMDSK = ++$3600 ; buffer for RAM - disk
500 D800=       PARTWO = $D800   ; 2nd part of program
510             ;
520 E7CB=       IRQL   = $E7CB
530 E7CC=       IRQH   = IRQL+1
540             ;
550             ;
560             ;  Z80 Handshaking Addresses
570             ;
580 E300=       PAD    = $E300  ; port A data/data direction
590 E301=       PAC    = PAD+1  ; port A control register
600 E302=       PBD    = PAD+2  ; port B data/data direction
610 E303=       PBC    = PAD+3  ; port B control register
620             ;
630             ;
640             ; Page Zero Addresses
650             ;
660 0000=       POINTL = 0      ; universal pointer
670 0001=       POINTH = 1
680 0002=       MEMNTL = 2      ; memory pointer
690 0003=       MEMNTH = 3
700 0004=       WRFLAG = 4      ; 0: data allready on disk
710 0005=       TRFLAG = 5      ; 0: data valid
720 0006=       IODRV  = 6      ; next access drive number
730 0007=       IOTRK  = 7      ; next access track number
740 000B=       CIFLAG = 8      ; 0: data in keyboard buffer
750 0009=       KEYPNT = 9      ; pointer into KEYBUF
760 000A=       COUNTR = 10     ; counter for UNLOAD - timeout
770 00FE=       MEMLO  = $FE
780 00FF=       MEMHI  = $FF
790             ;
800             ;
810 0200 4C2302         JMP START
820             ;
830             ;
840             ; I/O Vector Table
850             ;
860 0203 201206 CI      JSR GETCHA ; console input
870 0206 AD6323         LDA AHOLD
880 0209 60             RTS
890             ;
900 020A 8D6323 CO      STA AHOLD  ; video output
910 020D 4C00F0         JMP $F000
920             ;
930 0210 2070F7 READER JSR $F770  ; serial input
940 0213 AD6323         LDA AHOLD
950 0216 60             RTS
960             ;
970 0217 8D6323 PUNCH  STA AHOLD  ; serial output
980 021A 4C7EF7         JMP $F77E
990             ;
1000 021D 8D6323 LIST   STA AHOLD  ; Centronics output
1010 0220 4C1508        JMP INSCHR ; change into $F7B0, if no
1020                               ; printer-spooler used
1030            ;
1040            ;
1050            ;  Start of Program
1060            ;
1070 0223 A2FF  START  LDX #$FF   ; init stack pointer
1080 0225 9A            TXS
1090            ;
1100            ;
1110            ;  Init PIO
1120            ;
1130 0226 A928          LDA #%00101000
1140 0228 8D01E3        STA PAC
1150 022B A900          LDA #0    ; all inputs
1160 022D 8D00E3        STA PAD
1170 0230 A92C          LDA #%00101100
1180 0232 8D01E3        STA PAC
1190            ;
1200 0235 A928          LDA #%00101000
1210 0237 8D03E3        STA PBC
1220 023A A9FF          LDA #$FF    ; all outputs
1230 023C 8D02E3        STA PBD
1240 023F A92C          LDA #%00101100
1250 0241 8D03E3        STA PBC
1260 0244 A9FF          LDA #$FF    ; initialize handshaking
1270 0246 8D02E3        STA PBD
1280 0249 A214          LDX #20
1290            ;
1300 024B AD00E3 HNDSHK LDA PAD
1310 024E CA            DEX
1320 024F D0FA          BNE HNDSHK
1330            ;
1340            ;
1350            ; Other Initialisations
1760            ;
1370 0251 A2FF          LDX #$FF
1380 0253 8608          STX CIFLAG  ; no data in KEYBUF
1390 0255 E8            INX
1400 0256 8609          STX KEYPNT
1410 0258 8604          STX WRFLAG  ; all data on disk
1420 025A E8            INX
1430 025B 8605          STX TRFLAG  ; data in BLOCK not valid
1440 025D A935          LDA #GETISR ; init IRQ vector
1450 025F 8DCBE7        STA IRQL
1460 0262 A906          LDA #GETISR/256
1470 0264 8DCCE7        STA IRQH
1480 0267 AD01E1        LDA VAPAB   ; clear PA1 flag
1490 026A AD04E1        LDA VATACL  ; clear T1 flag
1500 026D AD0BE1        LDA VAACR   ; get auxiliary control reg
1510 0270 293F          AND #$3F    ; reset bits 6 & 7:
1520 0272 8D0BE1        STA VAACR   ; timer 1 one shot mode
1530 0275 A9C2          LDA #%11000010
1540 0277 8D0EE1        STA VAIER   ; enable timer & keyb IRQ
1550 027A 58            CLI
1560 027B A200          LDX #0      ; after booting, head
1570 027D 8E5D23        STX TRACK   ; is on track zero,
1580 0280 8E5C23        STX DRIVE   ; drive A
1590 0283 A900          LDA #RAMDSK
1600 0285 8500          STA POINTL  ; clear RAM - Directory
1610 0287 A938          LDA #RAMDSK/256
1620 0289 8501          STA POINTH
1630 028B A208          LDX #2048/256
1640 028D A000          LDY #0
1650 028F A9E5          LDA ##E5
1660            ;
1670 0291 9100  CLRMDI STA (POINTL),Y
1680 0293 C8            INY
1690 0294 D0FB          BNE CLRMDI
1700 0296 E601          INC POINTH
1710 0298 CA            DEX
1720 0299 D0F6          BNE CLRMDI
1730 029B 200204        JSR PRMES
1740 029E 20C909        JSR COUNT   ; get the CPU frequency
1750 02A1 18            CLC
1760 02A2 6907          ADC #7
1770 02A4 29F0          AND #$F0    ; mask off last bits
1780 02A6 C940          CMP #$40    ; this is the 1 MHz count
1790 02A8 D00B          BNE TWOMHZ  ; if not, 2 MHz?
1800 02AA A931          LDA #'1
1810 02AC 8D5D04        STA FREQ
1820 02AF 201B04        JSR FRQMES  ; print message
1830 02B2 4CD402        JMP MAIN    ; continue
1840            ;
1850 02B5 C980  TWOMHZ CMP #$80    ; 2 MHz count
1860 02B7 D00E          BNE NOSTND  ; no standart frequency
1870 02B9 20E009        JSR ADJTIM  ; adjust timing for 2 MHz
1880 02BC A932          LDA #'2
1890 02BE 8D5D04        STA FREQ
1900 02C1 201B04        JSR FRQMES  ; print message
1910 02C4 4CD402        JMP MAIN    ; continue
1920            ;
1930 02C7 1008  NOSTND BPL ONEADP
1940 02C9 20E009        JSR ADJTIM  ; adjust timing for 2 MHz
```

```
02CC A972          LDA # 2     ; real frequency is higher
02CE 8D7E04        STA FREQAD  ; than 2 MHz
                ;
02D1 202204     ONEADF JSR PRADAP  ; print a message
                ;
                ;
02D4 A900     MAIN  LDA #0      ; select drive A
02D6 203007         JSR SETDRV  ; if error occurs
02D9 D02D           BNE ERROR   ; RESET the system
02DB A901           LDA #1      ; move head on track 1
02DD 20EF06         JSR SETTK
02E0 D026           BNE ERROR
02E2 A900           LDA #PARTWO ; init MEPNT to store
02E4 8502           STA MEPNTL  ; 2nd part of program
02E6 A9D8           LDA #PARTWO/256
02E8 8503           STA MEPNTH
02EA 78             SEI         ; disable IRQ
02EB 205C07         JSR READ    ; read track into BLOCK
02EE 58             CLI         ; enable IRQ again
02EF D017           BNE ERROR
02F1 205B03         JSR SAVEBU  ; move data to PARTWO
02F4 A90F           LDA #EXTVDU ; install extended VDU routines
02F6 8D0E02         STA CO+4    ; change this into NOPs,
02F9 A9DB           LDA #EXTVDU/256 ; if not used
02FB 8D0F02         STA CO+5
02FE 201208         JSR INSPLR  ; init spooler
0301 A902           LDA #2      ; put head on track 2
0303 20EF06         JSR SETTK
0306 F003           BEQ MAINA   ; branch if no error
                ;
0308 6CFCFF   ERROR  JMP ($FFFC) ; JMP via RESET vector
                ;
030B A903     MAINA LDA #3      ; read next 3 tracks
030D 8500           STA POINTL
030F A900           LDA #CPMA   ; store data from CPMA on
0311 8502           STA MEPNTL
0313 A90A           LDA #CPMA/256
0315 8503           STA MEPNTH
                ;
0317 78       LDCPM SEI
0318 205C07         JSR READ    ; read one track
031B 58             CLI
031C D0EA           BNE ERROR
031E 205B03         JSR SAVEBU
0321 C600           DEC POINTL  ; next track
0323 F00C           BEQ MAINB   ; was it last track?
0325 AE5D23         LDX TRACK
0328 E8             INX
0329 8A             TXA
032A 20EF06         JSR SETTK
032D D0D9           BNE ERROR
032F F0E6           BEQ LDCPM
                ;
0331 A902     MAINB LDA #2      ; Z80: start program
0333 204F03         JSR OUT
                ;
0336 A9F8           LDA #$F8    ; at $F80F: COLDBOOT
0338 204F03         JSR OUT
033B A90F           LDA #$0F
033D 204F03         JSR OUT
                ;
0340 202904         JSR BYMES   ; print bios version message
                ;
0343 4CAD04         JMP MAINLP  ; wait for Z80-command
                ;
                ;
                ;
                ; Subroutines
                ;
                ; IN : receive one byte from Z80
                ;
0346 2C01E3   IN    BIT PAC     ; wait until PIA contains
0349 10FB           BPL IN      ; new data
034B AD00E3         LDA PAD     ; get data
034E 60             RTS
                ;
                ;
                ; OUT : transmit one byte to Z80
                ;
034F 2C03E3   OUT   BIT PBC     ; wait until last byte
0352 10FB           BPL OUT     ; accepted
0354 CD02E3         CMP PBD     ; reset PBC flag
0357 8D02E3         STA PBD     ; store data in PIA
035A 60             RTS
                ;
                ; SAVEBU : move BLOCK to (MEPNTL)
                ;
035B A900    SAVEBU LDA #BLOCK
035D 85FE           STA MEMLO
035F A924           LDA #BLOCK/256
0361 85FF           STA MEMHI
0363 A208           LDX #8      ; there are 8 pages
0365 A000           LDY #0
                ;
0367 B1FE    SAVELP LDA (MEMLO),Y
0369 9102           STA (MEPNTL),Y
036B C8             INY
036C D0F9           BNE SAVELP
036E E6FF           INC MEMHI
0370 E603           INC MEPNTH
0372 CA             DEX
0373 D0F2           BNE SAVELP
0375 60             RTS
                ;
                ;
                ; SET : set drive, track
                ;
0376 204603   SET   JSR IN      ; drive number
0379 8506           STA IODRV   ; save it
037B 204603         JSR IN      ; track number
037E 8507           STA IOTRK
0380 204603         JSR IN      ; sector number
0383 8D5E23         STA SECTOR
0386 A505           LDA TRFLAG  ; valid data in BLOCK?
0388 D028           BNE SETB    ; if not
038A A506           LDA IODRV   ; drive already selected?
038C CD5C23         CMP DRIVE
038F D012           BNE SETA    ; no
0391 A507           LDA IOTRK   ; track already selected?
0393 AE5C23         LDX DRIVE   ; if OHIO format, use BCD-code
0396 E004           CPX #4      ; for track number
0398 B003           BCS SETAA
039A 20FE05         JSR HEXBCD
039D CD5D23   SETAA CMP TRACK
03A0 D001           BNE SETA    ; no
03A2 60             RTS         ; all right, ret Z=1
                ;
03A3 A504     SETA  LDA WRFLAG  ; data in BLOCK must be changed
                                ; is it allready on disk?
03A5 F008           BEQ SETB    ; yes
03A7 A900           LDA #0      ; data in BLOCK exists on disk
03A9 8504           STA WRFLAG
03AB 78             SEI
03AC 207C08         JSR WRITE   ; put it on disk
03AF 58             CLI
03B0 D010           BNE SETEND  ; error
                ;
03B2 A901     SETB  LDA #1      ; no valid data in BLOCK
03B4 8505           STA TRFLAG
03B6 A506           LDA IODRV
03B8 203007         JSR SETDRV
03BB D012           BNE SETEND  ; error
03BD A507           LDA IOTRK
03BF 20EF06         JSR SETTK
03C2 D00B           BNE SETEND
03C4 78             SEI
03C5 205C07         JSR READ    ; load new track
03C8 58             CLI
03C9 D004           BNE SETEND
03CB A900           LDA #0      ; valid data in BLOCK
03CD 8505           STA TRFLAG
                ;
03CF 60      SETEND RTS         ; Z=1: ok   Z=0: error
                ;
                ;
                ; CALCTR : calculate sector start address
                ;
03D0 AD5E23   CALCTR LDA SECTOR
03D3 4A             LSR A
03D4 08             PHP
03D5 18             CLC
03D6 65FF           ADC MEMHI
03D8 85FF           STA MEMHI
```

```
3580 03DA 28            PLP
3590 03DB 9006          BCC CALCA
3600 03DD A97F          LDA #$7F
3610 03DF 65FE          ADC MEMLO
3620 03E1 85FE          STA MEMLO
3630            ;
3640 03E3 A000   CALCA  LDY #0
3650 03E5 60            RTS
3660            ;
3670            ;
3680            ; MOVSEC : move sector to Z80
3690            ;
3700 03E6 20D003 MOVSEC JSR CALCTR
3710            ;
3720 03E9 B1FE   MVSCLP LDA (MEMLO),Y
3730 03EB 204F03        JSR OUT
3740 03EE C8            INY
3750 03EF C080          CPY #$80
3760 03F1 D0F6          BNE MVSCLP
3770 03F3 60            RTS
3780            ;
3790            ;
3800            ; RECSEC : receive sector from Z80
3810            ;
3820 03F4 20D003 RECSEC JSR CALCTR
3830            ;
3840 03F7 204603 RCSCLP JSR IN
3850 03FA 91FE          STA (MEMLO),Y
3860 03FC C8            INY
3870 03FD C080          CPY #$80
3880 03FF D0F6          BNE RCSCLP
3890 0401 60            RTS
3900            ;
3910            ;
3920            ; PRMES : print message
3930            ;
3940 0402 A930   PRMES  LDA #MESS
3950 0404 A204          LDX #MESS/256
3960            ;
3970 0406 8500   PRINT  STA POINTL
3980 0408 8601          STX POINTH
3990 040A A000          LDY #0
4000            ;
4010 040C B100   PRMSLP LDA (POINTL),Y
4020 040E F00A          BEQ RETPRM
4030 0410 8402          STY MEPNTL
4040 0412 200A02        JSR CO
4050 0415 A402          LDY MEPNTL
4060 0417 C8            INY
4070 0418 D0F2          BNE PRMSLP
4080            ;
4090 041A 60     RETPRM RTS
4100            ;
4110            ;
4120 041B A951   FRQMES LDA #FMESS
4130 041D A204          LDX #FMESS/256
4140 041F 4C0604        JMP PRINT
4150            ;
4160            ;
4170 0422 A966   PRADAP LDA #AMESS
4180 0424 A204          LDX #AMESS/256
4190 0426 4C0604        JMP PRINT
4200            ;
4210            ;
4220 0429 A99A   BVMES  LDA #BVMESS
4230 042B A204          LDX #BVMESS/256
4240 042D 4C0604        JMP PRINT
4250            ;
4260            ;
4270 0430 1B     MESS   .BYTE $1B,'1',$D
4270 0431 31
4270 0432 0D
4280 0433 20            .BYTE    == EC 65  CP/M system ==',$D,$A,$A
4280 0434 20
4280 0435 3D
4280 0436 3D
4280 0437 20
4280 0438 45
4280 0439 43
4280 043A 20
4280 043B 36
4280 043C 35
4280 043D 20
4280 043E 20
4280 043F 43
4280 0440 50
4280 0441 2F
4280 0442 4D
4280 0443 20
4280 0444 73                                4280 0449 20
4280 0445 79                                4280 044A 3D
4280 0446 73                                4280 044C 3D
4280 0447 74                                4280 044D 0D
4280 0448 65                                4280 044E 0A
4280 0449 6D                                4280 044F 0A
4290 0450 00            .BYTE 0
4300            ;
4310            ;
4320 0451 36     FMESS  .BYTE '6502 speed:
4320 0452 35
4320 0453 30
4320 0454 32
4320 0455 20
4320 0456 73
4320 0457 70
4320 0458 65
4320 0459 65
4320 045A 64
4320 045B 3A
4320 045C 20
4330 045D 78     FREQ   .BYTE 'x MHz',$D,$A,$A
4330 045E 20
4330 045F 4D
4330 0460 48
4330 0461 7A
4330 0462 0D
4330 0463 0A
4330 0464 0A
4340 0465 00            .BYTE 0
4350            ;
4360            ;
4370 0466 4E     AMESS  .BYTE 'No standard 6502 speed!
4370 0467 6F
4370 0468 20
4370 0469 73
4370 046A 74
4370 046B 61
4370 046C 6E                                4370 0475 32
4370 046D 64                                4370 0476 20
4370 046E 61                                4370 0477 73
4370 046F 72                                4370 0478 70
4370 0470 64                                4370 0479 65
4370 0471 20                                4370 047A 65
4370 0472 36                                4370 047B 64
4370 0473 35                                4370 047C 21
4370 0474 30                                4370 047D 20
4380 047E 31     FREQAD .BYTE '1 MHz delays installed!',$D,$A,$A,$A
4380 047F 20
4380 0480 4D
4380 0481 48                                4380 048E 74
4380 0482 7A                                4380 048F 61
4380 0483 20                                4380 0490 6C
4380 0484 64                                4380 0491 6C
4380 0485 65                                4380 0492 65
4380 0486 6C                                4380 0493 64
4380 0487 61                                4380 0494 21
4380 0488 79                                4380 0495 0D
4380 0489 73                                4380 0496 0A
4380 048A 20                                4380 0497 0A
4380 048B 69                                4380 0498 0A
4380 048C 6E
4380 048D 73
4390 0499 00            .BYTE 0
4400            ;
4410            ;
4420 049A 43     BVMESS .BYTE 'CBIOS65 vers 3.4',$D,$A
4420 049B 42
4420 049C 49
4420 049D 4F
4420 049E 53
4420 049F 36
4420 04A0 35
```

```
4420 04A1 20
4420 04A2 76
4420 04A3 65
4420 04A4 72
4420 04A5 73
4420 04A6 20
4420 04A7 33
4420 04A8 2E
4420 04A9 34
4420 04AA 0D
4420 04AB 0A
4430 04AC 00            .BYTE 0
4440            ;
4450            ;
4460 04AD 2018D8  MAINLP JSR PREADY  ; serve printer spooler
4470 04B0 2C01E3         BIT PAC     ; loop till PIA contains
4480 04B3 10F8           BPL MAINLP  ; new Z80 command
4490 04B5 AD00E3         LDA PAD     ; get it
4500 04B8 C900           CMP #0      ; console input
4510 04BA D009           BNE CONOUT
4520 04BC 200302         JSR CI
4530 04BF 20F003         JSR OUT
4540 04C2 4CAD04         JMP MAINLP
4550            ;
4560 04C5 C901    CONOUT CMP #1      ; console output
4570 04C7 D009           BNE LISTO
4580 04C9 204603         JSR IN
4590 04CC 200A02         JSR CO
4600 04CF 4CAD04         JMP MAINLP
4610            ;
4620 04D2 C902    LISTO  CMP #2      ; list output
4630 04D4 D009           BNE CONST
4640 04D6 204603         JSR IN
4650 04D9 201D02         JSR LIST
4660 04DC 4CAD04         JMP MAINLP
4670            ;
4680 04DF C903    CONST  CMP #3      ; console status
4690 04E1 D00A           BNE PUNCHO
4700 04E3 A508           LDA CIFLAG
4710 04E5 49FF           EOR #$FF
4720 04E7 20F003         JSR OUT
4730 04EA 4CAD04         JMP MAINLP
4740            ;
4750 04ED C904    PUNCHO CMP #4      ; punch output
4760 04EF D009           BNE READRI
4770 04F1 204603         JSR IN
4780 04F4 201702         JSR PUNCH
4790 04F7 4CAD04         JMP MAINLP
4800            ;
4810 04FA C905    READRI CMP #5      ; reader input
4820 04FC D005           BNE READDK
4830 04FE 201902         JSR READER
4840 0501 20F003         JSR OUT
4850 0504 4CAD04         JMP MAINLP
4860            ;
4870 0507 C906    READDK CMP #6      ; read sector
4880 0509 D018           BNE WRITDK
4890 050B 207603         JSR SET     ; load track into BLOCK
4900 050E D033           BNE ERROUT  ; error
4910 0510 A900           LDA #0      ; all right, track in
4920 0512 20F003         JSR OUT     ; buffer
4930 0515 A900           LDA #BLOCK  ; read data from BLOCK
4940 0517 85FE           STA MEMLO
4950 0519 A924           LDA #BLOCK/256
4960 051B 85FF           STA MEMHI
4970 051D 20E603         JSR MOVSEC  ; move sector to Z80
4980 0520 4CAD04         JMP MAINLP
4990            ;
5000 0523 C907    WRITDK CMP #7      ; write sector
5010 0525 D026           BNE HANCPM
5020 0527 207603         JSR SET     ; load track into BLOCK
5030 052A D017           BNE ERROUT  ; error
5040 052C A900           LDA #0      ; all right, track in
5050 052E 20F003         JSR OUT     ; buffer
5060 0531 A900           LDA #BLOCK  ; write data into BLOCK
5070 0533 85FE           STA MEMLO
5080 0535 A924           LDA #BLOCK/256
5090 0537 85FF           STA MEMHI
5100 0539 20F403         JSR RECSEC  ; receive sector from Z80
5110 053C A9FF           LDA #$FF    ; data not yet on disk
5120 053E 8504           STA WRFLAG
5130 0540 4CAD04         JMP MAINLP
5140            ;
5150            ;
5160 0543 A901    ERROUT LDA #1      ; read/write error
5170 0545 8505           STA TRFLAG  ; no valid data
5180 0547 20F003         JSR OUT
5190 054A 4CAD04         JMP MAINLP
5200            ;
5210 054D C908    HANCPM CMP #8      ; hand CP/M to Z80
5220 054F D02A           BNE CLRBUF
5230 0551 AD0B12         LDA CPMA+$808 ; get CP/M start addr
5240 0554 38             SEC
5250 0555 E908           SBC #8
5260 0557 20F003         JSR OUT
5270 055A A900           LDA #0
5280 055C 20F003         JSR OUT
5290 055F A900           LDA #CPMA
5300 0561 85FE           STA MEMLO
5310 0563 A90A           LDA #CPMA/256
5320 0565 85FF           STA MEMHI
5330 0567 A218           LDX #24     ; handle 24*256=6K bytes
5340 0569 A000           LDY #0
5350            ;
5360 056B B1FE    HANLPB LDA (MEMLO),Y
5370 056D 20F003         JSR OUT
5380 0570 C8             INY
5390 0571 D0F8           BNE HANLPB
5400 0573 E6FF           INC MEMHI
5410 0575 CA             DEX
5420 0576 D0F3           BNE HANLPB
5430 0578 4CAD04         JMP MAINLP
5440            ;
5450            ;
5460 057B C909    CLRBUF CMP #9      ; clear track buffer
5470 057D D017           BNE INIDSK
5480 057F A505           LDA TRFLAG
5490 0581 D009           BNE ENDCLR  ; if no valid data in BLOCK
5500 0583 A504           LDA WRFLAG
5510 0585 F005           BEQ ENDCLR  ; if data allready on disk
5520 0587 78             SEI
5530 0588 20ED08         JSR WRITE   ; put track on disk
5540 058B 58             CLI
5550            ;
5560 058C A200    ENDCLR LDX #0
5570 058E 8604           STX WRFLAG  ; track on disk,
5580 0590 E8             INX
5590 0591 8605           STX TRFLAG  ; track not valid
5600 0593 4CAD04         JMP MAINLP
5610            ;
5620            ;
5630 0596 C90A    INIDSK CMP #$A     ; init track on disk
5640 0598 D02F           BNE STODAT
5650 059A 204603         JSR IN      ; get drive nr
5660 059D 8506           STA IODRV
5670 059F 204603         JSR IN      ; get track nr
5680 05A2 8507           STA IOTRK
5690 05A4 A505           LDA TRFLAG  ; valid data in track buffer?
5700 05A6 F01D           BEQ INIERR  ; then error
5710 05A8 A506           LDA IODRV
5720 05AA C904           CMP #4      ; one of drives a..d?
5730 05AC B017           BCS INIERR  ; error if not
5740 05AE CD5C07         CMP DRIVE   ; drive allready selected?
5750 05B1 F005           BEQ INIDA
5760 05B3 203007         JSR SETDRV  ; select drive
5770 05B6 D00D           BNE INIERR
5780            ;
5790 05B8 206709  INIDA  JSR INITRK  ; initialize the track
5800 05BB D008           BNE INIERR
5810 05BD A900           LDA #0
5820 05BF 20F003         JSR OUT     ; o.k.: track initialized
5830 05C2 4CAD04         JMP MAINLP
5840            ;
5850 05C5 A901    INIERR LDA #1
5860 05C7 D0F5           BNE INIDA+7
5870            ;
5880 05C9 C90B    STODAT CMP #$B     ; store data in address
5890 05CB D015           BNE LODAT
5900 05CD 204603         JSR IN      ; get address upper part
5910 05D0 8DDE05         STA STORE+2 ; in code modification
5920 05D3 204603         JSR IN      ; address lower part
5930 05D6 8DDD05         STA STORE+1
```

```
5940 05D9 20460J        JSR IN      ; fetch data
5950 05DC 8DFFFF  STORE  STA $FFFF   ; store data in address
5960 05DF 4CAD04        JMP MAINLP
5970            ;
5980 05E2 C90C    LODAT  CMP #$C     ; load data from address
5990 05E4 D015           BNE EXIT    ; ERROR: leave CP/M system
6000 05E6 20460J         JSR IN      ; get upper address byte
6010 05E9 8DF405         STA LOAD+2  ; in code modification
6020 05EC 20460J         JSR IN      ; lower part of address
6030 05EF 8DF305         STA LOAD+1
6040 05F2 ADFFFF  LOAD   LDA $FFFF   ; load data
6050 05F5 204F0J         JSR OUT     ; and hand it to ZBO
6060 05F8 4CAD04         JMP MAINLP
6070            ;
6080            ;
6090 05FB 4C0B0J  EXIT   JMP ERROR   ; JMP via reset vector
6100            ;
6110            ;
6120            ;
6130            ; HEXBCD : convert HEX byte in acc into BCD
6140            ;
6150 05FE 38      HEXBCD SEC
6160 05FF A2FF           LDX #$FF
6170            ;
6180 0601 E8      HEXLP  INX
6190 0602 E90A           SBC #10
6200 0604 B0FB           BCS HEXLP
6210 0606 690A           ADC #10
6220 0608 8510           STA $10
6230 060A 8A             TXA
6240 060B 0A             ASL A
6250 060C 0A             ASL A
6260 060D 0A             ASL A
6270 060E 0A             ASL A
6280 060F 0510           ORA $10
6290 0611 60             RTS
6300            ;
6310            ;
6320            ; GETCHA : get character from keyboard buffer
6330            ;
6340 0612 201B0B  GETCHA JSR PREADY  ; serve printer spooler
6350 0615 A508           LDA CIFLAG  ; character in buffer?
6360 0617 D0F9           BNE GETCHA  ; if not then wait
6370 0619 AD0022         LDA KEYBUF  ; get character
6380 061C 8D6323         STA AHOLD   ; and save it
6390 061F A201           LDX #1
6400 0621 78             SEI
6410            ;                    ; and update buffer
6420 0622 BD0022  GETLP  LDA KEYBUF,X
6430 0625 9DFF21         STA KEYBUF-1,X
6440 0628 E8             INX
6450 0629 D0F7           BNE GETLP
6460 062B C609           DEC KEYPNT
6470 062D D004           BNE RETGET  ; if this was last character,
6480 062F A2FF           LDX #$FF    ; then set flag: buffer empty
6490 0631 860B           STX CIFLAG
6500            ;
6510 0633 58      RETGET CLI
6520 0634 60             RTS
6530            ;
6540            ;
6550            ; GETISR : interrupt service routine
6560            ;
6570 0635 48      GETISR PHA          ; save A, X
6580 0636 8A             TXA
6590 0637 48             PHA
6600 0638 AD00E1         LDA VAIFR   ; get the flag register
6610 063B 2902           AND #2      ; mask off PA1 bit (keyb)
6620 063D F015           BEQ NOGET   ; branch if no keyb IRQ
6630 063F AD01E1         LDA VAPAD   ; get data
6640 0642 29FF           AND #$7F
6650 0644 A609           LDX KEYPNT
6660 0646 E0FF           CPX #$FF    ; KEYBUF full?
6670 0648 F00A           BEQ NOGET   ; then forget character
6680 064A 9D0022         STA KEYBUF,X
6690 064D E8             INX
6700 064E 8609           STX KEYPNT
6710 0650 A200           LDX #0
6720 0652 860B           STX CIFLAG  ; update flag
6730            ;
6740 0654 AD00E1  NOGET  LDA VAIFR  ; get the flag reg. again
6750 0657 2940           AND #$40   ; mask off T1 bit
6760 0659 F017           BEQ RETISR ; branch if no timer IRQ
6770 065B AD04E1         LDA VATACL ; clear IRQ flag
6780 065E C60A           DEC COUNTR
6790 0660 D00B           BNE LDCNTR
6800 0662 A980           LDA #$80   ; unload drive head
6810 0664 0D02E0         ORA FDRB   ; set PB7
6820 0667 8D02E0         STA FDRB
6830 066A 4C7206         JMP RETISR
6840            ;
6850 066D A9FF    LDCNTR LDA #$FF   ; load the counter again
6860 066F 8D05E1         STA VATACH ; and start it
6870            ;
6880 0672 68      RETISR PLA
6890 0673 AA             TAX
6900 0674 68             PLA
6910 0675 58             CLI
6920 0676 40             RTI
6930            ;
6940            ;
6950            ;
6960            ; disk i/o routines
6970            ;
6980 06EF=        SETTK  = $6EF
6990 0730=        SETDRV = $730
7000 075C=        READ   = $75C
7010 083C=        WRITE  = $83C
7020 0963=        INITRK = $963
7030            ;
7040            ; routines for timing adjustment
7050            ;
7060 09C9=        COUNT  = $9C9
7070 09E0=        ADJTIM = $9E0
7080            ;
7090            ; extended VDU & spooler routines
7100            ;
7110 D80F=        EXTVDU = $D80F
7120 D812=        INSPLR = $D812
7130 D815=        INSCHR = $D815
7140 D818=        PREADY = $D818
```

```
  10            ; ****************************************
  20            ; *                                      *
  30            ; *  CP/M BIOS for EC 65 / 6502 part     *
  40            ; *       File Nr. 1                     *
  50            ; *                                      *
  60            ; *  Disk I/O Routines for OHIO kompatible *
  70            ; *           Floppy Interface Board     *
  80            ; *           using timer at $E100       *
  90            ; *           35/40/80 track version     *
 100            ; *           1/2 MHz auto select        *
 110            ; *           program version 3.4        *
 120            ; *           last revision date: 22.03.86 *
 130            ; *                                      *
 140            ; *  Written  1985/86 by  Frank Schmidt  *
 150            ; *                                      *
 160            ; ****************************************
 170            ;
 180            ;
 190 0680                 * = $6B0
 200            ;
 210            ;
 220            ; Addresses of the main program
 230            ;
 240 05FE=        HEXBCD = $5FE
 250            ;
 260            ;
 270            ; Floppy Interface Addresses
 280            ;
 290 E000=        FDRA   = $E000  ; data/data dir. reg A
 300 E001=        FCRA   = $E001  ; control register A
 310 E002=        FDRB   = $E002  ; data/data dir. reg B
 320 E003=        FCRB   = $E003  ; control register B
 330            ;
 340 E010=        CACIA  = $E010  ; ACIA control/status reg
 350 E011=        DACIA  = $E011  ; ACIA data register(s)
 360            ;
 370            ;
 380            ;
 390            ; page zero addresses
 400            ;
```

```
410 0000=       POINTL = $0000  ; universal pointer
420 0001=       POINTH = $0001
430 0002=       MEMTL = $0002
440 0003=       MEMTH = $0003
450 0004=       WRFLAG = $0004  ; 0: data allready on disk
460 0005=       TRFLAG = $0005  ; 0: data valid
470 0006=       IODRV  = $0006  ; next access drive number
480 0007=       IOTRK  = $0007  ; next access track number
490 0008=       CIFLAG = $0008  ; 0: data in keyboard buffer
500 0009=       KEYPNT = $0009  ; pointer for keybuf
510 000A=       COUNTR = $000A  ; counter for UNLDHD - timeout
520 00F9=       PAGES  = $00F9  ; nr of pages to be stored
530 00FA=       TRBUF  = $00FA  ; temporary buffer for track nr
540 00FB=       IOSTPS = $00FB  ; max I/Osteps within r/w attempt
550 00FD=       ATTMPT = $00FD  ; max number of attempts
560 00FE=       MEMLO  = $00FE  ; read/write memory pointer
570 00FF=       MEMHI  = $00FF
580             ;
590             ;
600             ; DOS Locations
610             ;
620 235E=       SECTOR = $235E  ; sector nuaber
630 235C=       DRIVE  = $235C  ; drive number
640 2370=       VECLO  = $2370  ; buffer for page 0 load vector
650 2371=       VECHI  = $2371
660 2350=       TRACK  = $2350  ; current track number
670             ;
680 0200=       START  = $0200  ; start of 6502 program
690             ;
700 0A00=       CPMA   = START+$0800 ; buffer for CP/M (6Kbytes)
710 2200=       KEYBUF = START+$2000 ; buffer for keyboard input
720 2400=       BLOCK  = START+$2200 ; buffer for track (5Kbytes)
730 3800=       RAMDSK = START+$3600 ; buffer for RAM - disk
740             ;
750             ;
760             ; Addresses of the bootstrap loader
770             ;
780 F50A=       DUMMY  = $F50A  ; a JSR needs 12 us @ 1 MHz
790 F4F6=       FDELAY = $F4F6  ; a JSR needs X*1248+14 us @ 1 MHz
800 F501=       RBYTE  = $F501  ; read one byte from disk
810             ;
820             ;
830             ; Timer addresses
840             ;
850 E104=       VATACL = $E104  ; timer 1 counter - low
860 E105=       VATACH = $E105  ; timer 1 counter - high
870 E10E=       VAIER  = $E10E  ; interrupt enable register
880             ;
890             ;
900             ;
910             ; HOME : put head on track 0
920             ;
930 0680 A954   HOME   LDA #84     ; do max 84 STPINs
940 0682 85FD          STA ATTMPT
950 0684 20CB06        JSR LDHEAD  ; do a head load
960 0687 20AE06        JSR STPOUT
970 068A 20C606        JSR STPDEL  ; wait & return with X=0
980 068D 86SD23        STX TRACK   ; update TRACK
990             ;
1000 0690 20A706 HOMELP JSR STPIN
1010 0693 A902          LDA #2      ; PA1 is TRO input
1020 0695 2C00E0        BIT FDRA    ; is it low?
1030 0698 F007          BEQ RETHOM  ; if, then return
1040 069A C6FD          DEC ATTMPT  ; was it last attempt?
1050 069C 10F2          BPL HOMELP  ; if not, then next one
1060 069E CE5D23        DEC TRACK   ; mark TRACK as not valid
1070            ;
1080 06A1 08    RETHOM PHP          ; save Z-flag
1090 06A2 20DE06        JSR UNLDHD  ; unload the head again
1100 06A5 28            PLP         ; restore Z-flag
1110 06A6 60            RTS         ; Z=1: ok   Z=0: error
1120            ;
1125            ;
1130            ; STPIN / STPOUT : move head one step in / out
1140            ;
1150 06A7 AD02E0  STPIN  LDA FDRB    ; set DIR bit
1160 06AA 0904           ORA #4
1170 06AC D005           BNE STPOUT+5 ; continue
1180            ;
1190 06AE AD02E0  STPOUT LDA FDRB    ; reset DIR bit
1200 06B1 29FE           AND #$FB
1210 06B3 8D02E0         STA FDRB
1220 06B6 200AF5         JSR DUMMY   ; delay 12 us @ 1 MHz
1230 06B9 29F7           AND #$F7    ; reset bit 3 = STEP
1240 06BB 8D02E0         STA FDRB
1250 06BE 200AF5         JSR DUMMY   ; delay 12 us @ 1 MHz
1260 06C1 0908           ORA #8      ; set STEP bit
1270 06C3 8D02E0         STA FDRB
1280            ;
1285            ;
1290            ; STPDEL : delay approx 10 ms
1300            ;
1310 06C6 A210   STPDEL LDX #16     ; (34)
1320 06C8 4CF6F4        JMP FDELAY
1330            ;
1340            ;
1350            ; LDHEAD : do a head-load
1360            ;
1370 06CB A9BF   LDHEAD LDA #%10111111
1380 06CD 2D0EE1        AND VAIER  ; disable timer IRQ
1390 06D0 8D0EE1        STA VAIER
1400 06D3 A97F          LDA #$7F   ; reset PB7
1410 06D5 2D02E0        AND FDRB
1420 06D8 8D02E0        STA FDRB
1430 06DB 4CC606        JMP STPDEL  ; wait & return
1440            ;
1450            ;
1460            ; UNLDHD : unload the head
1470            ;
1480 06DE A2FF   UNLDHD LDX #$FF
1490 06E0 A940          LDA #$40
1500 06E2 850A          STA COUNTR  ; wait for $40 timeouts
1510 06E4 0D0EE1        ORA VAIER   ; allow timer IRQ
1520 06E7 8D0EE1        STA VAIER
1530 06EA 8E05E1        STx VATACH  ; start timer
1540 06ED E8            INX         ; set Z flag
1550 06EE 60            RTS
1560            ;
1565            ;
1570            ; SETTK : set head on track in acc
1580            ;
1590 06EF AA     SETTK  TAY         ; save track number
1600 06F0 AD5C23        LDA DRIVE
1610 06F3 C904          CMP #4
1620 06F5 8A            TXA         ; restore track number
1630 06F6 9003          BCC SETTKO
1640 06F8 4C0D09        JMP FSETTK
1650            ;
1660 06FB F003   SETTKO BEQ HOME
1670 06FD 20FE05        JSR HEXBCD
1680 0700 85FA          STA TRBUF
1690 0702 C980          CMP #$80    ; track nr < 80?
1700 0704 B027          BCS RETSET  ; error exit
1710 0706 20CB06        JSR LDHEAD  ; load the head
1720            ;
1730 0709 A5FA   SETLP  LDA TRBUF   ; track reached?
1740 070B CD5D23        CMP TRACK
1750 070E F01A          BEQ RESET   ; unload head & ret
1760 0710 B007          BCS SETTKA  ; if track nr too big
1770 0712 20A706        JSR STPIN   ; then step in
1780 0715 A999          LDA #$99    ; = -1
1790 0717 D005          BNE ADJTRK  ; branch always
1800            ;
1810 0719 20AE06  SETTKA JSR STPOUT  ; else step out
1820 071C A901          LDA #1
1830            ;
1835            ;
1840 071E 18     ADJTRK CLC
1850 071F F8            SED
1860 0720 6D5D23        ADC TRACK
1870 0723 8D5D23        STA TRACk
1880 0726 D8            CLD
1890 0727 4C0907        JMP SETLP
1900            ;
1910 072A A900   RESET  LDA #0
1920 072C 60            RTS
1930            ;
1940 072D 0901   RETSET ORA #1      ; clear Z-bit
1950 072F 60            RTS         ; Z=1: ok   Z=0: error
1960            ;
1970            ;
1980            ; SETDRV : select drive
1990            ;
2000 0730 C905   SETDRV CMP #5      ; drivenr < 5 ?
```

```
2010 0732 B0F9          BCS RETSET  ; else error exit
2020 0734 CD5C23        CMP DRIVE   ; drive already selected?
2030 0737 F0F6          BEQ RETSET+2 ; return with Z=1
2040 0739 8D5C23        STA DRIVE   ; new drive nr
2050 073C C904          CMP #4
2060 073E 9003          BCC STDRVO
2070 0740 4C03D8        JMP FSTDRV
2080            ;
2090 0743 0A    STDRVO ASL A      ; X=0,2,4,6
2100 0744 AA            TAX
2110 0745 BD5407        LDA DRVTAB,X
2120 0748 8D00E0        STA FDRA
2130 074B BD5507        LDA DRVTAB+1,X
2140 074E 8D02E0        STA FDRB
2150 0751 4C8006        JMP HOME    ; put head on tr 0 & ret
2160            ;
2170 0754 40FF  DRVTAB .DBYTE $40FF
2180 0756 00FF          .DBYTE $00FF
2190 0758 40DF          .DBYTE $40DF
2200 075A 00DF          .DBYTE $00DF
2210            ;
2220            ;
2230            ; READ : read  memory into BUFFER
2240            ;
2250 075C AD5C23 READ   LDA DRIVE
2260 075F C904          CMP #4
2270 0761 9003          BCC READO
2280 0763 4C0608        JMP FREAD
2290            ;
2300 0766 A903  READO   LDA #3      ; max 3 attempts with STPIN/OUT
2310 0768 85FB          STA IOSTPS
2320 076A A907  READA   LDA #7      ; max 7 simple read attempts
2330 076C 85FD          STA ATTMPT
2340 076E A900  READB   LDA #BLOCK  ; store data start address
2350 0770 85FE          STA MEMLO
2360 0772 A924          LDA #BLOCK/256
2370 0774 85FF          STA MEMHI
2380 0776 AD5D23        LDA TRACK   ; read track zero?
2390 0779 D020          BNE READC
2400            ;
2410 077B A901          LDA #1      ; no STPIN/OUT
2420 077D 85FB          STA IOSTPS
2430 077F 20DA07        JSR TRKBEG  ; init ACIA, load head, wait in
2440 0782 202908        JSR RBYTET  ; read byte & test index
2450 0785 D03C          BNE NXREAD  ; error, next read attempt
2460 0787 8D7123        STA VECHI   ; save the load vector
2470 078A 202908        JSR RBYTET
2480 078D D034          BNE NXREAD
2490 078F 8D7023        STA VECLO
2500 0792 2001F5        JSR RBYTE   ; sector length
2510 0795 C908          CMP #8
2520 0797 D02A          BNE NXREAD  ; not the right format
2530 0799 F008          BEQ READD
2540            ;
2550 079B 20F207 READC  JSR RWBEG   ; load head, wait until index
2560                                ; pulse over and read header
2570 079E 201508        JSR HEADC   ; read sector header
2580 07A1 D020          BNE NXREAD
2590            ;
2600 07A3 A208  READD   LDX #8      ; there are 8 pages to load
2610 07A5 A000          LDY #0
2620            ;
2630 07A7 AD10E0 READLP LDA DACIA   ; receive data reg full?
2640 07AA 4A            LSR A
2650 07AB 90FA          BCC READLP  ; if not, then wait
2660 07AD AD11E0        LDA DACIA   ; else load data
2670 07B0 2C10E0        BIT CACIA   ; and test if parity error
2680 07B3 700E          BVS NXREAD  ; try again
2690 07B5 91FE          STA (MEMLO),Y ; store data in BUFFER
2700 07B7 C8            INY         ; next byte
2710 07B8 D0ED          BNE READLP
2720 07BA E6FF          INC MEMHI
2730 07BC CA            DEX         ; next page read
2740 07BD D0EB          BNE READLP
2750 07BF 20DE06        JSR UNLDHD  ; unload head & ret Z=1
2760 07C2 60            RTS         ; ready, ret with Z=1
2770            ;
2780 07C3 C6FD  NXREAD DEC ATTMPT
2790 07C5 D0A7          BNE READB
2800 07C7 C6FB          DEC IOSTPS
2810 07C9 F009          BEQ RDERR
2820 07CB 20A706        JSR STPIN
2830 07CE 20AE06        JSR STPOUT
2840 07D1 4C6A07        JMP READA
2850            ;
2860 07D4 20DE06 RDERR  JSR UNLDHD
2870            ;
2880 07D7 0901   ERR    ORA #1      ; ret with Z=0: error
2890 07D9 60            RTS
2900            ;
2910            ;
2920            ; TRKBEG : init ACIA, load head & wait for index pu
2930            ;
2940 07DA 20CB06 TRKBEG JSR LDHEAD
2950            ;
2960 07DD AD00E0 INDST  LDA FDRA    ; wait till index start
2970 07E0 30FB          BMI INDST
2980 07E2 AD00E0 INDEND LDA FDRA    ; wait till index end
2990 07E5 10FB          BPL INDEND
3000            ;
3010 07E7 A903  INACIA LDA #3       ; init the ACIA
3020 07E9 8D10E0        STA CACIA   ; master reset
3030 07EC A958          LDA #%01011000
3040 07EE 8D10E0        STA CACIA
3050 07F1 60            RTS
3060            ;
3070            ;
3080            ; RWBEG : start read/write access
3090            ;
3100 07F2 20DA07 RWBEG  JSR TRKBEG  ; init ACIA etc
3110 07F5 202908 HEADA  JSR RBYTET  ; read byte & test INDEX
3120 07F8 D01A          BNE RET
3130 07FA C943   HEADB  CMP #$43    ; track header 1st byte
3140 07FC D0F7          BNE HEADA
3150 07FE 202908        JSR RBYTET
3160 0801 D011          BNE RET
3170 0803 C957          CMP #$57    ; track header 2nd byte
3180 0805 D0F3          BNE HEADB
3190 0807 2001F5        JSR RBYTE
3200 080A CD5D23        CMP TRACK   ; 3rd byte
3210 080D D0E5          BNE HEADB
3220 080F 2001F5        JSR RBYTE
3230 0812 C958          CMP #$58    ; 4th byte
3240            ;
3250 0814 60     RET    RTS
3260            ;
3270 0815 202908 HEADC  JSR RBYTET
3280 0818 D0FA          BNE RET
3290 081A C976   HEADD  CMP #$76    ; sector header 1st byte
3300 081C 2001F5        JSR RBYTE
3310 081F C901          CMP #$01    ; 2nd byte (sector nr)
3320 0821 D0F7          BNE HEADD
3330 0823 2001F5        JSR RBYTE
3340 0826 C908          CMP #$8     ; 3rd byte (sector length)
3350 0828 60            RTS         ; all right, ret with Z=1
3360            ;
3370            ;
3380            ; RBYTET : read byte, error exit if INDEX=0
3390            ;
3400 0829 AD00E0 RBYTET LDA FDRA
3410 082C 10A9          BPL ERR     ; error if INDEX=0
3420 082E AD10E0        LDA CACIA   ; receive data reg full?
3430 0831 4A            LSR A
3440 0832 90F5          BCC RBYTET  ; wait, if not
3450 0834 AD11E0        LDA DACIA
3460 0837 2C3B08        BIT ZERO    ; set Z flag
3470 083A 60            RTS         ; Z=1: ok   Z=0: error
3480            ;
3490 083B 00     ZERO   .BYTE 0
3500            ;
3510            ;
3520            ; WRITE : write memory from buffer onto disk
3530            ;
3540 083C AD5C23 WRITE  LDA DRIVE
3550 083F C904          CMP #4
3560 0841 9003          BCC WRITEO
3570 0843 4C09D8        JMP FWRITE
3580            ;
3590 0846 A920   WRITEO LDA #%00100000
3600 0848 2C00E0        BIT FDRA    ; is track write protected?
3610 084B F08A          BEQ ERR
3620 084D A904          LDA #4      ; max 4 write attempts
3630 084F 85FB          STA IOSTPS
```

```
3640 0851 A902    WRITEA LDA #2      ; max 2 reread attempts
3650 0853 85FD           STA ATTMPT
3660              ;
3670 0855 A900    WRITEB LDA #BLOCK
3680 0857 85FE           STA MEMLO
3690 0859 A924           LDA #BLOCK/256
3700 085B 85FF           STA MEMHI
3710 085D A908           LDA #8
3720 085F 85F9           STA PAGES
3730 0861 AD5D23         LDA TRACK   ; write on track zero?
3740 0864 D052           BNE WRITEC
3750              ;
3760 0866 20C906         JSR LDHEAD
3770 0869 20E707         JSR INACIA  ; init the ACIA
3780 086C AD00E0  YINEND LDA FDRA
3790 086F 10FB           BPL YINEND  ; wait till index end
3800 0871 AD00E0  YINDST LDA FDRA
3810 0874 30FB           BMI YINDST  ; wait for index start
3820 0876 A9FE           LDA #$FE
3830 0878 2D02E0         AND FDRB    ; clear WRITE
3840 087B 8D02E0         STA FDRB
3850 087E AD00E0  YINDEX LDA FDRA
3860 0881 10FB           BPL YINDEX  ; wait till index end
3870 0883 A208           LDX #8      ; (16)
3880 0885 20F3F4         JSR FDELAY  ; delay 10 as
3890 0888 A202           LDX #2
3900 088A 205409         JSR WRBYTE  ; put load vector on disk
3910 088D A200           LDY #0
3920 088F 205409         JSR WRBYTE
3930 0892 A208           LDX #8      ; sector length
3940 0894 205409         JSR WRBYTE
3950 0897 A000           LDY #0
3960              ;
3970 0899 B1FE    YWRLOP LDA (MEMLO),Y
3980 089B AA             TAX
3990 089C 205409         JSR WRBYTE
4000 089F C8             INY
4010 08A0 D0F7           BNE YWRLOP
4020 08A2 E6FF           INC MEMHI
4030 08A4 C6F9           DEC PAGES
4040 08A6 D0F1           BNE YWRLOP
4050 08A8 AD00E0  ENDTRI LDA FDRA
4060 08AB 30FB           BMI ENDTRI
4070 08AD A901           LDA #1
4080 08AF 0D02E0         ORA FDRB    ; set WRITE
4090 08B2 8D02E0         STA FDRB
4100 08B5 4C6607         JMP READD   ; check track
4110              ;
4120 08B8 20F207  WRITEC JSR RWBEG   ; load head, wait until index
4130                                 ; pulse over and read header
4140 08BB A24E           LDX #78     ; (157) delay 400 us
4150 08BD 205F09         JSR DELX
4160 08C0 A9FE           LDA #$FE
4170 08C2 2D02E0         AND FDRB    ; clear WRITE
4180 08C5 8D02E0         STA FDRB
4190 08C8 A212           LDX #18     ; (37) delay 100 us
4200 08CA 205F09         JSR DELX
4210 08CD A900           LDA #BLOCK
4220 08CF 85FE           STA MEMLO
4230 08D1 A924           LDA #BLOCK/256
4240 08D3 85FF           STA MEMHI
4250 08D5 A908           LDA #8      ; there are 8 pages to be
4260 08D7 85F9           STA PAGES   ; written on disk
4270 08D9 A276           LDX #$76    ; sector header 1st byte
4280 08DB 205409         JSR WRBYTE
4290 08DE A201           LDX #1      ; 2nd byte (sector nr)
4300 08E0 205409         JSR WRBYTE
4310 08E3 A208           LDX #8      ; 3rd byte (sector length)
4320 08E5 205409         JSR WRBYTE
4330 08E8 A000           LDY #0
4340              ;
4350 08EA B1FE    WRITLP LDA (MEMLO),Y
4360 08EC AA             TAX
4370 08ED 205409         JSR WRBYTE
4380 08F0 C8             INY
4390 08F1 D0F7           BNE WRITLP
4400 08F3 E6FF           INC MEMHI
4410 08F5 C6F9           DEC PAGES
4420 08F7 D0F1           BNE WRITLP
4430 08F9 A247           LDX #$47    ; sector trailer 1st byte
4440 08FB 205409         JSR WRBYTE
4450 08FE A253           LDX #$53    ; sector trailer 2nd byte
4460 0900 205409         JSR WRBYTE
4470 0903 A212           LDX #18     ; (37) delay 100 us
4480 0905 205F09         JSR DELX
4490 0908 A901           LDA #1
4500 090A 0D02E0         ORA FDRB    ; set WRITE bit
4510 090D 8D02E0         STA FDRB
4520              ;
4530 0910 A924    CHECK  LDA #BLOCK/256
4540 0912 85FF           STA MEMHI
4550 0914 20F207         JSR RWBEG   ; read track header
4560 0917 201508         JSR HEADC   ; read sector header
4570 091A D021           BNE NXCHEK
4580 091C A208           LDX #8      ; there are 8 pages to check
4590 091E A000           LDY #0
4600              ;
4610 0920 AD10E0  CHEKLP LDA CACIA   ; receive data reg full?
4620 0923 4A             LSR A
4630 0924 90FA           BCC CHEKLP
4640 0926 AD11E0         LDA DACIA
4650 0929 2C10E0         BIT CACIA   ; parity error?
4660 092C 700F           BVS NXCHEK
4670 092E D1FE           CMP (MEMLO),Y
4680 0930 D00B           BNE NXCHEK
4690 0932 C8             INY
4700 0933 D0EB           BNE CHEKLP
4710 0935 E6FF           INC MEMHI
4720 0937 CA             DEX
4730 0938 D0E6           BNE CHEKLP
4740 093A 4CDE06         JMP UNLDHD  ; unload head & ret Z=1
4750              ;
4760 093D C6FD    NXCHEK DEC ATTMPT
4770 093F D0CF           BNE CHECK
4780 0941 C6FB           DEC IOSTFS
4790 0943 F009           BEQ WRERR
4800 0945 20A706         JSR STPIN
4810 0948 20AE06         JSR STPOUT
4820 094B 4C5108         JMP WRITEA
4830              ;
4840 094E 20DE06  WRERR  JSR UNLDHD
4850 0951 0901           ORA #1
4860 0953 60             RTS         ; Z=1: ok   Z=0: error
4870              ;
4880              ;
4890              ; WRBYTE : write byte in X on disk
4900              ;
4910 0954 AD10E0  WRBYTE LDA CACIA   ; transmit data reg empty?
4920 0957 4A             LSR A
4930 0958 4A             LSR A
4940 0959 90F9           BCC WRBYTE
4950 095B BE11E0         STX DACIA   ; write data
4960 095E 60             RTS
4970              ;
4980              ;
4990              ; DELX : a JSR needs 5*X+14 us @ 1 MHz
5000              ;
5010 095F CA      DELX   DEX
5020 0960 D0FD           BNE DELX
5030 0962 60             RTS
5040              ;
5050              ; INITRK : init one track on disk
5060              ;
5070 0963 AD5C23  INITRK LDA DRIVE   ; get drive number
5080 0966 C904           CMP #4
5090 0968 9003           BCC INITRA  ; then o.k.
5100 096A 4C0CD8         JMP FINTRK  ; else enter 1797 driver
5110              ;
5120 096D A507    INITRA LDA IOTRK
5130 096F 20EF06         JSR SETTK   ; set track
5140 0972 D053           BNE REINTR  ; error
5150 0974 A920           LDA #$20
5160 0976 2D00E0         AND FDRA    ; disk write protected?
5170 0979 4920           EOR #$20    ; complement bit
5180 097B D04A           BNE REINTR  ; error
5190 097D 78             SEI         ; disable IRQ
5200 097E 20DA07         JSR TRKBEG  ; init ACIA, load head
5210                                 ; & wait for index pulse
5220 0981 A9FE           LDA #$FE
5230 0983 2D02E0         AND FDRB    ; clear WRITE
5240 0986 8D02E0         STA FDRB
5250 0989 A208           LDX #8      ; (17)
```

```
5260 09BB 20F6F4        JSR FDELAY  ; delay approx. 10 ms
5270 09BE A247          LDX #$47    ; track header 1st byte
5280 09C0 205409        JSR WRBYTE
5290 09C3 A257          LDX #$57    ; track header 2nd byte
5300 09C5 205409        JSR WRBYTE
5310 09C8 AE5D23        LDX TRACK   ; track header 3rd byte
5320 09CB 205409        JSR WRBYTE
5330 09CE A258          LDX #$58    ; track header 4th byte
5340 09D0 205409        JSR WRBYTE
5350 09D3 A924          LDA #BLOCK/256
5360 09D5 8501          STA POINTH
5370 09D7 A900          LDA #BLOCK
5380 09D9 8500          STA POINTL
5390 09DB A000          LDY #0
5400 09DD A208          LDX #2048/256
5410 09DF A9E5          LDA #$E5
5420            ;
5430            ; Fill 2 K of BLOCK with $E5 and write it on disk
5440            ;
5450 09E1 9100   INTRLP STA (POINTL),Y
5460 09E3 C8            INY
5470 09E4 D0FB          BNE INTRLP
5480 09E6 E601          INC POINTH
5490 09E8 CA            DEX
5500 09E9 D0F6          BNE INTRLP
5510 09EB A901          LDA #1
5520 09ED 0D02E0        ORA FDRB
5530 09F0 8D02E0        STA FDRB
5540 09F3 203C08        JSR WRITE
5550 09F6 58            CLI         ; enable IRQ
5560            ;
5570 09F7 60     REINTR RTS
5580            ;
5590            ;
5600 09F8 00     CNTR   .BYTE 0
5610            ;
5615            ;
5620            ; COUNT : get time from index pulse to index pulse
5630            ;
5640 09F9 A000   COUNT  LDY #0      ; reset lowest counter
5650 09FB 20DA07        JSR TRKBEG  ; load head & wait index over
5660            ;
5670 09FE C8     COUNTA INY         ; increment the counter
5680 09FF D003          BNE COUNTB
5690 0901 EEC809        INC CNTR
5700            ;
5710 0904 AD00E0 COUNTB LDA FDRA    ; get floppy signals
5720 0907 30F5          BMI COUNTA
5730 0909 202E06        JSR UNLDHD
5740 090C ADC809        LDA CNTR    ; return counter
5750 090F 60            RTS
5760            ;
5770            ;
5780            ; ADJTIM : adjust timing for 2 MHz CPU
5790            ;
5800 0910 A920   ADJTIM LDA #32
5810 0912 8DC706        STA STPDEL+1
5820 0915 A910          LDA #16
5830 0917 8D8408        STA *INDEX+5
5840 091A A99D          LDA #157
5850 091C 8D8C08        STA WRITEC+4
5860 091F A925          LDA #37
5870 0921 8DC908        STA WRITEC+17
5880 0924 8D0409        STA WRITLP+26
5890 0927 A911          LDA #17
5900 0929 8D8A09        STA INITRA+29
5910 092C 60            RTS
5920            ;
5930            ; I/O Routines for 2nd Floppy Controller
5935            ;               and RAM Disk
5940            ;
5950 D800=      FGETTK = $D800
5960 D803=      FSTDRV = $D803
5970 D806=      FREAD  = $D806
5980 D809=      FWRITE = $D809
5990 D80C=      FINTRK = $D80C

tic ,01
```


kontakt-adressen

Bij de onderstaande computer-enthousiasten kunt u terecht voor het uitwisselen van informatie en vrije software op het gebied van de EC-computers (in het bijzonder de Octopus 65 met alle uitbreidingen):

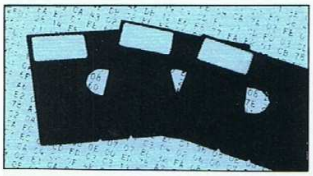
W. v. Dinther
Molenrijnselaan 27
5262 TN Vught
tel. 073-565401

W. Cuijpers
Mossellaan 63
4691 KE Tholen
tel. 01660-3566

M. Broek
Beekmansbos 10
1971 BV IJmuiden
tel. 02550-32169

T. v.d. Hombergh
Beukenstraat 20
5931 KG Tegelen
tel. 077-730151

A. Hovinga
Schoenmakerslaan 14
7875 BJ Exloo



CP/M-software voor de Octopus

Enkele interessante CP/M-programma's

Aangezien de Octopus zelf geen disk-formaten van andere CP/M-computers kan lezen, moet men een andere weg zoeken om programma's te kunnen overbrengen. We hebben in een van de vorige artikelen al de mogelijkheid genoemd om dat serieel te doen. Om de nieuwe CP/M-Octopus-bezitters een beetje op gang te helpen, zijn enkele handige CP/M-programma's op Ohio-disk-formaat gezet. Deze willen we in dit artikel graag aan u voorstellen.

Alle hier voorgestelde programma's komen uit het public-domain-bereik. Ze werden door computergebruikers uit de hele wereld ontwikkeld en door de SIG/M-user-group samengebracht. De interessantste programma's van deze groep hebben we voor u uitgekozen.

MONITOR

De monitor is het enige programma dat niet in de vorm van een programma op schijf staat. Het zit namelijk in de EPROM op de Z80-kaart. Het programma wordt gestart met het ZCPR2-kommando "JUMP F000H". Nadat dit is gebeurd, kan men een overzicht van alle beschikbare kommando's opvragen door middel van "T <CR>". Figuur 1 laat zien wat dan op het scherm verschijnt. Zo geeft het kommando "DF000 F7FF <CR>" bijvoorbeeld een hexdump op het beeldscherm van het opgegeven geheugenbereik.

Met Ctrl-C keert men weer terug naar ZCPR2-kommando-nivo. Dat veroorzaakt, zoals gewoonlijk, een warm-start. U kunt ook het kommando "B <CR>" geven. Dan gebeurt hetzelfde, maar u keert nu terug in de ZCPR2-Kernel zonder warm-start.

Enige voorzichtigheid is geboden bij het gebruik van de kommando's voor de port-input/output. Daar het hele input- en output-gebeuren bij de Octopus geschiedt via "slechts" twee ports, kan een schrijf- of leeskommando op het verkeerde moment fatale gevolgen hebben voor het komplete systeem. De hele computer kan daarmee plat worden gelegd.

Het monitorprogramma gebruikt overigens geen BDOS- en andere CP/M-funkties. Het daarom misschien verstandig om nog even stil te staan bij de werking van de diverse kontroletekens.

Ctrl-S onderbreekt, evenals bij CP/M, de weergave van een listing op het scherm. Nogmaals Ctrl-S doet weergave weer verder lopen.

Ctrl-X stopt de weergave op het scherm. Het programma keert nu direkt terug naar de kommando-mode. Met Ctrl-X beëindigt men ook de invoer van ASCII-tekens bij de uitvoering van het "AL"-kommando. **Ctrl-H** (backspace) en **Rubout** doen beide hetzelfde. Ze wissen het laatst ingevoerde karakter.

Een uitgebreide dokumentatie van de monitor staat op CP/M-disk nummer 2 (EC-65-disk nr. 41). De informatie op deze schijf krijgt u op het scherm door de disk in drive B te stoppen en vervolgens in te typen "TYPE B:MONITOR.DOC".

1

```
->T

ASCII LOAD, DUMP -->A[ L  D  S ] [1ST ADR] [2ND ADR]
      or SEARCH
BACK GENTLY      -->B
CALL SUBROUTINE  -->C[ADR]
DUMP MEMORY      -->D[1ST ADR] [2ND ADR]
FILL MEMORY      -->F[1ST ADR] [2ND ADR] [HEX BYTE]
                 -->F[1ST ADR] [2ND ADR] '[ASCII]
GO ANYWHERE      -->G[ADR]
HEX MATH + -     -->H[1ST HEX] [2ND HEX]
INPUT FROM PORT  -->I[PORT NUMBER]
LOAD MEMORY      -->L[START ADR]
                   ADR BYTE [NEW BYTE]   ^X HALT
MOVE MEMORY      -->M[1ST ADR] [2ND ADR] [NEW START]
OUTPUT TO PORT   -->O[PORT NUMBER] [BYTE]
PRINTER TOGGLE   -->P
REPLACE MEMORY   -->R[1ST ADR] [2ND ADR] [TARGET] [BYTE]
SEARCH FOR BYTE  -->S[1ST ADR] [2ND ADR] [BYTE1] [BYTE2]
COMMAND TABLE    -->T
COMPARE BLOCKS   -->V[1ST ADR] [2ND ADR] [1ST ADR BLOCK2]
DISP STACK POINT -->X


->                                          86263-1
```

Figuur 1. Al deze kommando's kent de monitor die in de EPROM van de Z80-kaart aanwezig is. Deze lijst kan men te allen tijde opvragen met het kommando "T".

Figuur 2. Dit zijn alle kommando's van de 24 Kbyte lange Forth-compiler. Hij bevat zelfs een editor, een assembler, een debugger, een decompiler en een multitasker.

Forth

Alweer, denkt u? Forth bestond toch al lang voor de Octopus-65! Dat is zo, maar deze nieuwe versie is wel iets heel bijzonders. Figuur 2 toont u alle beschikbare kommando's voor deze 24 Kbyte (!) lange compiler. Heel omvangrijk, nietwaar? De compiler bevat onder andere een eigen editor, een komplete 8080-assembler, een Forth-run-time-debugger, een decompiler, een multitasker en een op wens toevoegbare BASIC-compiler — een vrij kort Forth-programma overigens, dat op indrukwekkende wijze de mogelijkheden van dit systeem demonstreert. Figuur 3 laat zo'n BASIC-programma zien dat ook op de Forth-diskette staat.

Voor een eerste kennismaking is het voldoende als u de Forth-diskette in drive B stopt en op het keyboard intikt: "B:F83". Nadat Forth is geladen, meldt het zich op het scherm en wacht op een reaktie van uw kant. Wie al een beetje op de hoogte is met deze programmeertaal, zal waarschijnlijk meteen "VLIST" intypen en ... krijgt meteen een foutmelding. De compiler kent dit kommando namelijk niet. Hier moet men het kommando "WORDS" geven om een overzicht van alle beschikbare kommando's te krijgen. Deze compiler is namelijk opgezet volgens de Forth-83-standaard en niet volgens die van Fig-Forth. U kunt nu nog wat spelen met Forth en daarna terugkeren naar het operating-system met het kommando "Bye".

Wilt u wat dieper in deze Forth duiken, dan moet u eerst nog wat werk verzetten. De meeste files op de schijf zijn namelijk in hun huidige vorm nog niet "gebruiksklaar". Ze staan op de diskette in een "gekomprimeerd" formaat. Alleen op deze manier was het mogelijk om alle bytes op de schijf te zetten. De files kunnen weer in hun originele vorm worden gebracht met behulp van de utility "USQ.COM". Dit gaat als volgt:

— Forth-diskette in drive B steken.
— Met het kommando "E: <CR>" de RAM-disk selekteren.
— "DIR B: <CR>" intypen, waarna de directory van drive B verschijnt.

2

```
EMPTY   MARK   HELLO   BACKGROUND:   ACTIVATE   SET-TASK   TASK:
RESUME   DEBUG   LISTING   SHOW   (SEMIT)   (PAGE)   FORM-FEED   PAGE
#PAGE   LOGO   L/PAGE   FOOTING   INIT-PR   EPSON   SEE   (SEE)
ASSOCIATIVE:   CASE:   MAP   OUT   DL   DU   DUMP   .HEAD   ?.A   ?.N
DLN   EMIT.   D.2   .2   A   SHADOW   (WHERE)   FIX   EDIT   ED   DONE
EDITOR   DARK   AT   -LINE   BLOT   REPLACE   INSERT   DELETE   SEARCH
SCAN-1ST   FOUND   TO   CONVEY   (CONVEY)   .TO   HOP   CONVEY-COPY
U/D   HOPPED   VIEW   §VIEW   COPY   (COPY)   ESTABLISH   L   B   N
::   MANY   TIMES   #TIMES   WORDS   LARGEST   IND   INDEX   .LINE0
TRIAD   LIST   .SCR   ?CR   ?LINE   RMARGIN   LMARGIN   HIDDEN   0<=
0>=   >=   <=   U>=   U<=   MS   FUDGE   PC!   PC§   MULTI   SINGLE
STOP   WAKE   SLEEP   !LINK   §LINK   LOCAL   RESTART   (PAUSE)
UNBUG   BUG   DOES?   DOES-SIZE   DOES-OP   LABEL   UTILITY.BLK
CPU8080.BLK   EXTEND80.BLK   KERNEL80.BLK   VIEWS   VIEW-FILES
SAVE-SYSTEM   FROM   OPEN   DEFINE   B:   A:   DRIVE?   DIR
CREATE-FILE   MORE   ROOT   -->   +THRU   THRU   ?ENOUGH   ?   (S   Ö
L/SCR   C/L   RECURSE   DUMP   .ID   .S   DEPTH   BYE   START   OK
INITIAL   COLD   WARM   BOOT   QUIT   RUN   IS   (IS)   >IS   USER
#USER   CODE   AVOC   2VARIABLE   2CONSTANT   DEFINITIONS   VOCABULARY
DEFER   VARIABLE   CONSTANT   RECURSIVE   ;   :   Ü   Ä   DOES>
;CODE   (;CODE)   ;USES   ASSEMBLER   (;USES)   REVEAL   HIDE   ?CSP
!CSP   CREATE   "CREATE   ,VIEW   WHILE   ELSE   IF   REPEAT   AGAIN
UNTIL   +LOOP   LOOP   ?DO   DO   THEN   BEGIN   ?LEAVE   LEAVE
?<RESOLVE   ?<MARK   ?>RESOLVE   ?>MARK   <RESOLVE   <MARK   >RESOLVE
>MARK   ?CONDITION   ABORT   ABORT"   (ABORT")   (?ERROR)   ?ERROR
WHERE   FORGET   (FORGET)   TRIM   FENCE   "   ."   ,"   (.")   (")
ÄCOMPILEÜ   Ä'Ü   '   ?MISSING   CRASH   CONTROL   ASCII   DLITERAL
LITERAL   IMMEDIATE   COMPILE   EVEN   ALIGN   C,   ,   ALLOT
INTERPRET   STATUS   ?STACK   DEFINED   ?UPPERCASE   FIND   #THREADS
(FIND)   HASH   VIEW?   >VIEW   ·>LINK   >NAME   >BODY   LINK>   NAME>
BODY>   L>NAME   N>LINK   FORTH-83   DONE?   TRAVERSE   ÖS   (   .(
>TYPE   WORD   'WORD   PARSE   PARSE-WORD   SOURCE   (SOURCE)   PLACE
/STRING   SCAN   SKIP   D.R   D.   (D.)   UD.R   UD.   (UD.)   .R   .
(.)   U.R   U.   (U.)   OCTAL   DECIMAL   HEX   #S   #   SIGN   #>
<#   HOLD   NUMBER   (NUMBER)   NUMBER?   (NUMBER?)   CONVERT
DOUBLE?   DIGIT   LOAD   (LOAD)   DEFAULT   VIEW#   FLUSH
SAVE-BUFFERS   EMPTY-BUFFERS   IN-BLOCK   BLOCK   (BLOCK)   BUFFER
(BUFFER)   MISSING   DISCARD   UPDATE   ABSENT?   LATEST?   CAPACITY
DOS   SWITCH   FILE?   .FILE   WRITE-BLOCK   READ-BLOCK   >UPDATE
BUFFER#   >END   >BUFFERS   INIT-RO   FIRST   >SIZE   LIMIT
DISK-ERROR   B/FCB   REC/BLK   B/REC   B/BUF   #BUFFERS   QUERY   TIB
EXPECT   CC-FORTH   CC   DEL-IN   CHAR   (CHAR)   CR-IN   P-IN
RES-IN   BACK-UP   (DEL-IN)   BS-IN   BEEP   BACKSPACES   SPACES
SPACE   TYPE   CRLF   (EMIT)   (PRINT)   PR-STAT   CR   KEY   KEY?
(CONSOLE)   (KEY)   (KEY?)   BIOS   BDOS   COMPARE   CAPS-COMP   COMP
-TRAILING   PAD   HERE   UPPER   UPC   MOVE   LENGTH   COUNT   BLANK
ERASE   FILL   CAPS   BELL   BS   BL   END?   #TIB   SPAN   >IN   BLK
VOC-LINK   WIDTH   'TIB   CONTEXT   #VOCS   CURRENT   CSP   LAST   R#
DPL   WARNING   STATE   PRIOR   SCR   EMIT   PRINTING   IN-FILE   FILE
HLD   BASE   OFFSET   #LINE   #OUT   DP   RP0   SP0   LINK   ENTRY
TOS   */   */MOD   MOD   /   /MOD   *   MU/MOD   M/MOD   *D   DMAX
DMIN   D>   D<   DU<   D=   D0=   ?DNEGATE   D-   D2/   D2*   DABS
S>D   DNEGATE   D+   2ROT   4DUP   3DUP   2OVER   2SWAP   2DUP   2DROP
2!   2§   WITHIN   BETWEEN   MAX   MIN   >   <   U>   U<   ?NEGATE
<>   =   0<>   0>   0<   0=   UM/MOD   U*D   UM*   2-   1-   2+   1+
8*   U2/   2/   2*   3   2   1   0   +!   ABS   -   NEGATE   +   OFF
ON   CTOGGLE   CRESET   CSET   FALSE   TRUE   NOT   XOR   OR   AND
ROLL   PICK   R§   >R   R>   ?DUP   FLIP   -ROT   ROT   NIP   TUCK
OVER   SWAP   DUP   DROP   RP!   RP§   SP!   SP§   CMOVE>   CMOVE   C!
C§   !   §   (?LEAVE)   (LEAVE)   J   I   PAUSE   NOOP   GO   PERFORM
EXECUTE   >NEXT   BOUNDS   (?DO)   (DO)   (+LOOP)   (LOOP)   ?BRANCH
BRANCH   (LIT)   UP   UNNEST   EXIT   RP   FORTH
```

3

```
F>secrtary <CR>

CP/M SECRTARY REL 9.76    820320A
USED: 0  AVAILABLE: 36775
CONFIGURE (Y/N)? Y

TYPE CHAR USED FOR BACKSPACE <BS>
STOP AT END OF EACH PAGE (Y/N)? N
LINES PER SCREEN OR ZERO? 24 <CR>
INPUT LINE LENGTH? 80 <CR>
LINE WRAP AROUND ON INPUT (Y/N)? Y
UNDERLINE USING BS OR CR (B/C)? C


IF YOU WANT TO SAVE THIS CONFIGURATION,
"QUIT" AND ENTER "SAVE 48 SECRTARY.COM"


CP/M SECRTARY REL 9.76    820320A
USED: 0  AVAILABLE: 36775
#
```

```
Scr # 10          BASIC.BLK
 0 ( basic: branching demo )
 1 INTEGER J   INTEGER K
 2
 3 : RUN   BASIC
 4 10 FOR K = 1 TO 15 STEP 3
 5 15 LET J = J + K
 6 20 IF K >= 8 THEN 35
 7 25 PRINT K
 8 30 GOTO 40
 9 35 PRINT K , J , " SUM "
10 40 NEXT K
11 50 PRINT " DONE  "
12 80 END
13
14 RUN
15
```

86263-3

4

```
'65-CP/M.003   0k  --  Octopus-65 CP/M-disk-nummer
BASIC   .BQK   5K  --  in Forth geschreven BASIC-compiler
CPU8080 .BQK  10k  --  assembler, run-time-debugger, multitasker (D)
EXTEND80.BQK   9k  --  Forth-CP/M-interface (D)
F83     .COM  24k  --  Forth-83-compiler
F83-FIXS.TQT   4k  --  updates voor oude versie (D)
HUFFMAN .BQK  13k  --  Forth-programma voor het komprimeren van files
                       niet kompatibel met USQ / SHOW
META80  .BQK  13k  --  Forth-metacompiler (D)
README  .BQK  13k  --  beknopte Forth-handleiding (D)
SHOW    .COM   3k  --  utility voor het lezen van gekomprimeerde files
USQ     .COM   2k  --  vertaalt gekomprimeerde files naar originele vorm
UTILITY .BQK  37k  --  diverse Forth-utilities (D)

12 files occupy  133K bytes
```

86263-4

5

```
F>secrtary <CR>

CP/M SECRTARY REL 9.76    820320A
USED: 0  AVAILABLE: 36775
CONFIGURE (Y/N)? Y

TYPE CHAR USED FOR BACKSPACE <BS>
STOP AT END OF EACH PAGE (Y/N)? N
LINES PER SCREEN OR ZERO? 24 <CR>
INPUT LINE LENGTH? 80 <CR>
LINE WRAP AROUND ON INPUT (Y/N)? Y
UNDERLINE USING BS OR CR (B/C)? C


IF YOU WANT TO SAVE THIS CONFIGURATION,
"QUIT" AND ENTER "SAVE 48 SECRTARY.COM"


CP/M SECRTARY REL 9.76    820320A
USED: 0  AVAILABLE: 36775
#
```

86263-5

Figuur 3. Zo ziet een programma er uit voor de in Forth geschreven BASIC-compiler. De compiler wordt geladen vanuit Forth met de kommando's "OPEN BASIC.BLK <CR>" en "OK <CR>".

Figuur 4. Deze files staan op de Forth-diskette. De files met de extensions ".TQT" en ".BQK" staan in een gekomprimeerde vorm op de schijf. Deze moeten vóór gebruik eerst weer in hun normale vorm worden teruggebracht met behulp van de utility "USQ.COM". Een (D) achter de file-omschrijving geeft aan dat de desbetreffende file alleen dokumentatiemateriaal bevat.

Figuur 5. De initialiseringsprocedure voor de tekstverwerker "SECRETARY". In dit voorbeeld wordt gewerkt met kettingpapier. Wil men losse bladen gebruiken, dan moet de vraag "STOP AT ... ?" met "Y" beantwoord worden.

— Zoek de files uit die weer in hun originele vorm moeten worden "vertaald". Alle gekomprimeerde files bevatten een "Q" in de extension van de file-naam (bijv. BASIC.BQK).
— Met het kommando "B:USQ B:fn.ext <CR>" worden de files met de opgegeven naam vertaald (hierbij zijn de wildcards "*" en "?" ook toegestaan). De vertaalde files worden in de RAM-disk opgeslagen.
— De zo "geëxpandeerde" files kunnen op een "echte" diskette worden gezet, zodat men ze voor toekomstig gebruik onder handbereik heeft.

Met files met de extension ".BLK" kunt u alleen vanuit Forth werken. Ze bevatten de verschillende "screens" via welke Forth de handling van de massageheugens verzorgt.
U kunt Forth ook starten met "F83E:BASIC.BLK". Dan heeft u toegang tot alle screens die onder deze verzamelnaam in de RAM-disk zijn opgeslagen. Met "1 LIST" kunt u de inhoud van het eerste screen op het scherm krijgen, met "1 7 INDEX" krijgt u de titels van de screens 1...7. Op deze wijze kan men praktisch de hele source van Forth en een uitgebreide dokumentatie opvragen. Figuur 4 geeft een korte omschrijving van de op de diskette aanwezige files.

SECRETARY

De Nederlandse vertaling van dit woord is zeker niet moeilijk: secretaris of secretaresse. Toch dekt dat woord niet helemaal de omvang van het zogeheten programma. We hebben hier te maken met een vrij krachtige tekstverwerker op basis van een (helaas) regelgeoriënteerde editor. Het programma kan worden gebruikt voor het schrijven van gewone teksten en voor het ontwikkelen van source-teksten. Ondanks de regelgeoriënteerde opzet is het programma heel bruikbaar. Alle denkbare features zijn aanwezig. We zullen de belangrijkste ervan opsommen:
— Automatische (en instelbare) word-wrap, die plaats vindt tijdens de weergave van de tekst op het scherm.
— Tussenvoegen van kommentaren die later niet hoeven te worden geprint.
— Automatisch formatteren van de tekst: titel, paginanummer, paginalengte, marges...
— Tussenvoegen van data uit speciaal daarvoor opgezette files (bijvoorbeeld adresbestanden).

Dat was slechts een selektie uit de vele mogelijkheden.
Op de diskette staat ook een komplete, zeer uitgebreide handleiding (in het Engels). Als u in het bezit bent van een printer, kunt u de 38 pagina's van deze handleiding op de volgende wijze op papier zetten:
Stop de SECRETARY-diskette in drive B en start het programma met "SECRTARY", gevolgd door RETURN (de schrijfwijze is juist; de E ontbreekt hier omdat filenamen bij CP/M slechts acht karakters lang mogen zijn). Nu volgt een korte initialiseringsdialoog, zoals in figuur 5 is afgebeeld. Zet uw printer startklaar en schakel de output met behulp van het kommando "DEVICE L" om naar de printer (is geen

Figuur 6. Voor de Z80-Octopus zijn in totaal zes verschillende diskettes gemaakt. Elke diskette bevat circa 150 Kbytes programma's en dokumentatie (bij 80-track-schijven). Voor Octopus-bezitters die nog met 40-track-drives werken, is de inhoud van die zes schijven verdeeld over 11 40-track-diskettes. Elke diskette bevat ongeveer 70 Kbytes.

6

```
80-track-diskettes (enkelzijdig)
--------------------------------

disk 40:   ZCPR2-systeem-disk, utilities
disk 41:   ROM-source, ZCPR2-source (1), utilities
disk 42:   F83 met source en dokumentatie, zonder KERNEL.BLK
disk 43:   XLISP met source en dokumentatie
disk 44:   SECRETARY met dokumentatie, zonder source
disk 45:   SECRETARY-source, ZCPR2-source (2) en KERNEL.BLK


40-track-diskettes (enkelzijdig)
--------------------------------

disk 46:   ZCPR2-systeem-disk
disk 47:   utilities (1)
disk 48:   utilities (2)
disk 49:   ROM-source, ZCPR2-source (1)
disk 50:   F83 met dokumentatie, zonder HUFFMAN.BLK
disk 51:   F83-source, zonder KERNEL.BLK
disk 52:   KERNEL.BLK
disk 53:   XLISP met dokumentatie
disk 54:   SECRETARY met dokumentatie (1)
disk 55:   SECRETARY-dokumentatie (2)
disk 56:   SECRETARY-source en ZCPR2-source (2)
```

printer aanwezig, dan wordt dat laatste natuurlijk niet ingetypt, de tekst verschijnt dan op het beeldscherm). Laad vervolgens met "LOAD USERMAN0.TXT" het eerste gedeelte van de dokumentatie (de directory) en stuur dit naar de printer met het kommando "PRINT". Daarna doet u hetzelfde met "USERMAN1.TXT". Deze file bevat de eigenlijke dokumentatie. Nadat de printer alles op papier heeft gezet, komt het laatste stukje (nog drie pagina's). Dat draagt de naam "USERMAN5.TXT". De delen 2, 3 en 4 werden na het eerste gedeelte automatisch ingelezen en naar de printer gestuurd.

De tekstverwerker kan worden verlaten met het kommando "QUIT". Er is ook een "HELP"-funktie, maar die biedt alleen echte hulp als u het programma al vrij goed kent en eens een kommandonaam vergeten bent. Na het intypen van "HELP" verschijnt namelijk alleen een overzicht van alle beschikbare kommando's, zonder verder kommentaar.

Het laatste deel van de dokumentatie neemt een bijzondere plaats in. Het programma zelf werd namelijk helemaal niet geschreven voor CP/M en praktisch de hele dokumentatie heeft betrekking op deze "oude" versie. Het laatste stuk van de dokumentatie gaat uitsluitend over de CP/M-versie van SECRETARY. Als dus iets niet funktioneert zoals het volgens de "oude" dokumentatie zou horen, dan vindt u hiervoor in het laatste gedeelte zeker een verklaring.

XLISP

Dit gedeelte is interessant voor ieder die zich wel eens wil bezig houden met het thema kunstmatige intelligentie. De auteur van XLISP — dat overigens in "C" werd geschreven — heeft voor deze interpreter de computertaal LISP als voorbeeld genomen. Het gaat hier wel om een betrekkelijk vrije implementatie. De dokumentatie hierbij is zeer gedetailleerd, maar komplex. Het is aan te bevelen om eerst eens wat te werken met LISP, zodat men een beetje vertrouwd raakt met deze taal die kwa opzet sterk afwijkt van de andere bekende programmeertalen.

De dokumentatie kunt u oproepen met het kommando "TYPE XLISP.DOC" of "LIST XLIST.DOC". In het laatste geval wordt de output naar de printer gestuurd. Aangezien de files in gekomprimeerde toestand op de diskette staan, moet men ze eerst nog bewerken met de utility "USQ.COM".

De interpreter wordt gestart met het kommando "XLISP". Wil men tevens een XLISP-programma starten, dan luidt het kommando "XLISP fn.ext". Bij de extension ".LSP" hoeft men deze niet in te typen.

LISP-source-files kunnnen ook worden gemaakt met SECRETARY. Voordat u het op deze wijze gemaakte programma laat "vertalen", moet het eerst worden weggeschreven met het kommando "BSAVE". Dus niet zoals bij gewone tekstfiles met "SAVE". Bovendien moeten de regelnummers uit de programma's worden verwijderd. Dat gaat heel eenvoudig. Na het geven van het kommando "BSAVE" vraagt de computer u namelijk of de regelnummers moeten worden verwijderd. U hoeft op deze vraag alleen maar te antwoorden met "Y" (yes).

Ook de komplete source van de interpreter staat op schijf (de file "XLISP.LQR"), maar wel in een dubbel gekodeerde vorm. Eerst moet "USQ.COM" hierop worden losgelaten, zodat de library-file "XLISP.LBR" ontstaat. Vervolgens kan de utility "LU.COM" worden gebruikt om de gewenste files hieruit op te halen.

In figuur 6 is een overzicht gegeven van alle diskettes die betrekking hebben op het Octopus-Z80-systeem. We willen hierbij nog de opmerking plaatsen dat de meeste programma's ook op 40-track-schijven verkrijgbaar zijn, maar dat het werken met CP/M op zo'n "krap" formaat vrij moeizaam verloopt. Daarom adviseren we u op deze plaats nogmaals om de Z80-kaart meteen bij de bouw te voorzien van 256-Kbit-DRAM's, zodat een RAM-disk ter beschikking staat. Dat werkt veel prettiger.

Aan het einde van dit artikel willen we nog een opmerking maken over de verkrijgbaarheid van de in dit artikel genoemde schijven. In verband met licentierechten en het feit dat sommige programma's van Duitse origine zijn, kan het voorkomen dat niet al deze schijven in Nederland leverbaar zijn. Mocht u geen adres hiervoor kunnen vinden, dan kunt u kontakt opnemen met de redaktie. Deze heeft in elk geval adressen in West-Duitsland, waar u deze schijven zonder problemen kunt bestellen.