All Elektuur Junior articles 1984

2-75 Basicode-2 voor Junior met VDU-kaart

2-66 6502-tracer

3-40 get&co Tapemonitor uitbreiding

4-53 ID-list

4-71 motorschakeling voor floppy drives

7-68 2716 voor 2708

8-12 start-omleiding voor 6502

8-15 beeldruiskiller

9-72 DOS-uitbreidingen

11-33 de 6845 geprogrammeerd

Na de komst van de nieuwe Elektuur VDU-kaart kon het natuurlijk niet uitblijven dat ook een Basicode-2 werd gemaakt voor de kombinatie Junior computer en VDU-kaart. Oplettende lezers wisten natuurlijk al dat er een Basicode-2 voor deze kombinatie kwam, want zoals in het novembernummer aangekondigd was, werd op 6 november jongstleden deze al via het radioprogramma "Hobbyscoop" de ether ingestraald. Nu krijgt u het nog eens zwart op wit.

Basicode-2 voor Junior met VDU-kaart

Voor een beschrijving van de Basicode-2 en in het bijzonder de Basicode-2 voor de Junior verwijzen we naar de twee artikelen die daarover in het oktobernummer zijn gepubliceerd. Alle wetenswaardigheden voor het gebruik van Basicode-2 met de Junior, plus de benodigde vertaalprogramma's en interface-schakeling zijn daar te vinden, dus dat wordt op deze plaats niet meer herhaald.
Het enige dat aangepast moet worden om Basicode-2 bij de Junior met VDU-kaart te kunnen gebruiken, zijn de standaard-subroutines. In dit artikel vindt u dan ook twee tabellen met deze subroutines: een tabel voor de uitgebreide Junior met VDU-kaart en een tabel voor de DOS-Junior met VDU-kaart.

Enkele veranderingen

Ten opzichte van de "oude" subroutines is er hier en daar iets gewijzigd of toegevoegd.
Subroutine 110 is gewijzigd. Om het positioneren van de cursor (naar HO, VE) zo snel mogelijk te laten verlopen, hebben we hiervoor een klein machinetaal-programma gemaakt. Wanneer in het Basicode-2-programma naar regel 20 wordt gesprongen (en dat gebeurt standaard bij elk Basicode-programma), dan wordt eerst een stukje machinetaal in RAM weggeschreven. Als het programma daarna ergens een GOSUB 110 tegen komt, dan wordt dit machinetaal-programma aangeroepen en wordt de cursor zeer snel naar plaats HO, VE gebracht.
Subroutine 120, het opvragen van de positie van de cursor, is bij deze kombinatie ook mogelijk (bij de Junior met Elekterminal ging dat niet).
De enige routine die bij deze kombinatie niet mogelijk is, is subroutine 200. De Junior kan namelijk niet rechtstreeks detekteren of op een bepaald moment een toets is ingedrukt. Een GOSUB 200 in een programma moet dus worden veranderd. Eigenlijk zijn er twee routines die niet lopen, namelijk 200 en 250. Maar het piepje dat in subroutine 250 zou moeten worden gegenereerd, is toch niet belangrijk voor de goede werking van een programma. Tenslotte nog een belangrijke opmerking.

Als het Basicode-2-vertaalprogramma wordt gebruikt bij de DOS-Junior, dan moet men goed opletten wanneer tussendoor het kommando DISK!". . ." wordt gebruikt. Stel dat een BASIC-programma van de floppy wordt geladen met het kommando DISK!"LO . . ." en men wil dit programma vervolgens in Basicode-formaat op de band gaan zetten. Dat wegschrijven kan dan misgaan omdat bij een DISK!" . . ."-kommando page zero "geswapped" wordt. Dat heeft namelijk tot gevolg dat de pointers die in het Basicode-2-vertaalprogramma nodig zijn niet meer kloppen.
Hiervoor bestaat een heel eenvoudige oplossing. Nadat men iets van floppy heeft gehaald (of weggezet) wordt even het cijfer 1 ingetikt en dan een (CARRIAGE) RETURN gegeven. Er is dan een "dummy" regel ingevoerd, waardoor de pointers weer allemaal goed staan. Alles loopt in dat geval weer goed (op regel 1 mag natuurlijk wel niets staan, anders moet men een ander ongebruikt regelnummer ingeven).

Tabel 1. De standaard-subroutines voor de uitgebreide Junior met VDU-kaart.

Tabel 2. De subroutines voor de DOS-Junior met VDU-kaart.

1

```
LIST

 10 GOTO 1000
 20 DATA32,135,15,173,112,3,141,57,26,32,136,13
 21 DATA206,113,3,48,6,32,56,15,24,144,245,96
 22 FOR O=1 TO 24:READ OD:POKE881+O,OD:NEXT
 23 GOTO 1010
100 PRINT CHR$(27);:PRINT CHR$(49);
101 RETURN
110 IF HO>79 THEN RETURN
111 IF VE>23 THEN RETURN
112 POKE880,HO:POKE881,VE
113 OS=PEEK(8256):OT=PEEK(8257)
114 POKE8256,114:POKE8257,3
115 PRINT CHR$(13);
116 O=USR(O)
117 POKE8256,OS:POKE8257,OT
118 RETURN
120 HO=PEEK(6713):VE=PEEK(6712)
121 RETURN
200 IN$="":RETURN
210 OS=PEEK(8256):OT=PEEK(8257)
211 POKE8256,174:POKE8257,18
212 O=USR(O)
213 POKE8256,OS:POKE8257,OT
214 OX=(PEEK(6754)AND127)
215 IN$=CHR$(OX)
216 RETURN
250 RETURN
260 RV=RND(1):RETURN
270 FR=FRE(0):RETURN
300 IF SR<.01 AND SR>-.01 THEN SR=0
301 IF SGN(SR)=-1 THEN SR$=STR$(SR):RETURN
302 SR$=MID$(STR$(SR),2):RETURN
310 OS=ABS(SR)+.5*10^-CN:OI=INT(OS):OD=OS-OI+1
311 SR$=""
312 IF OS>=1E9 THEN 321
313 IF CN=0 THEN OD$="":GOTO 317
314 IF OD=1 THEN OD$=".":GOTO 316
315 OD$=MID$(STR$(OD),3,CN+1)
316 IF LEN(OD$)<CN+1 THEN OD$=OD$+"0":GOTO 316
317 SR$=MID$(STR$(OI),2)+OD$
318 IF SR<0 AND VAL(SR$)<>0 THEN SR$="-"+SR$
319 IF LEN(SR$)<CT THEN SR$=" "+SR$:GOTO 319
320 IF LEN(SR$)>CT THEN SR$=""
321 IF LEN(SR$)<CT THEN SR$=SR$+"X":GOTO 321
322 RETURN
350 PRINT SR$;:RETURN
360 PRINT:RETURN

Ok
```

2

```
LIST

 10 GOTO 1000
 20 DATA32,25,243,173,112,225,141,203,239,32,26,241
 21 DATA206,113,225,48,6,32,202,242,24,144,245,96
 22 FOR O=1 TO 24:READ OD:POKE57713+O,OD:NEXT
 23 GOTO 1010
100 PRINT CHR$(27);:PRINT CHR$(49);
101 RETURN
110 IF HO>79 THEN RETURN
111 IF VE>23 THEN RETURN
112 POKE57712,HO:POKE57713,VE
113 OS=PEEK(574):OT=PEEK(575)
114 POKE574,114:POKE575,225
115 PRINT CHR$(13);
116 O=USR(O)
117 POKE574,OS:POKE575,OT
118 RETURN
120 HO=PEEK(61387):VE=PEEK(61386)
121 RETURN
200 IN$="":RETURN
210 OS=PEEK(574):OT=PEEK(575)
211 POKE574,27:POKE575,254
212 O=USR(O)
213 POKE574,OS:POKE575,OT
214 IN$=CHR$(PEEK(9859))
215 RETURN
250 RETURN
260 RV=RND(1):RETURN
270 FR=FRE(0):RETURN
300 IF SR<.01 AND SR>-.01 THEN SR=0
301 IF SGN(SR)=-1 THEN SR$=STR$(SR):RETURN
302 SR$=MID$(STR$(SR),2):RETURN
310 OS=ABS(SR)+.5*10^-CN:OI=INT(OS):OD=OS-OI+1
311 SR$=""
312 IF OS>=1E9 THEN 321
313 IF CN=0 THEN OD$="":GOTO 317
314 IF OD=1 THEN OD$=".":GOTO 316
315 OD$=MID$(STR$(OD),3,CN+1)
316 IF LEN(OD$)<CN+1 THEN OD$=OD$+"0":GOTO 316
317 SR$=MID$(STR$(OI),2)+OD$
318 IF SR<0 AND VAL(SR$)<>0 THEN SR$="-"+SR$
319 IF LEN(SR$)<CT THEN SR$=" "+SR$:GOTO 319
320 IF LEN(SR$)>CT THEN SR$=""
321 IF LEN(SR$)<CT THEN SR$=SR$+"X":GOTO 321
322 RETURN
350 DISK!"IO ,00":PRINT SR$;:DISK!"IO ,01":RETURN
360 DISK!"IO ,00":PRINT :DISK!"IO ,01":RETURN

Ok
```

Het kan vaak erg gemakkelijk zijn als we de processor op de voet kunnen volgen bij het uitvoeren van een machinetaalprogramma. Dankzij het hier voorgestelde programma is dat mogelijk: bij elke stap van de processor wordt de inhoud van de registers, de stack en de pointer zichtbaar gemaakt, samen met de bijbehorende instruktie. Stap voor stap wordt het te testen programma op deze wijze doorgewerkt.

J. Ruppert

6502-tracer

analyse-programma voor machinetaal-programma's

Met het 6502-tracer-programma richten we ons nu eens niet alleen tot de junior-computer-bezitters, maar ook tot alle andere 6502-bezitters. Van de ½ Kbyte geheugenruimte die het programma in beslag neemt, staan slechts twee bytes in page zero. Een minimum aan veranderingen is dus voldoende om het programma aan te passen aan andere 6502-computers.

De kneep van de tracer

De werking van het programma kan eigenlijk worden samengevat in de woorden "stap-voor-stap-monitor". Het programma dat men wil analyseren wordt namelijk instruktie na instruktie afgewerkt, waarbij na het uitvoeren van elke instruktie de inhoud van de registers A, X en Y, de inhoud van het status register (flags NV DIZC) en de inhoud van de stack zichtbaar worden gemaakt. Hierbij valt misschien op dat bij het status register (NV DIZC) de flag "break" ontbreekt. Dit komt omdat het tracer-programma alle instrukties aksepteert met uitzondering van degene die een onderbreking van het programma tot gevolg hebben (BRK, IRQ en NMI).

Zoals men in tabel 3 kan zien wordt de analyse van het verloop van het programma (dat in het voorbeeld bestaat uit een reeks instrukties, afgewisseld met enkele register- en flag-bewerkingen) vergemakkelijkt door de informatie die door het tracer-programma wordt gegeven in de drie rechter kolommen. De eerste kolom, helemaal rechts, heeft betrekking op de stack: $FF is het lage byte van de stack pointer (het hoge byte is $01). Aan het einde van de listing ziet men enkele adressen die horen bij het uitvoeren van de instrukties JSR en RTS. De volgende kolom geeft de logische nivo's van de flags NV DIZC van het status register. Daarnaast staan tenslotte ook nog de inhouden van de registers A, X en Y van de processor. De listing van de adressen en de instrukties in gedisassembleerde vorm in de twee voorste kolommen volgt nauwkeurig het verloop van het programma, kompleet met sprongen en subroutines. Zo is te zien dat het programma steeds van adres $020D naar $0209 springt zolang flag Z logisch nul is.

De opbouw van het programma

De beschikbare bladruimte staat niet toe dat we in dit artikel een komplete sourcelisting afdrukken. We geven hier dan ook alleen maar een "gebruiksaanwijzing" van het programma, een hexdump en een korte beschrijving van de opbouw en werking.

Voordat het programma wordt uitgevoerd moet eerst het startadres van het te testen programma worden gezet in de geheugenplaatsen $00ED en $00EE, dat dan de pseudo program counter vervangt. Het te testen programma mag in ROM of RAM staan, maar het tracer-programma moet zich in RAM bevinden. In de hier gegeven vorm is het startadres $0500.

Van $0500 tot $0523 worden enkele bytes gezet die nodig zijn voor de pseudo-stack die begint op $0713 (zie verderop), de kop voor de kolommen wordt gemaakt en de

Tabel 1. 6502-tracer is een programma voor het analyseren van machinetaal-programma's. Het ½ Kbyte lange programma moet in RAM worden gezet.

Tabel 1

```
JUNIOR

M
HEXDUMP: 500,721
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0500: 58 20 95 06 A9 00 A0 0F 99 13 07 88 D0 FA B9 CC
0510: 06 20 A5 06 C8 C0 36 D0 F5 A9 26 8D 7E 1A A9 05
0520: 8D 7F 1A 4C A2 05 8D 1B 07 68 8D 20 07 68 68 8C
0530: 1C 07 8E 1D 07 BA 8E 14 07 D8 58 A0 03 B9 15 07
0540: 20 A0 06 20 A3 06 C8 C0 06 B0 11 AD 16 07 D0 09
0550: 20 A3 06 20 A3 06 4C 43 05 CE 16 07 C0 09 D0 DD
0560: AD 20 07 29 CF 8D 13 07 A2 08 0E 13 07 90 04 A9
0570: 31 D0 02 A9 2E 20 A5 06 CA D0 EF 20 A3 06 AD 14
0580: 07 20 A0 06 A9 2D 20 A5 06 BA E0 FF B0 14 68 8D
0590: 16 07 20 A0 06 E0 FE B0 05 68 48 20 A0 06 AD 16
05A0: 07 48 A0 00 20 95 06 A5 EE 20 A0 06 A5 ED 20 A0
05B0: 06 20 A3 06 B1 ED 8C 1A 06 8C 1B 06 8C 1A 07 8C
05C0: 19 07 20 A8 06 8C 1E 07 98 8D 16 07 CE 16 07 88
05D0: B1 ED 99 19 06 99 18 07 98 D0 F4 E6 ED D0 02 E6
05E0: EE CE 1E 07 D0 F5 AD 18 07 29 0F D0 13 AD 18 07
05F0: C9 20 F0 29 C9 40 F0 2E C9 60 F0 2E 29 10 D0 62
0600: AD 18 07 C9 4C F0 2C C9 6C F0 3D AE 1D 07 AC 1C
0610: 07 AD 20 07 48 AD 1B 07 28 D0 00 00 00 A5 ED 48
0620: A5 EE 48 4C 33 06 68 8D 20 07 68 85 EE 68 85 ED
0630: 4C 3D 06 AD 1A 06 85 ED AD 1B 06 85 EE A9 00 8D
0640: 19 06 20 9A 06 4C 0B 06 AD 1A 06 85 ED AD 1B 06
0650: 85 EE A0 00 B1 ED AA C8 B1 ED 85 EE 8A 85 ED 4C
0660: 3D 06 AD 20 07 48 AD 18 07 8D 6D 06 28 D0 03 4C
0670: 82 06 58 D8 AD 1A 06 30 11 18 65 ED 85 ED 90 02
0680: E6 EE A9 00 8D 1A 06 4C 00 06 18 65 ED 85 ED B0
0690: F1 C6 EE 90 ED A9 0D 20 A5 06 A9 0A 20 A5 06 60
06A0: 4C 8F 12 A9 20 4C 34 13 A0 01 C9 00 F0 1A C9 40
06B0: F0 16 C9 60 F0 12 A0 03 C9 20 F0 0C 29 1F C9 19
06C0: F0 06 29 0F AA BC 03 07 8C 21 07 60 36 35 30 32
06D0: 20 2D 20 54 52 41 43 45 52 0D 0A 41 44 52 2E 20
06E0: 2D 49 4E 53 54 52 2E 2D 20 3A 41 20 3A 59 20 3A
06F0: 58 20 4E 56 31 31 44 49 5A 43 20 53 54 41 43 4B
0700: 20 0D 0A 02 02 02 01 02 02 02 01 01 02 01 01 03
0710: 03 03 03 80 FB 00 00 00 D0 FD 00 04 71 08 00 00
0720: 31 02
```

IRQ-vektor wordt gezet (de IRQ-routine begint op adres $0526).
Vanaf $05A2 begint het eigenlijke "tracen": uitlezen van de program counter, ophalen van de opcode, volschrijven van het "instruktieveld" met 00 en het berekenen van de lengte van de instruktie (deze routine begint op $06A8 en lijkt op de routine LENACC van de junior computer).
Het "instruktieveld" is een zone in RAM van vier bytes ($0619 . . . 061C) waarin steeds een instruktie van het te testen programma wordt opgeslagen om daarna te worden uitgevoerd. Aangezien instrukties hooguit drie bytes lang zijn en het instruktieveld vier bytes groot is, wordt elke hierin opgeslagen instruktie automatisch gevolgd door minstens één 00, een BRK-instruktie dus. Dat heeft tot gevolg dat na het uitvoeren van elke instruktie van het te analyseren programma een BRK volgt, die dan zorgt voor het uitvoeren van de IRQ-routine op adres $0526 en verder.
Op $05DB vindt het ophogen van de pseudo program counter plaats ($00ED, $00EE); dit verhogen hangt af van de lengte van de zojuist afgewerkte instruktie. De lengte van die instruktie staat op adres $071E.
Vanaf $05E6 worden de sprong-instrukties uit het programma gehaald om apart te worden uitgevoerd als ze aan de beurt zijn. Vanaf $060B worden de registers A, X en Y op de stack gezet.
Op adres $0619 en verder bevindt zich het instruktieveld waarin telkens een instruktie wordt opgeslagen. Doordat deze instruktie altijd wordt gevolgd door minstens één BRK-instruktie, zal meteen na de opgeslagen instruktie de IRQ-routine worden uitgevoerd. Deze routine begint direkt na het uitvoeren van de uit het programma gelichte instruktie met het wegschrijven van de processor-registers. Vervolgens wordt de inhoud van de registers zichtbaar gemaakt en dan gaat het programma verder met de volgende instruktie.
Op $061D en verder bevinden zich speciale routines voor het uitvoeren van de sprong-instrukties. Van $0672 tot $068A gebeurt het berekenen van de relatieve sprongadressen. Op $06A1, 06A2 en $06A6, 06A7 staan de adressen voor de routines PRBYT en PRCHA van de junior computer. Deze adressen moeten worden aangepast bij andere 6502-systemen.
Van $06CC tot $0702 staan de bytes voor het maken van de "kop" van de kolommen. De adressen $0703 . . . 0712 bevatten een tabel waarin de lengte van elke instruktie is opgeslagen. Op $0713 . . . 0721 staan nog enkele bytes die het tracer-programma gebruikt voor het opslaan van de stack pointer, de inhoud van de top van de stack, de opcode die op een bepaald moment behandeld wordt, de lengte van de instruktie, de program counter, enzovoorts.
Met deze informatie en de hexdump van het tracer-programma zal het u voortaan geen moeite meer kosten om elk machinetaal-programma tot op het laatste byte te "ontrafelen".

Tabel 2

```
JUNIOR

M
HEXDUMP: 200,23A
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0200: A9 03 A8 AA A9 09 85 00 F8 18 65 00 CA D0 FA 2A
0210: 6A 38 E5 00 88 D0 FA E5 00 D8 F0 00 F0 06 F0 02
0220: F0 04 F0 FC F0 F8 20 30 02 38 EA 4C 35 02 EA EA
0230: 20 34 02 60 60 4C 00 03 4C 00 02

JUNIOR

M
HEXDUMP: 2F0,30F
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
02F0: 00 00 00 00 00 00 00 00 00 00 00 00 B0 06 B0 02
0300: B0 FC B0 F8 6C 07 03 00 02 00 00 00 00 00 00 00
0310:
```

Tabel 2. Met behulp van dit "voorbeeldprogramma" kan de tracer op zijn werking worden getest. Men moet hiermee het resultaat krijgen dat in tabel 3 is afgedrukt.

Tabel 3

```
ED
00ED 27 00.
00EE 09 02.
00EF 1C 500
0500 58 R
6502 - TRACER
ADR. -INSTR.- :A :Y :X NV11DIZC STACK
0200 A9 03     03 00 00 ........ FF-
0202 A8        03 03 00 ........ FF-
0203 AA        03 03 03 ........ FF-
0204 A9 09     09 03 03 ........ FF-
0206 85 00     09 03 03 ........ FF-
0208 F8        09 03 03 ....1... FF-
0209 18        09 03 03 ....1... FF-
020A 65 00     18 03 03 ....1... FF-
020C CA        18 03 02 ....1... FF-
020D D0 FA     18 03 02 ....1... FF-
0209 18        18 03 02 ....1... FF-
020A 65 00     27 03 02 ....1... FF-
020C CA        27 03 01 ....1... FF-
020D D0 FA     27 03 01 ....1... FF-
0209 18        27 03 01 ....1... FF-
020A 65 00     36 03 01 ....1... FF-
020C CA        36 03 00 ....1.1. FF-
020D D0 FA     36 03 00 ....1.1. FF-
020F 2A        6C 03 00 ....1... FF-
0210 6A        36 03 00 ....1... FF-
0211 38        36 03 00 ....1..1 FF-
0212 E5 00     27 03 00 ....1..1 FF-
0214 88        27 02 00 ....1..1 FF-
0215 D0 FA     27 02 00 ....1..1 FF-
0211 38        27 02 00 ....1..1 FF-
0212 E5 00     18 02 00 ....1..1 FF-
0214 88        18 01 00 ....1..1 FF-
0215 D0 FA     18 01 00 ....1..1 FF-
0211 38        18 01 00 ....1..1 FF-
0212 E5 00     09 01 00 ....1..1 FF-
0214 88        09 00 00 ....1.11 FF-
0215 D0 FA     09 00 00 ....1.11 FF-
0217 E5 00     00 00 00 ....1.11 FF-
0219 D8        00 00 00 ......11 FF-
021A F0 00     00 00 00 ......11 FF-
021C F0 06     00 00 00 ......11 FF-
0224 F0 F8     00 00 00 ......11 FF-
021E F0 02     00 00 00 ......11 FF-
0222 F0 FC     00 00 00 ......11 FF-
0220 F0 04     00 00 00 ......11 FF-
0226
     20 30 02  00 00 00 ......11 FD-0229
0230
     20 34 02  00 00 00 ......11 FB-0233
0234
     60        00 00 00 ......11 FD-0229
0233
     60        00 00 00 ......11 FF-
0229 38        00 00 00 ......11 FF-
022A EA        00 00 00 ......11 FF-
022B
     4C 35 02  00 00 00 ......11 FF-
0235
     4C 00 03  00 00 00 ......11 FF-
0300 B0 FC     00 00 00 ......11 FF-
02FE B0 02     00 00 00 ......11 FF-
0302 B0 F8     00 00 00 ......11 FF-
02FC B0 06     00 00 00 ......11 FF-
0304
     6C 07 03  00 00 00 ......11 FF-
0200 A9 03     03 00 00 .......1 FF-
0202 A8        03 03 00 .......1 FF-
0203 AA
JUNIOR
```

Tabel 3. Dit verschijnt op het scherm (of op de printer) als men het programma van tabel 2 analyseert met behulp van het tracer-programma. Voordat men de "tracer" start op adres $0500 moet eerst het start-adres van het te testen programma ($0200) in de geheugenplaatsen $00ED en $00EE worden gezet.

P. Barrat

Met behulp van een kort maar heel efficiënt programma kan het Tape Monitor-programma van de Junior computer worden uitgebreid met een nieuwe funktie, waarmee het mogelijk is programma's van cassette te lezen en ze daarna automatisch te laten starten. Of zoals de titel hier aangeeft: get = laad het programma, en go = voer het programma uit!

get & go

auto-run voor programma's op cassette

Het hier voorgestelde programma voor de Junior computer stelt ons in staat programma's die met behulp van TM van cassette in het RAM-geheugen worden geladen, automatisch te laten starten na het laden. Dit wordt bewerkstelligd door tijdens de RDTAPE-routine het return-adres dat op de stack is opgeslagen door de instruktie JSR-RDTAPE, te vervangen door het start-adres (SA) van het programma dat van de cassette wordt gelezen.
Als het inlezen wordt beëindigd en de processor de RDTAPE-routine verlaat via de RTS-instruktie, dan vindt hij op de stack niet meer het adres vanwaar hij was vertrokken voor het uitvoeren van RDTAPE, maar het start-adres van het programma dat zojuist is ingelezen. De processor gaat dus naar dat adres en voert het programma dan uit. Daarbij is het wel noodzakelijk dat het start-adres (SA) van het blok ingelezen data tevens het begin-adres van het programma is. Verder moet de stack leeg zijn (stack pointer is $FF) op het moment dat men de GET-toets bedient (uitvoering van de RDTAPE-routine). Aan die laatste voorwaarde wordt vanzelf voldaan als men "gewoon" de TM gebruikt, zoals verderop wordt beschreven.

Tabel 1. Dit stukje listing is bij de eerste druk van Junior-computer-boek 4 weggevallen op pagina 203 en bevat juist een gedeelte dat in dit artikel wordt besproken.

Tabel 1.

```
0673: 0B43 D0 F1          BNE   TENSYN RETURN IF LESS THAN 10 SYNS
0674:
0675: 0B45 20 36 0C  STAR  JSR   RDCH   WAIT FOR '*' CHARACTER
0676: 0B48 20 5D 0C        JSR   CHARVU
0677: 0B4B C9 2A           CMPIM '*
0678: 0B4D F0 06           BEQ   STARA
0679: 0B4F C9 16           CMPIM '       STILL SYN CHARACTER?
0680: 0B51 F0 F2           BEQ   STAR    IF YES, THEN WAIT
0681: 0B53 D0 AD           BNE   RDTAPE IF NOT, THEN RESYNC
0682:
0683: 0B55 20 5D 0C  STARA JSR   CHARVU DISPLAY '*'
0684: 0B58 20 F3 0B        JSR   RDBYT  READ ID FROM TAPE
0685: 0B5B CD 79 1A        CMP   ID     REQUESTED ID?
0686: 0B5E D0 3E           BNE   CHKID
0687:
0688: 0B60 20 F3 0B  RDSA  JSR   RDBYT  READ SAL FROM TAPE
0689: 0B63 20 4B 0C        JSR   CHKSUM CHECK SUM COMPUTATION
0690: 0B66 85 FA           STAZ  POINTL SET UP STORE POINTER
0691: 0B68 20 F3 0B        JSR   RDBYT  READ SAH FROM TAPE
0692: 0B6B 20 4B 0C        JSR   CHKSUM
0693: 0B6E 85 FB           STAZ  POINTH
0694:
0695: 0B70 20 F3 0B  FILMEM JSR   RDBYT  READ DATA BYTE FROM TAPE
0696: 0B73 30 8D           BMI   RDTAPE NOT VALID HEX CHARACTER
0697: 0B75 F0 13           BEQ   CHECK  END OF DATA CHARACTER?
0698: 0B77 20 4B 0C        JSR   CHKSUM
0699: 0B7A A0 00           LDYIM $00
0700: 0B7C 91 FA           STAIY POINTL STORE BYTE IN MEMORY
0701: 0B7E E6 FA           INCZ  POINTL SET POINTER FOR NEXT BYTE
0702: 0B80 D0 02           BNE   FMA
0703: 0B82 E6 FB           INCZ  POINTH
0704:
0705: 0B84 20 64 0C  FMA   JSR   VU     DISPLAY '*'
0706: 0B87 4C 70 0B        JMP   FILMEM READ NEXT DATA BYTE FROM TAPE
0707:
0708: 0B8A 20 F3 0B  CHECK JSR   RDBYT  READ CHECK SUM FROM TAPE
0709: 0B8D CD 6E 1A        CMP   CHKL   AND COMPARE IT
0710: 0B90 D0 09           BNE   SYNVEC
0711: 0B92 20 F3 0B        JSR   RDBYT
0712: 0B95 CD 6F 1A        CMP   CHKH
0713: 0B98 D0 01           BNE   SYNVEC
0714: 0B9A 60              RTS           RETURN TO CALLER
0715:
0716: 0B9B 4C 02 0B  SYNVEC JMP   RDTAPE
0717:
0718: 0B9E AD 79 1A  CHKID LDA   ID
0719: 0BA1 C9 00           CMPIM $00     ID = 00?
0720: 0BA3 F0 BB           BEQ   RDSA
0721: 0BA5 C9 FF           CMPIM $FF     ID = FF?
0722: 0BA7 D0 F2           BNE   SYNVEC
0723: 0BA9 20 F3 0B        JSR   RDBYT  READ SA FROM TAPE, BUT IGNORE IT
0724: 0BAC 20 4B 0C        JSR   CHKSUM
0725: 0BAF 20 F3 0B        JSR   RDBYT
0726: 0BB2 20 4B 0C        JSR   CHKSUM
0727: 0BB5 AD 70 1A        LDA   SAL     USE SA STORED IN BUFFER
0728: 0BB8 85 FA           STAZ  POINTL
```

DUMPB

Om het gewenste resultaat te verkrijgen maken we gebruik van een DUMPB-routine. Dit is in feite een gemodificeerde versie van de DUMP-routine van TM. De nieuwe routine zorgt voor een header op de band die drie data bevat. Dat zijn adres $01FE als laad-pointer, het start-adres van het programma dat RDTAPE op de adressen $01FE en $01FF zet, de top van de stack dus, en het byte $20 dat RDTAPE niet aksepteert, waardoor nog eens de gewone RDTAPE opnieuw wordt uitgevoerd. DUMPB wordt afgesloten met een sprong naar TM, wat betekent dat de routine DUMP verder op de gewone manier wordt uitgevoerd.
Als we de listing in tabel 2 vergelijken met de listing van DUMP (Junior-computer-boek 4, pagina 200), dan blijkt dat de instrukties voor het initialiseren van CHKL en CHKH zijn vervallen, evenals die van POINT en SA ($0A0A...0A19). Bovendien is er een initialiseringsinstruktie voor de stack pointer bij gekomen op $0730 (TXS in de listing van tabel 2). Verder is DUMPB identiek aan DUMP tot adres $0745.
Vervolgens is te zien hoe DUMPB adres $01FE op de band zet (dat RDTAPE zal zien als laad-vektor) en daarna het begin-adres van het blok te laden data verandert voordat dit ook op de band wordt gezet. Deze verandering is noodzakelijk voor een goed funktioneren na de RTS-instruktie aan het einde van RDTAPE. Het laatste karakter dat door DUMPB wordt verzonden is $20. De sprong JMP-TM

brengt ons weer terug in de normale procedure waarbij het programma verder gewoon op cassette wordt opgenomen door middel van DUMP.

Inlezen

Als we weer teruggrijpen naar de RDTAPE-listing (pagina 203 van boek 4), dan kunnen we goed zien wat er gebeurt bij het lezen van de header. Nadat de synchronisatiekarakters zijn gelezen, volgt het beginkarakter van de data (*), daarna komt het identifikatiekarakter ID. Vervolgens leest de RDTAPE-routine het adres $01FE als laad-vektor (POINT). Vlak daarna laadt de routine de volgende twee bytes en zet die in $01FE en 01FF, waardoor het juiste return-adres op de stack wordt gezet. Het nieuwe adres is niets anders dan het begin-adres van het programma dat nu geladen wordt. Het volgende byte dat daarna door RDTAPE wordt gelezen is het karakter "space" ($20), dat blijft steken bij de instruktie BMI op $0B73 (pagina 204 van boek 4). Dat heeft het opnieuw uitvoeren van RDTAPE tot gevolg, zodat het programma verder gewoon datgene leest dat door de DUMP-routine na het uitvoeren van DUMPB op de band is gezet.
Even een opmerking tussendoor. In de eerste druk van boek 4 is op pagina 203 een gedeelte van de listing weggevallen. In latere drukken is dat gedeelte wel aanwezig. Dat deel bevat juist de hier besproken adressen. Voor de boek 4-bezitters met zo'n "oude" druk geven we daarom in tabel 1 het ontbrekende deel.
Als het programma is geladen zorgt de RTS-instruktie op $0B9A er voor dat de processor het terugkeer-adres op de stack gaat zoeken. Zoals we reeds hebben gezien vindt hij hier het begin-adres van het zojuist geladen programma, zodat dit direkt daarna wordt uitgevoerd.

Gebruik

Om de bestaande TM niet te hoeven veranderen heeft de ontwerper van DUMPB een heel leuke oplossing bedacht. Het is voldoende als men het honderdtal bytes van tabel 1 in het geheugen zet vanaf adres $0700 (een ander adresbereik is ook zonder meer mogelijk). Vervolgens wordt de NMI-vektor ($1A7A, 1A7B) op het startadres van DUMPB gezet (in ons geval $0700). Nu kan men TM op de gewone manier gebruiken, met het enige verschil dat de toets ST/NMI van het hexadecimale toetsenbord voortaan de funktie SAVE heeft bij DUMPB.
Tenslotte willen we nog even wijzen op het feit dat bij een automatische programmastart de konfiguratie van de output-ports overeenkomt met die van RDTAPE en *niet* met die van de hex-monitor.

Tabel 2.

```
                                PAGE 01

0010: 0700                      ORG   $0700
0020:
0030:
0040:                   *PROGRAM  DUMPB*
0050:
0060:
0070:                   DEFINITIONS
0080:
0090: 0700              LOWER  *      $1A6D  HALF PERIOD BUFFER OF 2400 HZ
0100: 0700              HIGHER *      LOWER  -01 HALF PERIOD BUFFER OF 3600 HZ
0110: 0700              FIRST  *      $1A76  3600 CYCLE BUFFER
0120: 0700              SECOND *      FIRST  +01 2400 HZ CYCLE BUFFER
0130: 0700              GANG   *      $1A78  I/O TEMP.
0140: 0700              SYNCNT *      $1A74  SYNC. COUNTER
0150: 0700              OUTCH  *      $0AA3  OUTPUT CHAR. TO TAPE
0160: 0700              OUTBT  *      $0A8B  OUTPUT BYTE TO TAPE
0170: 0700              SAL    *      $1A70  START ADDRESS
0180: 0700              SAH    *      SAL    +01
0190: 0700              ID     *      $1A79  ID OF FILE
0200: 0700              PAD    *      $1A80  PORT A
0210: 0700              PADD   *      PAD    +01
0220: 0700              PBD    *      PAD    +02 PORT B
0230: 0700              PBDD   *      PAD    +03
0240: 0700              TM     *      $0856  DUMP
0250:
0260:
0270: 0700 A9 7D        DUMPB  LDAIM $7D    HALF PERIOD OF 3600 HZ
0280: 0702 8D 6C 1A            STA   HIGHER
0290: 0705 A9 C3               LDAIM $C3    HALF PERIOD OF 2400 HZ
0300: 0707 8D 6D 1A            STA   LOWER
0310: 070A A9 03               LDAIM $03    3 HALF PERIODS OF 3600 HZ
0320: 070C 8D 76 1A            STA   FIRST
0330: 070F A9 02               LDAIM $02    2 HALF PERIODS OF 2400 HZ
0340: 0711 8D 77 1A            STA   SECOND
0350:
0360: 0714 A9 47        DUMPT  LDAIM $47    PORT B PATTERN
0370: 0716 A2 FF               LDXIM $FF    PORT B IS OUTPUT
0380: 0718 8D 82 1A            STA   PBD
0390: 071B 8D 78 1A            STA   GANG
0400: 071E 8E 83 1A            STX   PBDD
0410: 0721 A9 00               LDAIM $00    PORT A PATTERN
0420: 0723 A2 7F               LDXIM $7F    PA6...PA0 IS OUTPUT.
0430: 0725 8D 80 1A            STA   PAD
0440: 0728 8E 81 1A            STX   PADD
0450: 072B A2 FF               LDXIM $FF
0460: 072D 8E 74 1A            STX   SYNCNT 255 SYNC CHARACTERS
0470: 0730 9A                  TXS          RESET STACK POINTER
0480:
0490: 0731 A9 16        SYNCS  LDAIM $16    SYNC. CHARACTER
0500: 0733 20 A3 0A            JSR   OUTCH  OUTPUT IT
0510: 0736 CE 74 1A            DEC   SYNCNT STILL MORE SYNCs?
0520: 0739 D0 F6               BNE   SYNCS
0530: 073B A9 2A               LDAIM '*     OPEN FILE CHARACTER
0540: 073D 20 A3 0A            JSR   OUTCH  OUTPUT IT
0550: 0740 AD 79 1A            LDA   ID     GET CURRENT ID
0560: 0743 20 8B 0A            JSR   OUTBT  OUTPUT IT


                                PAGE 02

0570: 0746 A9 FE               LDAIM $FE
0580: 0748 20 8B 0A            JSR   OUTBT
0590: 074B A9 01               LDAIM $01
0600: 074D 20 8B 0A            JSR   OUTBT  ADDRESS = $01FE
0610: 0750 AC 70 1A            LDY   SAL    GET START ADDRESS
0620: 0753 88                  DEY
0630: 0754 98                  TYA
0640: 0755 20 8B 0A            JSR   OUTBT  OUTPUT ADJUSTED START ADDRESS
0650: 0758 C8                  INY
0660: 0759 98                  TYA
0670: 075A 38                  SEC
0680: 075B E9 01               SBCIM $01
0690: 075D AD 71 1A            LDA   SAH
0700: 0760 E9 00               SBCIM $00
0710: 0762 20 8B 0A            JSR   OUTBT
0720: 0765 A9 20               LDAIM $20    SPACE
0730: 0767 20 A3 0A            JSR   OUTCH  OUTPUT A SPACE
0740: 076A 4C 56 08            JMP   TM     EXECUTE DUMP
0750:
0760:
-T

    SYMBOL TABLE 3400 3472
    DUMPB  0700    DUMPT  0714    FIRST  1A76    GANG   1A78
    HIGHER 1A6C    ID     1A79    LOWER  1A6D    OUTBT  0A8B
    OUTCH  0AA3    PADD   1A81    PAD    1A80    PBDD   1A83
    PBD    1A82    SAH    1A71    SAL    1A70    SECOND 1A77
    SYNCNT 1A74    SYNCS  0731    TM     0856
-


HEXDUMP:
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0700: A9 7D 8D 6C 1A A9 C3 8D 6D 1A A9 03 8D 76 1A A9 .ü.l....m....v..
0710: 02 8D 77 1A A9 47 A2 FF 8D 82 1A 8D 78 1A 8E 83 ..w..G......x...
0720: 1A A9 00 A2 7F 8D 80 1A 8E 81 1A A2 FF 8E 74 1A .... ..........t.
0730: 9A A9 16 20 A3 0A CE 74 1A D0 F6 A9 2A 20 A3 0A ... ...t....* ..
0740: AD 79 1A 20 8B 0A A9 FE 20 8B 0A A9 01 20 8B 0A .y. .... .... ..
0750: AC 70 1A 88 98 20 8B 0A C8 98 38 E9 01 AD 71 1A .p... ....8...q.
0760: E9 00 20 8B 0A A9 20 20 A3 0A 4C 56 08
```

Tabel 2. De listing van het programma DUMPB.

ID List

Junior zoekt automatisch programmanummers op band

P. Jenkins

Het goedkoopste massageheugen voor de Junior-computer is nog altijd de cassette-recorder. Naarmate het aantal programma's op band toeneemt wordt het echter steeds moeilijker om een bepaald programma terug te vinden. Of men moet nauwgezet alles op papier bijhouden, maar in de praktijk komt daar meestal niet zo veel van. Het IDList-programma is een soort elektronische inhoudsopgave, zodat men dit werkje niet meer zelf met de hand hoeft te doen. Bovendien wordt door het programma ook nog een pariteitskontrole uitgevoerd om te kijken of er geen fouten in de overdracht ontstaan.

Het lijkt misschien een tikkeltje overbodig om een programma voor de cassette-interface voor de Junior te brengen, terwijl de computer al een interface voor floppy-drives heeft. Er zijn echter nog vele lezers die met een cassette-recorder werken, omdat dit medium nog altijd stukken goedkoper is dan de vrij dure floppy-drives.

Net zoals bij verschillende andere Junior-programma's die de afgelopen maanden in Elektuur zijn gepubliceerd, maakt ook dit programma gebruik van bestaande Junior-routines (in dit geval TM en PM) om een nieuwe funktie te kreëren. Deze funktie maakt een lijst van alle programma-nummers (ID) met de bijbehorende begin- en eindadressen van de programma's die op een cassette staan. Tevens kontroleert het programma aan de hand van de kontrole-bytes of alles goed van de band wordt gelezen.

Het IDList-programma, waarvan de hex-dump in tabel 1 is gegeven, wordt eerst in de computer gezet en daarna kan men het programma starten op adres $Ø2ØØ. Stop een cassette in de recorder en start dan het programma door het indrukken van een willekeurige toets op het toetsenbord. Daarna hoeven we alleen maar te wachten totdat IDList de informatie op het display laat zien. Als inkorrekte data wordt gelezen, wordt dat ook op het display aangegeven. IDList kan worden onderbroken door middel van de BREAK-toets. Met de R-toets kan men weer verder gaan.

Bekende labels

Aangezien van het programma alleen een hex-dump is gegeven, zullen we nog een korte toelichting geven. Vanaf adres $Ø369 staan alle gegevens voor de display-berichtgeving (inklusief de naam van de auteur). Karakter $77 op $Ø368 en Ø3FB is het End-Of-File-karakter. Het centrale gedeelte van het programma bevat een aantal instrukties die ontleend zijn aan RDTAPE (lezers die bekend zijn met TM komen die instrukties zeker bekend voor). We geven nog een lijstje met labels voor degenen die geïnteresseerd zijn in het disassembleren van het programma:

Ø2ØØ: START
Ø2Ø3: RESET
Ø2Ø6: BRKTST
Ø22A: INIT
Ø247: RDTAPE (zie de source listing van de Tape Monitor)
Ø31Ø: IDSA
Ø32D: SUMERR
Ø33B: CORDAT
Ø349: MESSB
Ø357: MESEND
Ø358: CLS
Ø362: CLSA

Tabel 1

```
      Ø  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
Ø2ØØ: 4C 2A Ø2 2Ø BC 14 2C 8Ø 1A 1Ø FB A2 FF 9A 86 F2
Ø21Ø: A0 8Ø 2Ø 49 Ø3 A9 5F 8D 7C 1A A9 1Ø 8D 7D 1A A9
Ø22Ø: Ø2 85 FB A9 ØØ 85 FA 4C 6A 1Ø A9 Ø3 8D 7C 1A A9
Ø23Ø: Ø2 8D 7D 1A 2Ø 58 Ø3 EA EA EA A0 ØØ 2Ø 49 Ø3 2Ø
Ø24Ø: AE 12 A0 48 2Ø 49 Ø3 A9 32 8D 82 1A 8D 78 1A A9
Ø25Ø: 7E 8D 83 1A A9 7F 8D 81 1A A9 ØØ 8D 6E 1A 8D 6F
Ø26Ø: 1A A9 FF 8D 6B 1A 2C 8Ø 1A 1Ø 61 2Ø C2 ØB 6E 6B
Ø27Ø: 1A AD 6B 1A 2Ø E8 ØB C9 16 DØ EB A0 ØA 8C 69 1A
Ø28Ø: 2C 8Ø 1A 1Ø 47 2Ø 36 ØC 2Ø 5D ØC C9 16 DØ D2 CE
Ø29Ø: 69 1A DØ EC 2C 8Ø 1A 1Ø 33 2Ø 36 ØC 2Ø 5D ØC C9
Ø2A0: 2A F0 Ø7 C9 16 F0 ED 4C 47 Ø2 2Ø 5D ØC 2Ø F3 ØB
Ø2B0: 8D 79 1A 2Ø F3 ØB 2Ø 4B ØC 85 FA 8D 7Ø 1A 2Ø F3
Ø2C0: ØB 2Ø 4B ØC 85 FB 8D 71 1A 4C CF Ø2 4C Ø3 Ø2 2C
Ø2DØ: 8Ø 1A 1Ø F8 2Ø F3 ØB 3Ø 62 F0 ØF 2Ø 4B ØC E6 FA
Ø2E0: DØ Ø2 E6 FB 2Ø 64 ØC 4C CF Ø2 2Ø F3 ØB CD 6E 1A
Ø2F0: DØ 3B 2Ø F3 ØB CD 6F 1A DØ 33 2Ø BC 14 2Ø 1Ø Ø3
Ø3ØØ: A5 FB 2Ø 8F 12 A5 FA 2Ø 8F 12 2Ø E8 11 4C 47 Ø2
Ø31Ø: AD 79 1A 2Ø 8F 12 A0 8A 2Ø 49 Ø3 AD 71 1A 2Ø 8F
Ø32Ø: 12 AD 7Ø 1A 2Ø 8F 12 A0 8E 2Ø 49 Ø3 6Ø 2Ø BC 14
Ø33Ø: 2Ø 1Ø Ø3 A0 5E 2Ø 49 Ø3 4C 47 Ø2 2Ø BC 14 2Ø 1Ø
Ø34Ø: Ø3 A0 6F 2Ø 49 Ø3 4C 47 Ø2 B9 69 Ø3 C9 Ø3 F0 Ø7
Ø35Ø: 2Ø 34 13 C8 4C 49 Ø3 6Ø A9 ØC 2Ø 34 13 A9 84 8D
Ø36Ø: F7 1A 2C D5 1A 1Ø FB 6Ø 77 22 49 44 4C 49 53 54
Ø37Ø: 22 ØD ØA 42 59 2Ø 5Ø 41 55 4C 2Ø 53 2Ø 4A 45 4E
Ø38Ø: 4B 49 4E 53 2Ø 2Ø ØD ØA 54 55 52 4E 2Ø 4F 4E 2Ø
Ø39Ø: 54 41 5Ø 45 2Ø 28 5Ø 4C 41 59 29 2Ø 41 4E 44 2Ø
Ø3A0: 5Ø 52 45 53 53 2Ø 41 4E 59 2Ø 4C 45 54 54 45 52
Ø3BØ: Ø3 ØD 2Ø ØD ØA 49 44 2Ø 2Ø 2Ø 53 54 41 52 54 2Ø
Ø3CØ: 2Ø 45 4E 44 ØD ØA Ø3 43 48 45 43 4B 53 55 4D 2Ø
Ø3DØ: 45 52 52 4F 52 ØD ØA Ø3 43 4F 52 52 55 5Ø 54 45
Ø3EØ: 44 2Ø 44 41 54 41 ØD ØA Ø3 ØD ØA 42 52 45 41 4B
Ø3FØ: ØD ØA Ø3 2Ø 3A 2Ø Ø3 2Ø 2D 2Ø Ø3 77 2C
```

Tabel 1. Hexdump van het IDList-programma, startadres $Ø2ØØ. Het programma geeft niet alleen de ID-nummers met begin- en eindadressen van de blokken op de band, maar het voert ook nog een datakontrole uit (CHKL/CHKH: $1A6E/1A6F).

In de technische gegevens van een floppy-drive staat meestal wel ergens een waarde vermeld voor de MTBF (mean time between failures), de gemiddelde levensduur. Bij deze tijd gaat men er van uit dat de motor maar een fraktie van die opgegeven MTBF draait. In de floppy-disk-interface is hiermee oorspronkelijk geen rekening gehouden: de motor draait hierbij kontinu, wat de levensduur helaas niet ten goede komt. Door het toevoegen van een kleine schakeling is het mogelijk de motor alleen te laten draaien als dat werkelijk nodig is. Als de drive zo'n twaalf sekonden lang niet wordt gebruikt, wordt de motor ook weer automatisch uitgeschakeld.

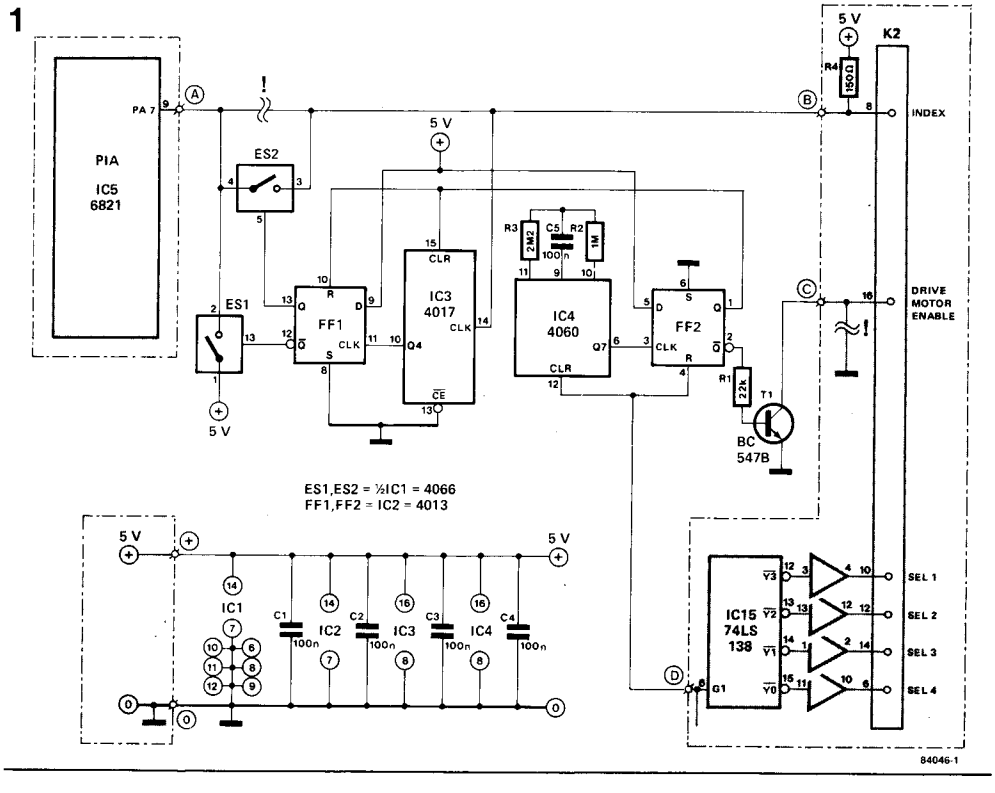
motorschakeling voor floppy-drives

voor een langere levensduur

De floppy-disk-interface uit Elektuur november en december 1982 is een betrouwbaar en goedkoop ontwerp (de schakeling bevat geen speciale dure IC's) voor het koppelen van een floppy-drive met een computer. Helaas heeft de schakeling (zoals later pas bleek) een klein minpuntje: de motor van de drive blijft kontinu draaien, ook als de floppy niet door de computer wordt "aangesproken". Dat kontinu draaien heeft wel enkele voordelen, zoals een korte toegangstijd omdat men niet steeds hoeft te wachten tot de motor weer op toeren is gekomen. Maar aan de andere kant geeft dat kontinu draaien van motor en floppy behoorlijk wat slijtage aan de floppy zelf en aan de koppen die (bij de meeste floppy-drives) voortdurend op de floppy rusten.
Er werd dus naar een mogelijkheid gezocht om de drive-motor alleen te laten draaien als het echt nodig is.

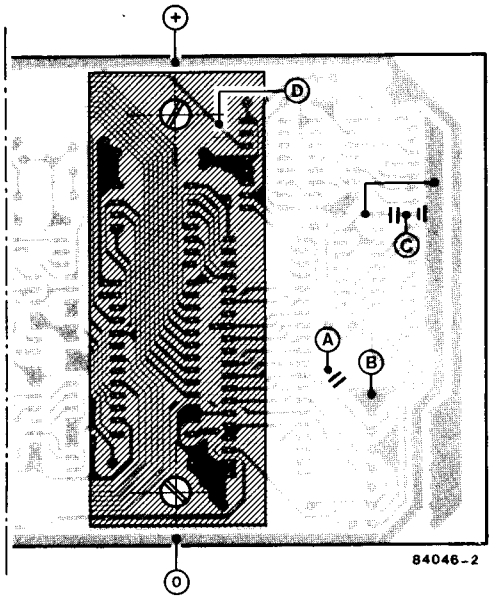
1 s ervoor en 12 s erna

Als we pen 16 van de floppy-konnektor op de floppy-disk-interface logisch nul maken, zal de drive-motor draaien. Het duurt echter eventjes voordat de motor zijn nominale toerental heeft bereikt. Daardoor is het alleen mogelijk om het select-signaal als drive motor enable-signaal te gebruiken als de eerste index-impulsen na het inschakelen worden "geëlimineerd". Deze methode heeft wel het nadeel dat de drive-motor weer meteen stopt als de floppy-drive door de computer wordt "uitgeschakeld". Dat is in de praktijk niet zo prettig omdat de disk vaak kort achter elkaar meerdere keren wordt geaktiveerd. Het zou dan ook handig zijn als de drive-motor nog enkele sekonden bleef doordraaien.
Al deze overwegingen hebben geleid tot de ontwikkeling van de in figuur 1 afge-



Figuur 1. De schakeling voor het in- en uitschakelen van de drive-motor. IC3, FF1, ES1 en ES2 zorgen er voor dat de eerste index-impulsen na het starten van de motor worden "geblokkeerd". De overige komponenten leveren een uitschakelvertraging, zodat de motor nog een tijdje blijft doorlopen nadat de drive is "afgeschakeld". Het is mogelijk de motorschakeling met de hand buiten bedrijf te stellen door middel van een schakelaar die pen 12 van IC4 en pen 6 van FF2 met ⊕ verbindt i.p.v. met pen 6 van IC15.

2



Figuur 2. Na enkele kleine veranderingen op de interfacekaart kan het printje met de motor-drive-schakeling aan de koperzijde van de grote print worden gemonteerd.

beelde schakeling. Het belangrijkste onderdeel in het schema is de elektronische schakelaar ES2 (en zijn "broertje" ES1). ES1 en ES2 worden in- en uitgeschakeld door flipflop FF1, die op zijn beurt weer wordt gestuurd door teller IC3. Als een drive moet worden geaktiveerd, zal signaal G1 (pen 6 van IC15 op de floppy-interface) logisch één worden. Dit signaal wordt gebruikt voor het resetten van teller IC4 en flipflop FF2. Dat betekent een "1" aan de $\overline{Q}$-uitgang van FF2, waardoor transistor T1 wordt opengestuurd en pen 16 van de konnektor naar nul wordt getrokken: de drive-motor gaat draaien. Vervolgens zullen de eerste (instabiele) index-impulsen verschijnen. Deze worden echter niet doorgegeven aan de PIA IC5, omdat ES2 geopend (en ES1 gesloten) is. IC3 telt de binnenkomende index-impulsen. Na de eerste vijf impulsen wordt uitgang Q4 van IC3 hoog, zodat FF1 een klokpuls ontvangt. ES2 sluit en ES1 opent en PA7 wordt weer verbonden met de aansluiting voor de index-impulsen.
Zolang ingang G1 van IC15 een logische één ontvangt blijven IC4 en FF2 in de reset-toestand. Als de computer de drive afschakelt wordt het G1-signaal weer nul. Vanaf dat moment kan IC4 de pulsen van de ingebouwde oscillator gaan tellen. Met de gegeven dimensionering duurt het ongeveer 12 sekonden totdat uitgang Q7 van IC4 hoog wordt. Op dat moment klapt FF2 om en gaat T1 sperren, zodat de drive-motor stopt. Het omschakelen van FF2 heeft tevens tot gevolg dat via zijn Q-uitgang FF1 en IC3 worden gereset. ES2 opent weer en ES1 sluit, zodat de index-signalen weer ontkoppeld zijn van de PIA.

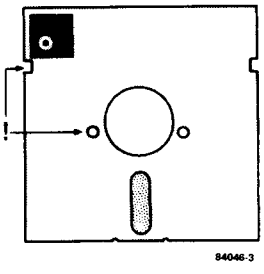
Andere mogelijkheden

Men kan de lengte van de "uitlooptijd" eenvoudig veranderen door het aanpassen van de waarden van R2 en/of C5 in het netwerkje voor de oscillator (R3 moet 2 . . . 10 keer zo groot zijn als R2). Ook de "voorlooptijd" kan men naar eigen wens aanpassen door het aantal getelde index-impulsen vóór het sluiten van ES2 te veranderen. Dat is mogelijk door pen 11 van FF1 met een andere Q-uitgang van IC3 te verbinden.
Men kan bij gebruik van meerdere drives ook alleen de motor van de geselekteerde drive laten draaien. In dat geval moet per drive een extra transistor met weerstand (zoals R1 en T1) voor het drive-motor-enable-signaal worden toegevoegd, waarbij het stuursignaal wordt geleverd door een NOR-poort (4001). De NOR-poort kombineert op zijn beurt het $\overline{Q}$-signaal van FF2 met het select-signaal van de bewuste drive (SEL1 . . . 4). Elke drive krijgt dan wel een aparte drive-motor-enable-line, dus aansluiting 16 op de konnektor kan niet meer voor alle drives gemeenschappelijk worden gebruikt.
In figuur 2 is getekend hoe de schakeling op de floppy-disk-interface kan worden ondergebracht. De vier IC's en de overige komponenten kunnen gemakkelijk worden gemonteerd op een stukje experimenteer-print. Dat printje wordt dan met behulp van twee boutjes en een paar afstandbusjes op de interface-print vastgezet. Voordat het printje wordt vastgeschroefd moet eerst nog het een en ander worden veranderd op de interface-print. Het koperspoor tussen pen 8 van konnektor K2 en pen 9 van IC5 moet worden onderbroken, evenals de verbinding tussen pen 16 van K2 en massa. Ook de verbinding tussen pen 16 en kondensator C11 wordt onderbroken, waarna het knooppunt C11/pen 6, 8 en 10 van IC12 apart met massa wordt verbonden. De "nieuwe" aansluitpunten A . . . D, + en 0 kunnen vervolgens worden verbonden met de overeenkomstige punten van de toegevoegde schakeling.

Enkele knutseltips

Voor het onderbreken van een printspoor kan men het beste gebruik maken van een scherp mes met een spitse punt. Maak twee inkervingen in het bewuste printspoor met een tussenruimte van twee tot drie millimeter. Verhit het "tussenstuk" dan met de soldeerbout totdat het loslaat van de print.
Tenslotte nog een diskette-tip. Een single-side-diskette heeft meestal aan beide zijden een magneetlaag. Het is dus mogelijk om ook de andere zijde van zo'n floppy te gebruiken. We hoeven daarvoor alleen maar een extra inkeping te maken voor de schrijf-beveiliging en een extra index-gat *in de hoes* (niet in de floppy). Dit moet wel met de nodige voorzichtigheid gebeuren. Op de floppy mogen beslist geen krassen komen, want dan wordt hij onbruikbaar. Probeer ook niet de floppy uit de hoes te halen! Figuur 3 laat zien op welke plaatsen de nieuwe gaten moeten worden gemaakt. Op deze wijze kan men zijn floppy-geheugenkapaciteit in een mum van tijd verdubbelen zonder dat het een cent kost!

3



Figuur 3. Het is mogelijk een single-side-diskette aan twee kanten te gebruiken door (voorzichtig) nog een beveiligings-inkeping en een index-gat in de hoes te maken.

Naarmate men zijn "programmatheek" verder uitbreidt, zal steeds meer de behoefte ontstaan om delen uit verschillende BASIC-programma's te gebruiken om zelf een nieuw programma samen te stellen. Maar hoe doet men dat? Als antwoord daarop geven we hier een programma waarmee men deze mogelijkheid kan toevoegen aan een Junior computer met DOS. Bezitters van andere systemen kunnen het programma vrij gemakkelijk aanpassen, op voorwaarde dat de aanwezige DOS of BASIC is uitgerust met een I/O-besturing die het mogelijk maakt het geheugen te beschouwen als een randapparaat, zoals dat bij de Junior kan.

merge voor BASIC-files

een programma voor het samenvoegen van twee BASIC-files

De funktie van dit merge-programma is het stap-voor-stap samenvoegen van verschillende BASIC-programma's. Niet alleen dàt is interessant, maar ook de wijze waarop dit verwezenlijkt wordt. Hiervoor wordt namelijk gebruik gemaakt van een interessante eigenschap van de BASIC en de DOS van de Junior computer, die ons in staat stelt om het geheugen als een I/O-device te gebruiken. Deze eigenschap is trouwens bij de meeste moderne computers aanwezig.
De I/O-besturing kan worden vergeleken met een schakelaar die, als hij op de juiste manier geprogrammeerd is, het werkgeheugen kan koppelen met de konventionele randapparaten (toetsenbord, beeldscherm, printer, enz.). Ook het geheugen kan echter als "randapparaat" worden gezien. Bij 0S65D heeft het geheugen I/O-nummer 5. Bij andere systemen zal men even in de handleiding moeten kijken om te zien hoe de I/O-sturing daar gebeurt.
De I/O-besturing staat onder kontrole van het DOS-systeem, maar het is ook mogelijk dit direkt vanuit BASIC te doen. Met behulp van de instruktie LIST#5 wordt bijvoorbeeld een BASIC-file in het werkgeheugen ($3A7E. . .), die daar in een kompakte (tokenized) vorm is opgeslagen, overgebracht naar adres $8000 en verder. Op de nieuwe geheugenplaatsen staat het hele programma dan wel in ASCII-vorm, dus in dezelfde vorm als het op het scherm of op de printer zou verschijnen. Het beginadres $8000 wordt automatisch door het DOS-systeem gekozen, maar de gebruiker kan zelf een ander beginadres kiezen.
Om het belang van deze manipulatie in te zien, moeten we goed weten in welke vorm een BASIC-programma in het geheugen wordt opgeslagen. Gewoonlijk wordt iedere BASIC-instruktie in de vorm van een kode in het werkgeheugen opgeslagen, en niet als een stel ASCII-kodes die samen het woord van de instruktie vormen. Na het geven van het kommando LIST#5 staat het programma wèl geheel in de vorm van ASCII-tekens in het geheugen, dus letterlijk zoals we het ook

Tabel 1.

```
2000 FORX=1TO24:PRINT:NEXT
2010 PRINTTAB(10)"--------------------
2020 PRINTTAB(10)"-FILE MERGE UTILITY-
2030 PRINTTAB(10)"--------------------
2040 PRINT:PRINT:PRINTTAB(10)"written by A. Nachtmann
2050 PRINT:PRINTTAB(10)"feb. 19, 1984
2060 PRINT:PRINT:PRINT
2070 PRINT"Be sure that both files to be linked have different line numbers.
2080 PRINT"If both files have some common line numbers boot up your system
2090 PRINT"with the RSEQ utility to renumber the lines.
2100 PRINT:INPUT"In which drive are the files to be merged A/B/C/D";D$
2110 D$=LEFT$(D$,1):D=ASC(D$):IF D<ASC("A") OR D>ASC("D") THEN2000
2120 PRINT:INPUT"enter first file name ";F$
2130 INPUT"enter second file name";S$
2140 PRINT:INPUT"are you ready";I$
2150 IF LEFT$(I$,1)<>"Y" THEN2140
2160 REM---RESET MEMORY INPUT POINTER
2170 POKE9098,0:POKE9099,128
2180 DISK!"SE A":DISK!"CA E400=12,7": DISK!"SE "+D$:DISK!"GO E481"
2190 A1=8*16^3+11: A2=8*16^3+2*16+4
2200 REM---
2210 A=A1
2220 FORX=1 TO LEN(F$)
2230 POKE A,ASC(MID$(F$,X,1)):A=A+1
2240 NEXT
2250 REM---
2260 A=A2
2270 FOR X=1TO LEN(S$)
2280 POKEA,ASC(MID$(S$,X,1)):A=A+1
2290 NEXT
2300 POKE8993,16
```

Tabel 2.

```
HEXDUMP: E400,E4FF
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
E400: 50 4F 4B 45 38 39 39 33 2C 31 0D 0A 00 0D 0A 44 POKE8993,1.....D
E410: 49 53 4B 21 22 4C 4F 20 20 20 20 20 20 20 22 3A ISK!"LO       ":
E420: 4C 49 53 54 23 35 0D 0A 44 49 53 4B 21 22 4C 4F LIST#5..DISK!"LO
E430: 20 20 20 20 20 20 20 22 3A 4C 49 53 54 23 35 0D        ":LIST#5.
E440: 0A 44 49 53 4B 21 22 47 4F 20 45 34 35 32 22 0D .DISK!"GO E452".
E450: 0A 00 AE 91 23 AD 92 23 8E 66 E4 8D 67 E4 A2 00
E460: BD 00 E4 F0 1B 8D FF FF EE 66 E4 D0 03 EE 67 E4
E470: AD 66 E4 8D 91 23 AD 67 E4 8D 92 23 E8 D0 E1 60
E480: 60 A2 00 A9 80 8E 66 E4 8D 67 E4 A2 0D D0 D1
```

Tabel 3.

```
HEXDUMP: 2200,2251
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
2200: A9 01 8D 5E 26 20 BC 26 A9 2A 85 FF 20 54 27 86
2210: FE 20 67 29 A0 BF 20 EC 22 F0 03 88 D0 F8 8C 00
2220: 23 20 AC 15 20 9E 0F 20 94 15 20 73 2D 0D 0A 2D
2230: 2D 20 44 49 53 4B 20 32 20 2D 2D 0D 0A 0A 00 4C
2240: E6 2A 20 20 20 20 20 20 20 20 20 20 20 20 20 20
2250: 20 51
```

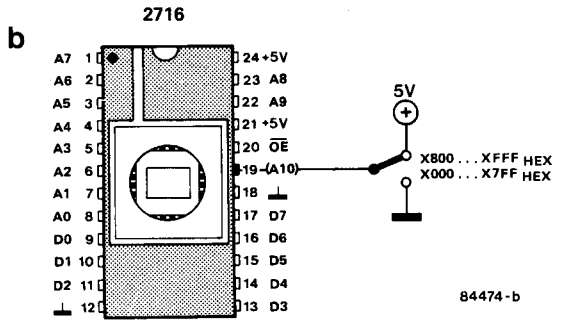
zouden intypen. Van deze eigenschappen maakt het merge-programma nu juist gebruik.

BASIC + BASIC

Het "koppel"-programma bestaat uit een stukje machinetaal (tabel 2) en een kort BASIC-programma (tabel 1). Nadat het programma weet in welke "eenheid" de bewuste files zich bevinden (D$) en wat de namen van de files zijn (F$ en S$), initialiseert de processor de pointer die het adres bepaalt waar begonnen wordt met het neerschrijven van de "verplaatste" file. Vervolgens worden een kort machinetaalprogramma en een look-up-table geladen vanaf $E400 (van sektor 7 van track 12; er blijft nog plaats achter de directory). Het machinetaalprogramma wordt gestart door de instruktie GO op regel 2180. In de buffer ($8000...) wordt dan een serie instrukties *zonder regelnummer* gezet (zie het gedeelte rechts in tabel 2). De regels 2190 tot 2290 van het BASIC-programma zorgen er voor dat de namen van de twee achter elkaar te koppelen files (F$ en S$) achter de twee instrukties LO (van de direkte instruktie uit tabel 2) worden geplaatst. De instruktie op regel 2300 programmeert de input-distributor zodanig dat het geheugen als input-device dienst doet. De BASIC-interpreter ontvangt nu de reeks instrukties van de buffer, vanaf adres $8000, alsof deze een voor een op het toetsenbord werden ingegeven... en hij voert ze een voor een uit. Dat betekent dat hij file F$ laadt, overbrengt naar $8000 en verder (LIST#5), vervolgens wordt file S$ geladen en dan achter file F$ in het geheugen gezet. Daarna wordt de instruktie DISK! "GO E452" uitgevoerd, de laatste direkte instruktie die de interpreter ontvangt van het geheugen als input-device.
Het machinetaalprogramma vanaf $E452 zet achter de twee geladen files in de buffer $8000 de instruktie POKE 8993,1. Deze "direct mode"-instruktie wordt niet voorafgegaan door een regelnummer, waardoor ze direkt wordt uitgevoerd als de interpreter haar tegenkomt. Deze laatste instruktie zorgt voor het "terugzetten" van de input-distributor naar de originele stand (het toetsenbord wordt dan weer input-device). Nu laadt de BASIC-editor in zijn werkgeheugen de twee files F$ en S$ vanuit de buffer $8000..., voegt ze samen tot één geheel, vertaalt alle instrukties weer in de betreffende kodes en list het geheel tevens op het scherm. Als hij de laatste regel van de tweede file heeft afgewerkt, ziet hij de instruktie POKE 8993,1. Deze wordt uitgevoerd en dat betekent dat het toetsenbord weer input-device wordt.
Als men nu een LIST geeft, ziet men dat het geheugen netjes achter elkaar de files F$ en S$ bevat.

RSEQ

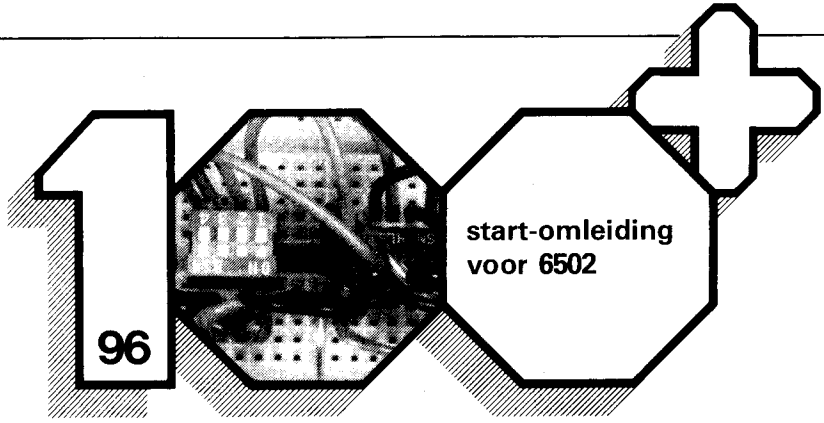
Voor een effektief gebruik van deze merge-faciliteit is het eigenlijk nodig dat men de programma's (of stukken daarvan) eenvoudig kan hernummeren. Op disk 2 van de serie van vijf diskettes van de Ohio-DOS bevindt zich een programma RSEQ dat deze taak verricht. Tot nu toe hebben we nog niet beschreven hoe deze disk 2 kan worden aangepast voor de Junior computer. Voor de aanpassing dient tabel 3, waarna RSEQ beschikbaar is voor het hernummeren van alle BASIC-programma's, speciaal diegene die men wil samenvoegen.
Het aanpassen van disk 2 is niet moeilijk. Men begint met het maken van een kopie van de originele disk (voor het geval er iets mis mocht gaan!), vervolgens wordt track 0 van disk 2 geladen met behulp van de TRACK 0 R/W UTILITY (RA200) op $A200 (of een andere plaats). Daarna wordt de inhoud van deze track veranderd volgens tabel 3 en dan kan men track 0 weer terugschrijven op de diskette (WA200/2200,8). Dat is alles!



kening) kan men bijvoorbeeld omschakelen tussen twee verschillende programma's in de EPROM.

— Pen 18, die bij de 2708 aan massa hangt, kan bij de 2716 ook met massa verbonden blijven. Dit is bij de 2716 wel de $\overline{CE}$-ingang, maar dat maakt voor het gewone gebruik niets uit. Het enige nadeel hiervan is een iets hogere dissipatie van het IC in de stand-by-toestand.

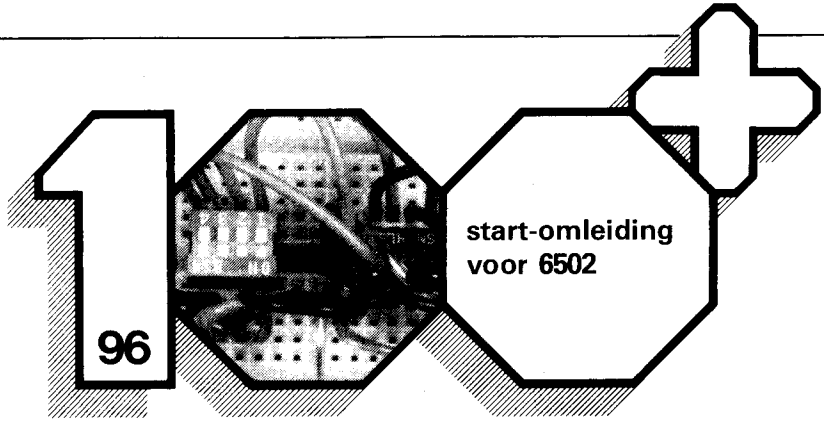
Voor het uitvoeren van deze modifikaties kan men gebruik maken van een extra 24-pens IC-voetje, of men sleutelt rechtstreeks aan printbanen. Men hoeft alleen maar pen 21 te verbinden met +5 V en pen 19 met +5 V of massa, afhankelijk van het gewenste geheugenblok (of men kiest voor de schakelaar, zoals in de tekening is afgebeeld).



start-omleiding voor 6502

initialisering naar eigen wens

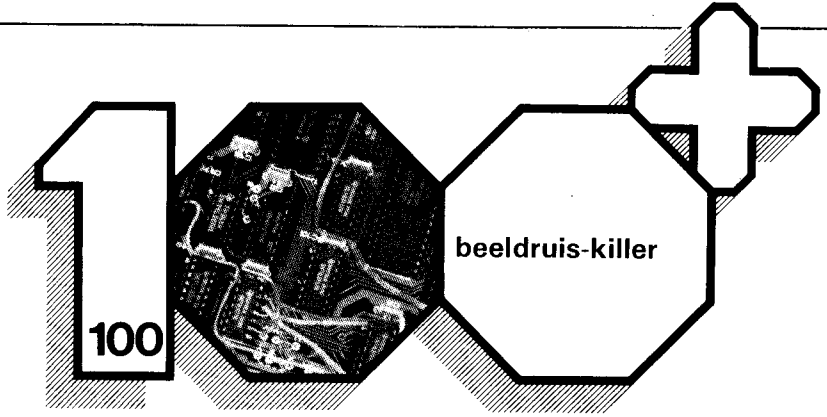
Bij de initialiseringsprocedure van de 6502-processor wordt begonnen met het ophalen van de initialiseringsvektor die op adres $FFFC/FFFD staat in ROM. Dit is een vast bedrade instruktie (beter gezegd: geïntegreerd in de chip van de μP), waaraan niets kan worden veranderd. Bovendien wijst die vektor naar een gedeelte van het niet-vluchtige geheugen dat, vooral bij een fabrieksklare computer, meestal vrij ontoegankelijk is voor de gebruiker. Met een kleine schakeling is het echter mogelijk de 6502 bij het starten om te leiden naar een ander initialiseringsadres dat men zelf kan kiezen: $XFFC/XFFD, waarbij X elke hexadecimale waarde tussen 0 en E mag zijn. Op die adressen vindt de processor dan een nieuwe vektor die verwijst naar een startroutine (in EPROM) die door de gebruiker zelf is gemaakt ter vervanging van de standaard-routine die de fabrikant van het apparaat heeft ingebouwd. Om dit te verwezenlijken maken we gebruik van een soort adres-omlegging. Steeds als de processor een adres tussen $FFF8 en FFFF neemt (de adresdekodering is iets ruimer uitgevallen dan noodzakelijk; in principe hoeven natuurlijk maar twee adressen omgeleid te worden), ontvangt de adresbus een adres dat ligt tussen $XFF8 en XFFF. Hierbij wordt X bepaald door de stand van vier schakelaars (of draadbruggen). Om deze verschuiving mogelijk te maken gaan we de adreslijnen A3 . . . A15 bekijken. Deze worden toegevoerd



start-omleiding voor 6502

initialisering naar eigen wens

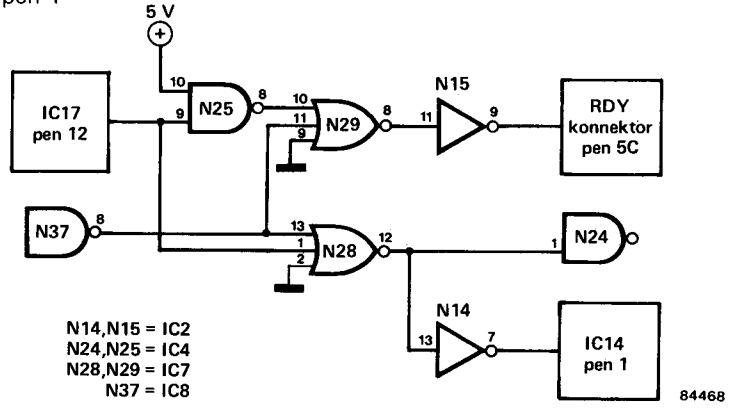
Bij de initialiseringsprocedure van de 6502-processor wordt begonnen met het ophalen van de initialiseringsvektor die op adres $FFFC/FFFD staat in ROM. Dit is een vast bedrade instruktie (beter gezegd: geïntegreerd in de chip van de μP), waaraan niets kan worden veranderd. Bovendien wijst die vektor naar een gedeelte van het niet-vluchtige geheugen dat, vooral bij een fabrieksklare computer, meestal vrij ontoegankelijk is voor de gebruiker. Met een kleine schakeling is het echter mogelijk de 6502 bij het starten om te leiden naar een ander initialiseringsadres dat men zelf kan kiezen: $XFFC/XFFD, waarbij X elke hexadecimale waarde tussen 0 en E mag zijn. Op die adressen vindt de processor dan een nieuwe vektor die verwijst naar een startroutine (in EPROM) die door de gebruiker zelf is gemaakt ter vervanging van de standaard-routine die de fabrikant van het apparaat heeft ingebouwd. Om dit te verwezenlijken maken we gebruik van een soort adres-omlegging. Steeds als de processor een adres tussen $FFF8 en FFFF neemt (de adresdekodering is iets ruimer uitgevallen dan noodzakelijk; in principe hoeven natuurlijk maar twee adressen omgeleid te worden), ontvangt de adresbus een adres dat ligt tussen $XFF8 en XFFF. Hierbij wordt X bepaald door de stand van vier schakelaars (of draadbruggen). Om deze verschuiving mogelijk te maken gaan we de adreslijnen A3 . . . A15 bekijken. Deze worden toegevoerd

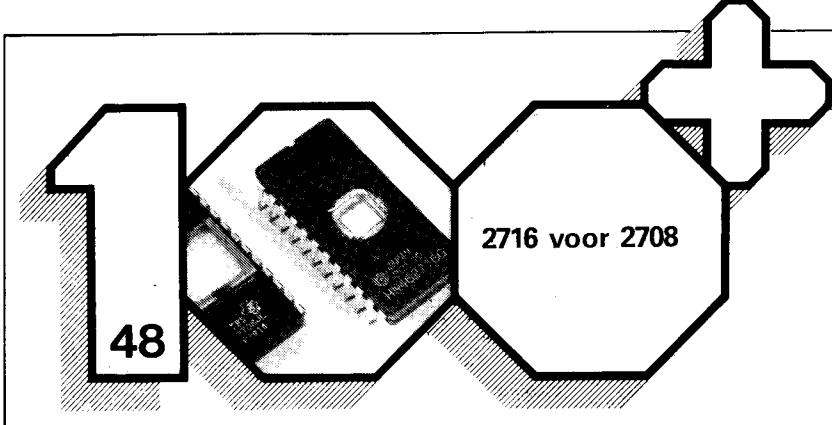


voor VDU-kaart met CMOS-junior

Bij de VDU-kaart wil het wel eens voorkomen dat er wat ruis in het beeld zichtbaar is, bijvoorbeeld tijdens het listen van een programma. Dit euvel kan door middel van enkele (bij de kombinatie Junior-VDU-kaart niet gebruikte) poorten op de VDU-kaart worden opgelost.
De truuk van de schakeling bestaat uit het stoppen van de processor wanneer deze in de video-RAM wil schrijven tijdens de "display enable"-tijd. Aangezien alleen een **65C02** tijdens het schrijven kan worden gestopt, werkt dit alleen bij een Junior met een 65C02-processor. Door dit tijdelijke stoppen ontstaat in principe enig tijdverlies bij het uitvoeren van programma's, maar in de praktijk is dat nauwelijks merkbaar.
Voor de modifikatie worden de volgende IC-pennen op de VDU-kaart opzij gebogen, zodat ze niet meer in de voetjes steken:
IC2-pen 7, 9, 11 en 13
IC4-pen 1, 8, 9 en 10
IC7-pen 1, 8, 9, 10, 11, 12 en 13
IC8-pen 8
IC17-pen 1

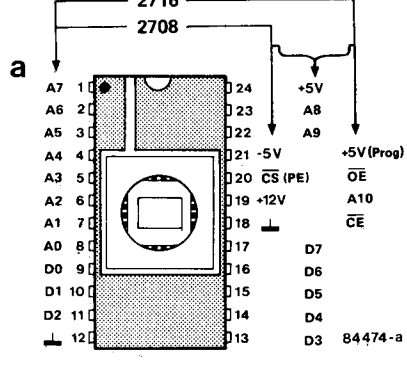
Daarna worden deze pennen volgens het schema met elkaar verbonden (de dikke lijnen zijn de nieuwe verbindingen). Pen 1 van IC17 blijft gewoon open hangen, terwijl pen 2 van IC7 op de print al aan massa ligt. Denk er aan dat aan pen 1 van IC14 en pen 12 van IC17 wel een draadje moet worden gesoldeerd, maar dat deze pennen ook in het IC-voetje moeten blijven steken.
Het is ook nog mogelijk om een licht venster op de monitor zichtbaar te maken, waarbinnen zich het videogebeuren afspeelt. Hiervoor hoeft slecht één weerstand van 1 k op de VDU-print te worden toegevoegd tussen pen 5 van IC17 en de kollektor van T1. ■





half gebruikt en toch goedkoper

De 2708-EPROM is tegenwoordig èen ietwat achterhaald geheugen. Zijn opvolger, de 2716, is namelijk momenteel goedkoper, hij heeft een twee maal zo grote geheugenkapaciteit (2048 × 8 bits) en hij hoeft maar één enkele voedingsspanning te hebben. Vooral dat laatste is een groot voordeel, want bij de 2708 zijn maar liefst drie verschillende voedingsspanningen nodig. Aangezien de 2716 dus goedkoper en bovendien beter verkrijgbaar is dan de 2708, kan het handig zijn als men een 2716 kan inzetten op de plaats van een 2708. Hiervoor zijn slechts enkele modifikaties nodig; de originele adresdekodering blijft ongewijzigd.

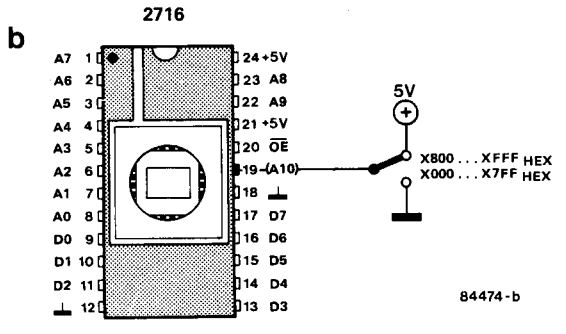


De meeste aansluitingen van een 2716 zijn direkt uitwisselbaar met die van de 2708. Er zijn slechts vier pennen waarbij we moeten opletten:

— Pen 21 (bij de 2708 aangesloten op −5 V) moet bij de 2716 worden verbonden met +5 V.

— Bij pen 20 hoeft niet echt iets te worden veranderd. Deze pen krijgt alleen een andere benaming. De aanduiding $\overline{CS}$ van de 2708 (chip select) is eigenlijk onjuist, want het is in werkelijkheid een $\overline{OE}$ (output enable). De $\overline{OE}$ van de 2716 op deze pen kunnen we dus gewoon blijven gebruiken.

— Pen 19, die bij de 2708 is verbonden met +12 V, wordt nu adresingang A10 van de 2716. Door deze pen aan +5 V of aan de nul te leggen kunnen we kiezen tussen de twee 1 K-blokken in de 2716. Met behulp van een wisselschakeling (zie te-



kening) kan men bijvoorbeeld omschakelen tussen twee verschillende programma's in de EPROM.

— Pen 18, die bij de 2708 aan massa hangt, kan bij de 2716 ook met massa verbonden blijven. Dit is bij de 2716 wel de $\overline{CE}$-ingang, maar dat maakt voor het gewone gebruik niets uit. Het enige nadeel hiervan is een iets hogere dissipatie van het IC in de stand-by-toestand.

Voor het uitvoeren van deze modifikaties kan men gebruik maken van een extra 24-pens IC-voetje, of men sleutelt rechtstreeks aan printbanen. Men hoeft alleen maar pen 21 te verbinden met +5 V en pen 19 met +5 V of massa, afhankelijk van het gewenste geheugenblok (of men kiest voor de schakelaar, zoals in de tekening is afgebeeld).

De DOS van Ohio Scientific, die bij de Junior computer wordt gebruikt, blijkt in de praktijk een fijn en doordacht systeem te zijn dat kan worden aangepast aan de persoonlijke wensen van elke gebruiker. Door de doorzichtige opzet van dit disk operating system kan de ondernemende hobbyist vrij eenvoudig zelf uitbreidingen aan het systeem toevoegen. We tonen in dit artikel enkele mogelijkheden: een uitbreiding van de DIR-instruktie (het lezen van de directory van een floppy zonder BEXEC* te gebruiken) en een uitbreiding van de PUT-instruktie (wegschrijven van een file naar disk zonder dat daarvoor eerst een naam in de directory is gezet). En als uitsmijter hebben we nog een mysterieus "turbo-byte".

DOS-uitbreidingen

twee extra disk-kommando's

F. Schmidt

Eigenlijk is het te gek... Aan de ene kant proberen we ons leven zo gemakkelijk mogelijk te maken met behulp van computers, en aan de andere kant doen we een hoop moeite om aan zo'n computer weer van alles te veranderen. De hier voorgestelde modifikaties kosten wel enige moeite, maar dat is een eenmalige kwestie; daarna heeft men er alleen maar voordeel van.

Twee kommando's

De nieuwe instruktie "DIRECTORY" (afkorting: DI) geeft ons de mogelijkheid om een komplete inhoudsopgave van een floppy op te vragen, zonder dat daarvoor gebruik hoeft te worden gemaakt van een of ander BASIC-programma. De vorm van de instruktie blijft in de oude staat, dus DI TT, waarbij TT het nummer is van het nummer is van de track waarvan men het aantal sektoren wil weten. Hierbij moet wel worden opgemerkt dat alleen de eerste helft van de directory toegankelijk is met de nieuwe instruktie (32 van de 64 mogelijke file-namen). In de praktijk vormt dat nauwelijks een beperking, want het komt zelden voor dat een floppy meer dan zo'n dertig verschillende files bevat.
De bestaande instruktie PUT FILENAME kan alleen worden gebruikt voor het wegschrijven van een file naar disk als voor deze file een naam in de directory is gereserveerd. Deze beperking heeft menige gebruiker al eens problemen bezorgd.
Met de nieuwe PUT-instruktie kan men ook een file wegschrijven als de naam van die file nog niet in de directory staat. Als de DOS de bewuste naam niet vindt in de directory, dan wordt er gekeken of nog voldoende vrije tracks op de floppy zijn om de file in op te slaan. Als dit onderzoek positief uitvalt schrijft de DOS eerst de nieuwe naam in de directory en daarna wordt de PUT-instruktie op de gewone wijze afgewerkt. Evenals bij de DIR-instruktie werkt de PUT-instruktie alleen bij de eerste helft van de directory. Mocht zich nog data op de gebruikte tracks bevinden (die geen naam in de directory heeft), dan wordt deze overschreven door de nieuwe file. Bovendien moet de schijf goed geformatteerd zijn voordat men de PUT-instruktie kan geven. Blijken er niet meer genoeg "lege" tracks voor de nieuwe file beschikbaar te zijn, dan verschijnt op het scherm de foutmelding ERR#E; bij het overschrijden van de eerste helft van de directory voor de nieuwe naam geeft de DOS de foutmelding ERR#F.

En dit moet men doen

Nu we toch bezig zijn met het aanbrengen van veranderingen, kunnen we gelijk van de gelegenheid gebruik maken om een klein foutje in de instrukties HO en SE weg te werken. Bij deze instrukties wordt de kop wel op de schijf gezet, maar er niet van af gehaald. En nu we toch bezig zijn, begin eens met het veranderen van het byte $D4_{HEX}$ op adres $26A5_{HEX}$ in het "turbo-byte" $D2_{HEX}$. Geef daarna enkele schrijf- en lees-kommando's op verschillende tracks en kijk of alles nog korrekt funktioneert. Ja? Dan reageert de drive goed op deze snelheidsaanpassing en kan men voortaan sneller een track op de floppy opzoeken. Reageert de drive niet meer na deze aanpassing, dan is het betreffende loopwerk niet geschikt voor deze hogere snelheid. Het "turbo-byte" kan men dan niet gebruiken.
Nu de procedure voor het inbrengen van de modifikaties. Om dit gedeelte zo eenvoudig mogelijk te houden, is gekozen voor een opzet waarbij het nieuwe gedeelte in RAM wordt gezet vanaf adres $E400_{HEX}$. Er zijn weliswaar andere mogelijkheden die minder uitgebreid zijn, maar de realisatie daarvan is een stuk moeilijker. Vandaar de keus voor deze opzet.
We beginnen met het maken van een kopie van disk 5 (tutorial disk 5) van Ohio, in een aangepaste vorm voor de Junior computer. Op deze kopie worden de volgende wijzigingen aangebracht:

- Ga naar de extended monitor (EM) en laad de bytes uit de hexdump van ta-

bel 1 op de aangegeven adressen.

- Schrijf deze dan weg met
  !SA 12,5 = E400/2
  (sektor 5 van track 12 is namelijk nog vrij!)
- Laad de tracks 1 en 0 op de volgende wijze:
  !CA 4A00 = 01,1
  !EX 41FD = 00
- Met behulp van de extended monitor kan men de veranderingen heel eenvoudig invoeren na het geven van de instruktie D42B0,42A0. De veranderingen zijn in tabel 2 gegeven. Daarna komen de volgende wijzigingen:
  4E42: FF 4E43: E3 4663: 4C
  4664: 6A 4665: E4 4E0D: 76
  4E0E: E4
  en tenslotte
  46A5: D2
  voor het versnellen van de kopverplaatsing.
- Schrijf de inhoud van track 1 terug naar disk met de instruktie
  !SA 01,1 = 4A00/8
- Laad de routine van track 0 op de volgende manier:
  !CA 0200 = 06,4
  en start deze met de instruktie: GO 0200. De inhoud van track 0 wordt gesaved met
  W4200/2200,8
  Klaar!

De zachte aanpak

Niet iedereen wil gaan "knoeien" in de DOS om daarin onherstelbare (wat die ene floppy betreft) modifikaties aan te brengen. Met het BASIC-programma van tabel 3 zijn deze extra's toe te voegen zonder ingrijpende wijzigingen. Dit geeft de gebruiker de mogelijkheid om naar keuze de DOS-uitbreidingen in en uit te schakelen (enable/disable DOS-extensions). Het RAM-bereik tussen $E400_{HEX}$ en $E5AD_{HEX}$ blijft gereserveerd voor de extra's als men de nieuwe funkties PUT en DIR gebruikt; men heeft hierbij het voordeel dat niet meer met de hand de tracks 0 en 1 hoeven te worden veranderd.
Als men de machinetaalroutines op een andere plaats wil opslaan dan sektor 5 van track 12, vergeet dan niet de laadinstruktie voor deze routines aan te passen in het programma van tabel 3 (DISK! "CA E400 = 12,5").
De nieuwe uitbreidingen zijn zeker de moeite van het modificeren waard. Als men dat eens geprobeerd heeft met het BASIC-programma, dan zal de definitieve modifikatie zeker niet lang uitblijven. ◀

Tabel 1

```
HEXDUMP: E400,E5AE
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
E400: AC E5 2C B1 E1 C9 20 30 0A C9 23 F0 06 20 2E 2D
E410: 4C F3 29 20 73 2D 0A 20 20 20 44 49 52 45 43 54
E420: 4F 52 59 0D 0A 0A 00 20 1F E5 A9 79 85 10 A9 2E
E430: 85 11 20 34 E5 F0 2D A0 00 B1 10 20 43 23 C8 C0
E440: 06 D0 F6 20 73 2D 20 20 00 A0 06 B1 10 20 92 2D
E450: 20 73 2D 20 2D 20 00 A0 07 B1 10 20 92 2D 20 73
E460: 2D 0D 0A 00 20 41 E5 D0 C9 60 20 54 27 20 8A 26
E470: 20 66 26 4C 61 27 68 AA 68 A8 48 8A 48 C9 DF D0
E480: 29 C0 2B D0 25 A5 E0 8D E5 2C 20 1F E5 20 70 E5
E490: AD 7D 3A 85 10 A8 A2 FF E8 E8 28 F0 7E 88 30 0E
E4A0: BD AE E5 F0 F3 A4 10 4C 98 E4 A9 0C D0 6A 38 8A
E4B0: E5 10 AA A9 00 18 F8 69 01 CA D0 FB D8 48 20 58
E4C0: E5 D0 52 68 A0 06 91 10 C8 18 F8 6D 7D 3A E9 00
E4D0: D8 91 10 38 A5 10 ED E5 2C 85 10 B0 02 C6 11 AC
E4E0: E5 2C A2 06 B1 E1 C9 23 F0 04 C9 19 10 06 C8 91
E4F0: E1 88 A9 20 91 10 C8 CA D0 EA A9 01 8D 5E 26 8D
E500: 5F 26 A9 79 85 FE A9 2E 85 FF 20 54 27 20 E1 27
E510: 68 68 4C DD 2B 68 A9 0F 4C 4B 2A A9 0E D0 F9 A9
E520: 79 85 FE A9 2E 85 FF A9 12 20 BC 26 A9 01 8D 5E
E530: 26 4C 1A 2B A0 05 B1 10 C9 23 D0 04 88 10 F7 C8
E540: 60 18 A5 10 69 08 85 10 A5 11 69 00 85 11 C9 2F
E550: D0 08 A5 10 C9 79 D0 02 A9 00 60 A9 79 85 10 A9
E560: 2E 85 11 20 34 E5 F0 07 20 41 E5 D0 F6 A9 FF 60
E570: A2 27 A9 00 9D AE E5 CA 10 FA 8E AE E5 A9 79 85
E580: 10 A9 2E 85 11 20 34 E5 F0 1E A0 07 B1 10 88 48
E590: A2 FF 38 F8 E9 01 E8 B8 FB 68 38 F1 10 D8 A8 A9
E5A0: FF 9D AE E5 CA 88 10 F9 20 41 E5 D0 D8 60
```

Tabel 2

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
4280: 8C 00 23 A2 01 8E C6 2A A9 00 85 FE A9 E4 85 FF
4290: A9 12 20 BC 26 A9 05 8D 5E 26 20 67 29 4C B3 22 .
```

Tabel 3

```
80 PRINT: PRINT: PRINT: PRINT
90 PRINT "CHOOSE ONE OF THE FOLLOWING OPTIONS:"
100 PRINT
110 PRINT "   - ENABLE  DOS-EXTENSIONS (1)"
120 PRINT "   - DISABLE DOS-EXTENSIONS (2)"
130 PRINT
140 PRINT SPC(7);: INPUT "YOUR CHOICE ";CHOICE
150 IF CHOICE=1 OR CHOICE=2 GOTO 200
160 END
200 DIM ADDR(6),BYTE(6)
210 REM ADDRESSES
220 DATA 11842,11843: REM POINTER TO DI-1
230 DATA 9827,9828,9829: REM JMP TO HOME
240 DATA 11789,11790: REM POINTER TO PUT
250 REM DATA
260 DATA 255,227,76,106,228,118,228
270 DATA 40,43,32,138,38,75,42
280 REM LOAD MACHINE LANGUAGE ROUTINE FROM TR 12, SEC 5
290 IF CHOICE=1 THEN DISK!"CA E400=12,5"
300 REM CHANGE ADDRESSES IN DOS
310 FOR I=0 TO 6: READ ADDR(I): NEXT
320 IF CHOICE=1 GOTO 340
330 FOR I=0 TO 6: READ DUMMY: NEXT
340 FOR I=0 TO 6: READ BYTE(I): NEXT
350 FOR I=0 TO 6: POKE ADDR(I),BYTE(I): NEXT
360 ON CHOICE GOTO 400,500
400 PRINT: PRINT "   --- DOS-EXTENSIONS ENABLED ---"
410 PRINT "!!! MEMORY FROM $E400 ON IN USE !!!"
420 NEW
500 PRINT: PRINT "   --- DOS-EXTENSIONS DISABLED ---"
510 NEW
```

Tabel 1. De uitbreidingsroutines voor de funkties DIR en PUT bij de aangepaste versie van de Ohio Scientific DOS voor de Junior computer.

Tabel 2. De modifikaties voor de inhoud van track 0 op disk 5. Een paar bytes moeten ook nog worden veranderd bij de inhoud van track 1; dit is in de tekst vermeld.

Tabel 3. Een eenvoudiger oplossing voor de extra funkties biedt dit BASIC-programma.

Als we deze tijd delen door de lijntijd krijgen we het aantal lijnen dat moet worden toegevoegd (in R5) om uit te komen op een rastertijd van 20 ms:
416/64 = 6,5 lijnen.
Dat wordt afgerond op 6.
Vervolgens moet de plaats van de vertikale synchronisatiepuls worden berekend. Dat gebeurt op praktisch dezelfde wijze als de berekening van de horizontale sync-puls:
VP = (VTT — (VT + 1500)) / 2 + VT.
Hierbij is VTT de rastertijd. In ons geval:
34 x 576 + 6 x 64 = 19968 μs.
Ook de "nieuwe" VT moet even worden berekend:
24 x 576 = 13824 μs
(hier dus zonder die extra speelruimte van 1 regel).
De inhoud van R7 wordt dan:
(19968 — (1500 + 13824)) / 2 + 13824 = 16146 μs.
Dat moet nog even worden gedeeld door de regeltijd:
16146/576 = 28,03.
In dat register komt dus de afgeronde waarde 28 te staan, $1C_{hex}$.
De inhoud van R8 bepaalt de werkwijze van de controller. Normaal werken we met een niet-geïnterlinieerd beeld, zodat de inhoud van R8 nul is.

De cursor

Het hier beschreven programma maakt slechts beperkt gebruik van de in de 6845 aanwezige cursor-mogelijkheden. Wie dit gedeelte wil uitbreiden, kan eenvoudig zelf nog enkele BASIC-regels aan het programma toevoegen (zie hiervoor ook Paperware 3).
De inhoud van de registers 10 en 11 bepaalt de hoogte en de plaats van de cursor (register 10: cursor start, register 11: cursor end). Verder bepalen bit 5 en 6 van register 10 de aanwezigheid en een eventueel knipperen van de cursor. Het programma kiest bij een karakterhoogte van 8 lijnen voor een knipperende cursor die bestaat uit een lijntje op de onderste karakterlijn. Als er meer dan 8 lijnen beschikbaar zijn voor een regel wordt de cursor juist onder het karakter geschreven.
De registers 12...17 worden bij de VDU-kaart in de huidige vorm niet gebruikt. Het volstaat om ze te vullen met nul.

Een voorbeeld

In tabel 1 staat het BASIC-programma voor de berekening van de register-inhouden. In principe kan het programma worden gebruikt voor elk systeem waarin een 6845 wordt toegepast. Na de start vraagt het programma eerst de breedte van de hele lijn (horizontal line length (char.)). Aan de hand daarvan wordt de vereiste kristalfrekwentie berekend en op het scherm getoond. De gebruiker kiest dan een goed verkrijgbare waarde die daarbij zo dicht mogelijk in de buurt ligt. Daarna moeten nog drie vragen worden beantwoord: aantal aktieve karakters per lijn, aantal lijnen per karakter en het aantal "karakterlijnen" (regels).

```
HORIZONTAL LINE LENGTH (CHAR.):
? 128
FREQUENCY =  16  MHZ
CRYSTAL FREQUENCY (MHZ):
? 16
NUMBER OF CHARACTERS PER LINE:
? 80
NUMBER OF SCAN LINES:
? 9
NUMBER OF CHARACTER LINES:
? 24


SCREEN FORMAT =  80  X  24


REGISTER R 0        = $7F
REGISTER R 1        = $50
REGISTER R 2        = $62
REGISTER R 3        = $08
REGISTER R 4        = $21
REGISTER R 5        = $06
REGISTER R 6        = $18
REGISTER R 7        = $1C
REGISTER R 8        = $00
REGISTER R 9        = $08
REGISTER R 10       = $49
REGISTER R 11       = $09
REGISTER R 12       = $00
REGISTER R 13       = $00
REGISTER R 14       = $00
REGISTER R 15       = $00

CLOCK PERIOD                .5 MICROSECONDS
LINE SYNC. PULSE WIDTH      4 MICROSECONDS
LINE SYNC. PULSE PERIOD     64 MICROSECONDS
HORIZONTAL DISPLAY TIME     40 MICROSECONDS
HORIZONTAL POSITION         49 MICROSECONDS
CHARACTER LINE PERIOD       576 MICROSECONDS
RASTER SYNC. PERIOD         19968 MICROSECONDS
VERTICAL DISPLAY TIME       13824 MICROSECONDS
VERTICAL POSITION           16128 MICROSECONDS
```

Tabel 2. Zo ziet een proefuitdraai eruit.

Vervolgens worden de register-inhouden berekend en op het scherm getoond in hexadecimale vorm (zie tabel 2). De ingegeven data zijn allemaal decimaal.
En wat doen we dan met de resultaten? Die moeten nu nog in de registers van de CRTC worden gezet. Als men geen Elektuur-systeem met VDU-kaart bezit, maar een ander systeem waarin de 6845 wordt toegepast, dan zal men de dokumentatie van dat systeem moeten raadplegen om er achter te komen hoe de initialiseringsroutine van de 6845 werkt.
In het Elektuur-systeem (zie ook Paperware 4) bestaat de initialiseringsprocedure uit twee delen: een voor het laden van de look-up-tabel met de parameters van ROM naar RAM (dat zijn de CRT timing tables en de routine MOVCRT) en de ander voor het overbrengen van de parameters van RAM naar de CRTC (routine CRTINT). Die laatste routine is hier van belang. Met behulp van poke-instrukties worden de data voor de registers eerst weggeschreven naar adres $EFDC_{hex}$ (in decimale vorm 61404) en de daaropvolgende plaatsen. Dan kan de routine worden gestart met DISK! "GO F36C". Het is aan te bevelen voor het laden van de nieuwe register-inhouden een master-reset te geven in de vorm DISK! "GO F330".

Op de VDU-kaart van Elektuur (september 1983) wordt een video-controller toegepast van het type 6845 (6545). Met behulp van het hier gepresenteerde BASIC-programma is het mogelijk om de inhoud voor alle registers van dit IC te berekenen. Men gaat hierbij uit van het gewenste beeldformaat, waarna het programma de daarvoor benodigde oscillatorfrekwentie en de registerinhouden in hexadecimale vorm geeft.

de 6845 geprogrammeerd

BASIC-programma berekent CRTC-registers

naar een idee van
P. Fransen

Niet iedere VDU-kaart-bezitter zal de behoefte hebben om zijn beeldformaat te veranderen. Meestal heeft men aan één formaat wel voldoende. Met dit BASIC-programma is het echter mogelijk om elke gewenste indeling te maken, zodat men bijvoorbeeld een nieuw formaat kan maken dat precies voldoet aan de eigen wensen. En zelfs als men dat niet wil, is het programma zelf toch interessant. Het laat namelijk op een duidelijke wijze zien hoe men moet werken met de verschillende CRTC-registers.

De verschillende parameters

We gaan hier niet verder in op de interne struktuur van de 6845, de werking en de scherm-opbouw. Daarvoor verwijzen we naar het artikel "karakters op het scherm" (september 1983) en de uitgave "Paperware 3, universele terminal". Hier worden alleen de registers nader bekeken.
In Europa gaan de videonormen uit van een lijnfrekwentie van 15625 Hz en een rasterfrekwentie van 50 Hz. Daaruit kan eenvoudig de vereiste lijntijd worden berekend:
1/15625 = 64 μs.
De rastertijd is
1/50 = 20 ms.
Laten we voor de berekening van de register-inhouden uitgaan van het bestaande kristal van 16 MHz op de VDU-kaart (in feite moet die frekwentie nog worden berekend, maar we hebben vast wat vooruit gerekend om straks op die waarde uit te komen).

Lijnfrekwentie
Elk karakter is in de breedte opgebouwd uit 8 "dots". Dit is een vastliggend gegeven. Voor elke dot is één klokperiode nodig. Als we de klokfrekwentie f_x noemen, dan is één dot-tijd $1/f_x$ en één karaktertijd 8 x $1/f_x$. Het programma begint nu met het vragen van het totale aantal (zichtbare en niet zichtbare) karakters voor één komplete lijntijd (de tijd voor de synchronisatiepuls is hierbij inbegrepen). We kiezen hiervoor 128 karakters. Aan de hand daarvan kan het totale aantal dots voor één lijntijd worden berekend en dus is dan ook de vereiste klokfrekwentie bekend:

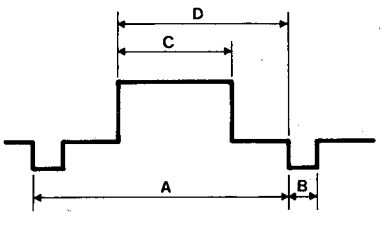
$$f_x = \frac{128.8}{65\ \mu s} = 16\ \text{MHz}.$$

Daar er niet altijd zo'n mooie ronde waarde uit deze berekening rolt, kan de gebruiker daarna een waarde intoetsen die zo dicht mogelijk in de buurt van de berekende waarde ligt (bijvoorbeeld een courante kristalfrekwentie). In dit geval kunnen we 16 MHz aanhouden; dat is een goed verkrijgbare kristalwaarde. Aan de hand van dit gegeven wordt vervolgens de werkelijke karakter-tijd uitgerekend. Dit is 8 x 1 dot-tijd:

$$\frac{8 \times 1}{16\ \text{MHz}} = 0{,}5\ \mu s.$$

Deze waarde kunnen we uitstekend gebruiken voor de volgende berekeningen.
In register Ø staat nu het totale aantal karakters per lijn minus 1. De inhoud is in ons voorbeeld dan
128 — 1 = 127, oftewel $7F_{hex}$.
In register R1 staat het aantal gebruikte (zichtbare) karakters per lijn, bijvoorbeeld 80 (50_{hex}).
De inhoud van register R2 bepaalt de positie van de horizontale synchronisatiepuls (zie ook figuur 1). Deze plaats wordt als volgt berekend:
HP = ((TSL — DT — 1,5 x LPB) / 2) + DT
waarbij DT = horizontale aktieve karaktertijd (in μs)
TSL = lijntijd (in μs)
LPB = tijdsduur van horizontale syncpuls (in μs)
en HP = de plaats van de syncpuls op de lijn
De waarde van DT is
80 x 0,5 = 40 μs.
En de waarde van LPB (die in R3 staat) leggen we vast op 8 karaktertijden, dat is dus
8 x 0,5 = 4 μs.
Daaruit volgt:
HP = (64 — 40 — 1,5 x 4) / 2 + 40 = 49 μs.
De faktor 1,5 is hier in de berekening gebouwd om straks in de praktijk nog

1



Figuur 1. Deze tekening toont het verband tussen de door de CRTC opgewekte signalen en de door de gebruiker gedefinieerde parameters. Tijd A is de lijntijd, B de duur van de syncpuls, C de duur van de aktieve karakters en D bepaalt de horizontale positie van het aktieve gedeelte op het beeld. Men kan de tekening ook gebruiken voor de vertikale tijdverdeling. A is dan de rastertijd, B de breedte van de rasterpuls, C de grootte van het vertikale beeldvenster en D bepaalt de plaats van het beeldvenster in vertikale zin.

Als we deze tijd delen door de lijntijd krijgen we het aantal lijnen dat moet worden toegevoegd (in R5) om uit te komen op een rastertijd van 20 ms:
416/64 = 6,5 lijnen.
Dat wordt afgerond op 6.
Vervolgens moet de plaats van de vertikale synchronisatiepuls worden berekend. Dat gebeurt op praktisch dezelfde wijze als de berekening van de horizontale sync-puls:
VP = (VTT — (VT + 1500)) / 2 + VT.
Hierbij is VTT de rastertijd. In ons geval:
34 x 576 + 6 x 64 = 19968 μs.
Ook de "nieuwe" VT moet even worden berekend:
24 x 576 = 13824 μs
(hier dus zonder die extra speelruimte van 1 regel).
De inhoud van R7 wordt dan:
(19968 — (1500 + 13824)) / 2 + 13824 = 16146 μs.
Dat moet nog even worden gedeeld door de regeltijd:
16146/576 = 28,03.
In dat register komt dus de afgeronde waarde 28 te staan, $1C_{hex}$.
De inhoud van R8 bepaalt de werkwijze van de controller. Normaal werken we met een niet-geïnterlinieerd beeld, zodat de inhoud van R8 nul is.

De cursor

Het hier beschreven programma maakt slechts beperkt gebruik van de in de 6845 aanwezige cursor-mogelijkheden. Wie dit gedeelte wil uitbreiden, kan eenvoudig zelf nog enkele BASIC-regels aan het programma toevoegen (zie hiervoor ook Paperware 3).
De inhoud van de registers 10 en 11 bepaalt de hoogte en de plaats van de cursor (register 10: cursor start, register 11: cursor end). Verder bepalen bit 5 en 6 van register 10 de aanwezigheid en een eventueel knipperen van de cursor. Het programma kiest bij een karakterhoogte van 8 lijnen voor een knipperende cursor die bestaat uit een lijntje op de onderste karakterlijn. Als er meer dan 8 lijnen beschikbaar zijn voor een regel wordt de cursor juist onder het karakter geschreven.
De registers 12...17 worden bij de VDU-kaart in de huidige vorm niet gebruikt. Het volstaat om ze te vullen met nul.

Een voorbeeld

In tabel 1 staat het BASIC-programma voor de berekening van de register-inhouden. In principe kan het programma worden gebruikt voor elk systeem waarin een 6845 wordt toegepast. Na de start vraagt het programma eerst de breedte van de hele lijn (horizontal line length (char.)). Aan de hand daarvan wordt de vereiste kristalfrekwentie berekend en op het scherm getoond. De gebruiker kiest dan een goed verkrijgbare waarde die daarbij zo dicht mogelijk in de buurt ligt. Daarna moeten nog drie vragen worden beantwoord: aantal aktieve karakters per lijn, aantal lijnen per karakter en het aantal "karakterlijnen" (regels).

```
HORIZONTAL LINE LENGTH (CHAR.):
? 128
FREQUENCY =  16  MHZ
CRYSTAL FREQUENCY (MHZ):
? 16
NUMBER OF CHARACTERS PER LINE:
? 80
NUMBER OF SCAN LINES:
? 9
NUMBER OF CHARACTER LINES:
? 24


SCREEN FORMAT =  80  X  24


REGISTER R 0        = $7F
REGISTER R 1        = $50
REGISTER R 2        = $62
REGISTER R 3        = $08
REGISTER R 4        = $21
REGISTER R 5        = $06
REGISTER R 6        = $18
REGISTER R 7        = $1C
REGISTER R 8        = $00
REGISTER R 9        = $08
REGISTER R 10       = $49
REGISTER R 11       = $09
REGISTER R 12       = $00
REGISTER R 13       = $00
REGISTER R 14       = $00
REGISTER R 15       = $00

CLOCK PERIOD                .5 MICROSECONDS
LINE SYNC. PULSE WIDTH      4 MICROSECONDS
LINE SYNC. PULSE PERIOD     64 MICROSECONDS
HORIZONTAL DISPLAY TIME     40 MICROSECONDS
HORIZONTAL POSITION         49 MICROSECONDS
CHARACTER LINE PERIOD       576 MICROSECONDS
RASTER SYNC. PERIOD         19968 MICROSECONDS
VERTICAL DISPLAY TIME       13824 MICROSECONDS
VERTICAL POSITION           16128 MICROSECONDS
```

Tabel 2. Zo ziet een proefuitdraai eruit.

Vervolgens worden de register-inhouden berekend en op het scherm getoond in hexadecimale vorm (zie tabel 2). De ingegeven data zijn allemaal decimaal.
En wat doen we dan met de resultaten? Die moeten nu nog in de registers van de CRTC worden gezet. Als men geen Elektuur-systeem met VDU-kaart bezit, maar een ander systeem waarin de 6845 wordt toegepast, dan zal men de dokumentatie van dat systeem moeten raadplegen om er achter te komen hoe de initialiseringsroutine van de 6845 werkt. In het Elektuur-systeem (zie ook Paperware 4) bestaat de initialiseringsprocedure uit twee delen: een voor het laden van de look-up-tabel met de parameters van ROM naar RAM (dat zijn de CRT timing tables en de routine MOVCRT) en de ander voor het overbrengen van de parameters van RAM naar de CRTC (routine CRTINT). Die laatste routine is hier van belang. Met behulp van poke-instrukties worden de data voor de registers eerst weggeschreven naar adres $EFDC_{hex}$ (in decimale vorm 61404) en de daaropvolgende plaatsen. Dan kan de routine worden gestart met DISK! "GO F36C". Het is aan te bevelen voor het laden van de nieuwe register-inhouden een master-reset te geven in de vorm DISK! "GO F330".

```
100 REM XXX CONSTANTS XXX
105 DIM R(15)
110 R(3)=8
120 K$="REGISTER"
130 L$="MICROSECONDS"
150 REM XXXXXXXXXX R0 XXXXXXXXXX
160 PRINT "HORIZONTAL LINE LENGTH (CHAR.): "
170 INPUT A0
180 R(0)=A0-1
190 TC=64/A0
200 FX=8/TC
210 PRINT "FREQUENCY = ";FX;" MHZ"
220 PRINT "CRYSTAL FREQUENCY (MHZ): "
230 INPUT FX
240 TC=1/(FX/8)
250 LPB=R(3)*TC
260 TSL=A0*TC
300 REM XXXXXXXXXX R1 XXXXXXXXXX
310 PRINT "NUMBER OF CHARACTERS PER LINE: "
320 INPUT R(1)
330 DT=R(1)*TC
400 REM XXXXXXXXXX R2 XXXXXXXXXX
410 HP=DT+(TSL-1.5*LPB-DT)/2
420 R(2)=HP/TC
500 REM XXXXXXXXXX R3 XXXXXXXXXX
600 REM XXXXXXXXXX R4 XXXXXXXXXX
610 PRINT "NUMBER OF SCAN LINES: "
620 INPUT A
623 IF A<8 THEN PRINT "MINIMUM 8 SCAN LINES !":GOTO 610
625 PRINT "NUMBER OF CHARACTER LINES: "
630 INPUT B
640 TR=(A)*TSL
650 VT=(B+1)*TR
660 IF VT<=20000 THEN 680
665 PRINT
670 PRINT " IMPOSSIBLE! "
675 PRINT "FEWER CHARACTER OR SCAN LINES, PLEASE. "
677 GOTO 600
680 Y=INT(20000/TR)
690 R(4)=Y-1
700 REM XXXXXXXXXX R5 XXXXXXXXXX
710 R(5)=INT((20000-Y*TR)/TSL)
800 REM XXXXXXXXXX R6 XXXXXXXXXX
810 R(6)=B
815 VD=R(6)*TR
900 REM XXXXXXXXXX R7 XXXXXXXXXX
910 R(7)=INT((((TR*Y+TSL*R(5))-(1500+B*TR))/2+B*TR)/TR)
915 VP=R(7)*TR
1000 REM XXXXXXXXXX R8 XXXXXXXXXX
1010 R(8)=0
1100 REM XXXXXXXXXX R9 XXXXXXXXXX
1110 R(9)=A-1
1200 REM XXXXXXXXXX R10 & R11 XXXXXXXXXX
1202 REM UNDERLINE CURSOR
1204 IF A=8 THEN R(11)=A :R(10)=64+A :GOTO 1300
1206 R(10)=73 :R(11)=9
1300 REM XXXXXXXXXX R12, R13, R14 & R15 XXXXXXXXXX
1310 R(12)=0
1320 R(13)=0
1330 R(14)=0
1340 R(15)=0
1350 PRINT :PRINT
1352 PRINT "SCREEN FORMAT = ";R(1);" X ";B
1354 PRINT: PRINT
1700 FOR Q=0 TO 15
1710 PRINT K$;" R";Q;
1720 PRINT TAB(20);" = ";
1727 Z2=R(Q)
1730 GOSUB 2000
1740 PRINT
1750 NEXT Q
1760 PRINT :PRINT:
1800 PRINT " CLOCK PERIOD                ";TC;L$
1810 PRINT " LINE SYNC. PULSE WIDTH      ";LPB;L$
1815 PRINT " LINE SYNC. PULSE PERIOD     ";TSL;L$
1830 PRINT " HORIZONTAL DISPLAY TIME     ";DT;L$
1840 PRINT " HORIZONTAL POSITION         ";HP;L$
1850 PRINT " CHARACTER LINE PERIOD       ";TR;L$
1855 VE=Y*TR+R(5)*TSL
1860 PRINT " RASTER SYNC. PERIOD         ";VE;L$
1865 PRINT " VERTICAL DISPLAY TIME       ";VD;L$
1867 PRINT " VERTICAL POSITION           ";VP;L$
1990 END
2000 REM XXXXXXXXXX DEC TO HEX XXXXXXXXXX
2010 PRINT "$";
2020 FOR Z=1 TO 0 STEP -1
2030 Z1=INT(Z2/16^Z)
2040 Z2=Z2-Z1*16^Z
2050 Z1=Z1+48
2060 IF Z1>57 THEN Z1=Z1+7
2070 PRINT CHR$(Z1);
2080 NEXT Z:RETURN
```

Tabel 1. Listing van het 6845-berekeningsprogramma.

enige speling te hebben om het venster te kunnen verschuiven.
De inhoud van R2 wordt dan
49/0,5 = 98.
In hexadecimale vorm wordt dat 62_{hex}.

Rasterfrekwentie
Voor de berekening van de rastergegevens moeten we eerst het aantal lijnen voor de opbouw van een karakter kennen. Bij de VDU-kaart is dat minimaal 8. In dat geval is er geen "witruimte" tussen twee onder elkaar staande regels. Dat aantal van 8 wordt gewoonlijk dan ook alleen gebruikt bij grafische toestanden. Het maximale aantal lijnen per regel bedraagt 25, terwijl men gewoonlijk uitgaat van 9. In het laatste geval kunnen 24 regels worden weergegeven op het scherm.
Bij 9 lijnen per regel (in R9 staat dit aantal minus één) is de regel-tijd:
9 x TSL = 9 x 64 μs = 576 μs,
en voor 24 regels wordt dat:
24 x 576 = 13824 μs.

Om wat speelruimte te houden tellen we hier een extra regeltijd bij:
13824 + 576 = 14400.
Deze tijd noemen we VT (vertikaal totaal). Als VT kleiner is dan 20000 (μs), dan is de gekozen kombinatie mogelijk. In R6 wordt daarna het aantal regels (24, 18_{hex}) gezet. Vervolgens kan het totale regel-aantal voor R4 worden berekend. We moeten zo dicht mogelijk in de buurt blijven van de rastertijd van 20 ms, zodat het mogelijke aantal regels wordt:
20000/576 = 34,72.
Dat wordt afgerond op 34. Bij ons gekozen formaat en frekwentie kunnen er dus 34 regels op het scherm worden geschreven, waarvan er 24 worden gebruikt. De inhoud van R4 wordt 34 minus 1, dat is 21_{hex}. In verband met de gemaakte afronding is de rastertijd niet exakt 20000 μs, maar
34 x 576 = 19584 μs.
Er ontbreken nog
20000 — 19584 = 416 μs.