Disk Copy

Wer hat sich als stolz Kopieren von Program

```
100 REM *** INITIALISIERUNG ***
110 POKE56,PEEK(46)+14:CLR:RB=255-PEEK(56):PA=1:AN=0:BL=0:NF$=""
120 PE=PEEK(45)+256*PEEK(46):MR=PE-135:MW=PE-79:MD=PE-24
130 DIMNF$(140),CF%(140),BL%(140),P%(10),AL%(90),AH%(90)
140 P%(0)=0:AL%(0)=0:AH%(0)=PEEK(56)-1
150 :
160 REM *** MENUE ***
170 PRINT"■"TAB(9)"***** DISK COPY *****":PRINTTAB(10)"VON D.WEINECK 2/84"
180 PRINT"■■■■■1.  DIRECTORY
190 PRINT"■■■2.  KOPIEREN
200 PRINT"■■■3.  FORMATIEREN
210 PRINT"■■■4.  ENDE
220 PRINTSPC(212)"■BITTE WAEHLEN SIE
230 GETDC$:DC=VAL(DC$):IFDC<1ORDC>4THEN230
240 ONDCGOTO910,270,700,670
250 :
260 REM *** KOPIEREN ***
270 PRINT"■■ORIGINALDISKETTE EINLEGEN"
280 GOSUB990
290 REM *** FILES EINLESEN ***
300 OPEN1,8,0,"$0"
310 GOSUB760:IFNF$<>""THEN340
320 IFST=0THEN310
330 GOTO350
340 BL%(AN)=ASC(BL$+CHR$(0)):NF$(AN)=NF$:IFST=0THENAN=AN+1:NF$="":GOTO310
350 CLOSE1:AN=AN-1:IFAN=0THENPRINT"■■■EERE DISKETTE■":GOSUB990:RUN
360 REM *** KOPIERAUSWAHL ***
370 PRINT"■■ANTWORTEN SIE MIT J/N■"
380 FORI=1TOAN:PRINTBL%(I);TAB(5)NF$(I)" ? ";:POKE198,0
390 WAIT198,1:GETA$:IFA$="J"THENCF%(I)=-1:BL=BL+BL%(I):PRINTTAB(30)"■ JA ■":GOTO
420
400 CF%(I)=0:IFA$<>"N"THEN390
410 PRINTTAB(30)"NEIN"
420 IFBL>RBTHENP%(PA)=I-1:PA=PA+1:BL=BL%(I)
430 NEXTI:P%(PA)=AN
440 IFBL=0THEN640
450 REM *** KOPIE ***
460 PRINT"■■KOPIE IN ARBEIT■"
470 FORI=1TOPA
480 FORRW=0TO1:NR=0:IFRW=1THENPRINT"■■IELDISK EINLEGEN":GOSUB990
490 FORJ=P%(I-1)+1TOP%(I)
500 IFNOTCF%(J)THENNEXTJ:GOTO540
510 NF$=NF$(J):PRINTBL%(J);TAB(5)NF$:GOSUB570:IFST=0ORST=64THEN530
520 GOSUB880:RUN
530 NEXTJ
540 NEXTRW:IFI=PATHEN640
550 PRINT"■■ORIGINALDISK EINLEGEN":GOSUB990
```

Zeile 100 bis 140

setzt zuerst die Speicherobergrenze für Basic herunter (Speicherstelle 56) und berechnet, wieviel Speicher zum Kopieren zur Verfügung steht (RB). Um spätere Erweiterungen einbauen zu können (zum Beispiel Backup für relative Dateien), arbeite ich hier nicht mit festen Zahlen, sondern mit Variablen, die das Programm selbst berechnet. Das gilt ebenso für die Startadressen der Maschinenroutinen. Diese befinden sich direkt hinter dem Basic-Teil und brauchen deshalb nicht erst über DATA-Zeilen bei jedem Programmlauf »eingepoked« werden. Das spart nicht nur Zeit, sondern vor allem auch Speicherplatz (zirka 700 Bytes). Dafür müssen Sie beim Abschreiben zwei Teile zusammenfügen. Im Initialisierungsteil werden auch alle Variablen und Arrays eingerichtet (siehe Variablentabelle).

Zeile 160 bis 240

enthält das Menü. Sie können hier jederzeit weitere Funktionen einfügen.

Wird der Programmteil »Formatieren« gewählt, springt das Programm in die Zeilen 700 bis 750. Interessant ist hier die Möglichkeit, bei der Frage nach der Disk — ID keine Eingabe zu machen, sondern »RETURN« zu drücken. Dies ist bei Disketten sinnvoll, die bereits formatiert sind, aber gelöscht werden sollen. Das DOS der 1541 löscht dann nur die BAM und die erste Seite des Directory, was viel schneller geht als vollständiges Neuformatieren. Das Programm schließt diesen Teil mit Ausgabe der Fehlermeldung ab und springt wieder ins Menü.

Der Programmteil »Directory« ermöglicht ein schnelles Einlesen des Directorys, natürlich ohne Programmverlust. Man kann sich so einen kurzen Überblick über Original- und Zieldiskette verschaffen, ohne die Kopierroutine aufrufen zu müssen. Hier wird ein Maschinenteilprogramm benutzt, um Speicherplatz und Zeit zu sparen.

Zeile 270 bis 650

Kommen wir nun zum Kern der Sache, dem eigentlichen Kopierteil. Dieser befindet sich in den Zeilen 270 bis 650. Im ersten Teil (Zeile 270 bis 350) werden alle Files, die auf der Diskette sind, in ein Stringarray (NF$(I)) eingelesen,

Schlimmer wird es, wenn es sich um Maschinenprogramme oder gar um sequentielle Files handelt. Das Anlegen einer Sicherheitskopie wird zur zeitraubenden, umständlichen Prozedur und unterbleibt deshalb, bis man eines Tages feststellt, daß das Original aus irgendwelchen Gründen nicht mehr läuft. Mir ist das mit einer Datei passiert, die aus über 400 Einträgen bestand und die ich eines Tages einfach nicht mehr einladen konnte. Seitdem gibt es bei mir von allen wichtigen Disketten Sicherheitskopien, bei deren Erstellung mir das folgende Programm gute Dienste leistet.

Das Besondere an diesem Programm ist, daß alle Arten von Files mit Ausnahme relativer Dateien kopiert werden können und das recht komfortabel und schnell. Der Trick besteht darin, daß die Files, die man kopieren möchte, der Reihe nach in den Speicher gelesen werden bis dieser voll ist. Da das Programm auch die »versteckten« RAM-Bereiche mitbenutzt, die man normalerweise gar nicht ansprechen kann, weil Basic-Interpreter und Betriebssystem an derselben Speicheradresse liegen, kommt man in einem Durchgang auf stattliche 226 Blöcke (zirka 56 KByte). Sollte das noch nicht ausreichen, startet das Programm einen zweiten oder auch dritten Durchgang. Dann ist spätestens die gesamte Diskette kopiert (bei ganz ungünstigen Verhältnissen wird noch ein vierter Durchgang gebraucht). Damit ergeben sich Zeiten von unter 20 Minuten für eine Komplettkopie.

Wie funktioniert's?

Sehen wir uns zuerst das Basicprogramm an (Bild 1): Ich habe es absichtlich in mehrere Teile zerlegt, um es übersichtlicher zu machen. Die REM-Zeilen können beim Eintippen natürlich ebenso wegfallen wie die Zeilen, in denen nur ein Doppelpunkt steht.

er Besitzer eines Commodore 64 und einer 1541-Floppy noch nicht beim men geärgert? Solange es sich um reine Basic-Programme handelt, geht es noch:

Originaldiskette einlegen, Programm laden, Diskette wechseln, Programm »saven«, Diskette wechseln, Programm laden ....

```
560 NEXTI:RUN
570 IFRW=1THEN610
580 OPEN1,8,5,NF$+",R":POKE252,0:POKE253,AH%(NR)+1
590 SYSMR:NR=NR+1:AL%(NR)=PEEK(254):AH%(NR)=PEEK(255)
600 CLOSE1:RETURN
610 OPEN1,8,5,NF$+",W":POKE252,0:POKE253,AH%(NR)+1
620 POKE254,AL%(NR+1):POKE255,AH%(NR+1):SYSMW
630 NR=NR+1:CLOSE1:RETURN
640 PRINT"■■■KOPIE FERTIG !
650 GOSUB990:RUN
660 REM *** ENDE ***
670 POKE56,160:END
680 :
690 REM *** FORMATIEREN ***
700 INPUT"■■■DISKNAME";FO$:ID$="":INPUT"■DISK-ID";ID$:IFID$<>""THENID$=","+ID$
710 FO$=FO$+ID$
720 PRINT"■■BITTE ZIELDISKETTE EINLEGEN"
730 GOSUB990
740 OPEN1,8,15,"N:"+FO$:CLOSE1
750 GOSUB880:GOTO170
760 REM DIRECTORY EINLESEN
770 GET#1,A$,B$
780 GET#1,BL$,B$
790 GET#1,A$
800 GET#1,B$:IFST<>0THENRETURN
810 IFB$<>CHR$(34)THEN800
820 GET#1,B$:IFB$<>CHR$(34)THENNF$=NF$+B$:GOTO820
830 GET#1,B$:IFB$=CHR$(32)THEN830
840 NF$=NF$+","+B$:FORI=0TO1:GET#1,B$:NF$=NF$+B$:NEXT
850 GET#1,B$:IFB$<>""THEN850
860 RETURN
870 REM *** FEHLER-AUSGABE ***
880 OPEN15,8,15:INPUT#15,A,B$,C,D:PRINTA;B$;C;D:CLOSE15:GOSUB990:RETURN
890 :
900 REM *** DIRECTORY ***
910 PRINT"■"
920 OPEN3,8,0,"$0":GET#3,A$,A$
930 GET#3,A$,A$,BL$,BH$
940 IFA$=""THENCLOSE3:GOTO980
950 BL$=BL$+CHR$(0):BH$=BH$+CHR$(0)
960 PRINT256*ASC(BH$)+ASC(BL$);
970 SYSMD:GOTO930
980 GOSUB 990:GOTO170
990 PRINTSPC(69)"_____":PRINTSPC(29)"■*TASTE*■"
1000 POKE198,0:WAIT198,1:GETA$:RETURN
READY.
```

Bild 1. Basicprogramm »Disk Copy«

```
Variablen - Liste

RB:     Anzahl der zur Verfuegung stehenden RAM - Bloecke
PA:     Anzahl der Durchgaenge beim Kopieren
AN:     Anzahl der Files auf der Diskette
BL:     Anzahl der zu kopierenden Bloecke
NF$:    Name und Typ des Files
PE:     Programmende
MR:     Adresse der Maschinenroutine zum Lesen
MW:     Adresse der Maschinenroutine zum Schreiben
MD:     Adresse der Maschinenroutine fuer Directory-Ausgabe
NF$(I): Feld fuer Filenamen
CF%(I): Feld fuer Kopierflags
BL%(I): Blocklaenge der einzelnen Files
P%(I):  Anzahl der Files in einzelnen Durchgaengen
AL%(I): Endadresse der einzelnen Files im Speicher (Low-Byte)
AH%(I): Endadresse der einzelnen Files im Speicher (High-Byte)
RW:     Flag fuer Lesen oder Schreiben
I,J:    Schleifenvariable
```

die zugehörigen Blocklängen in ein Variablenarray (BL%(I)). Das dauert etwas länger, weil der Speicherplatz begrenzt ist und der Basic-Interpreter mitunter eine Garbage-Collection (Müll-Sammlung) durchführt, um Platz zu schaffen. Die Anzahl der Files wird in der Variablen AN festgehalten. Ist die Diskette leer, weil man zum Beispiel noch die gerade formatierte Zieldiskette im Laufwerk hatte, springt das Programm ins Menü zurück.

In Zeile 370 bis 440 erfolgt die Auswahl, welche Files kopiert werden sollen. Das Programm schreibt dazu die einzelnen Namen mit der zugehörigen Blocklänge auf den Bildschirm und Sie können mit »J« oder »N« aussuchen. Die Entscheidung merkt sich das Programm wieder in einem Variablenarray (CF%(I)): Ist CF%=—1, wird kopiert, sonst nicht. Gleichzeitig wird in der Variablen BL aufaddiert, wieviele Blöcke zu kopieren sind, damit das Programm herausbekommt, ob ein Durchgang ausreicht oder nicht. Wenn Sie alle Files mit »N« kennzeichnen, springt das Programm wieder ins Menü.

Nachdem nun endlich alle Entscheidungen und Vorbereitungen abgeschlossen sind, geht es ans Kopieren (Zeile 460 bis 650). Der Reihe nach werden alle gekennzeichneten Files in den Speicher eingelesen. Dazu wird das Laufwerk mit »OPEN«-Befehlen angesprochen, weil so alle Arten von Files geladen und auch abgespeichert werden können. (Mit »LOAD« könnten nur Programmfiles geladen werden.) Erfreulicherweise enthält die Variable NF$ nicht nur den Namen des jeweiligen Files, sondern auch den Typ, also PRG, SEQ oder USR. Deshalb können alle Filetypen mit ein und derselben Routine verarbeitet werden.

Der Unterschied zwischen Lesen und Schreiben liegt lediglich darin, daß dem »OPEN«-Befehl im ersten Falle ein R (für Read), im zweiten ein W (für Write) angehängt wird. Die Datenübertragung selbst erledigt das Maschinenprogramm, dem wir uns gleich zuwenden werden. Um Variablen an diese Routinen übergeben zu können, benutzen wir die Zeropage-Speicherstellen 252 bis 255 ($FC bis $FF).

Werfen wir nun noch einen Blick auf die Maschinensprach-Teile (Bild 3 bis 5). Dabei soll vor allem erläutert werden, wie man den vollen RAM-Speicher des C 64 nutzen kann.

Dazu ist ein kurzer Blick auf die Speicheraufteilung des C 64 erforderlich, insbesondere den Teil ab $A000 bis $FFFF (dez. 40960 — 65535). Hier liegt normalerweise der Basic-Interpreter, der 8 KByte Adreßraum benötigt ($A000 - $C000). Darüber liegen in 4 KByte die Ein-Ausgabe-Einheiten ($D000 - $E000). Ganz oben im Speicher befindet sich das Betriebssystem, das genau wie der Basic-Interpreter 8 KByte belegt ($E000 - $FFFF). Zusätzlich ist aber der gesamte Bereich auch noch mit RAM bestückt. Woher weiß der Prozessor nun, was er benutzen soll? Lediglich 3 Bits in Speicherstelle 1 sind für die Auswahl zuständig: Bit 0 schaltet den Basic-Interpreter ein und aus, Bit 1 gleichzeitig Basic-Interpreter und Betriebssystem, Bit 2 ist für uns schon uninteres-

Disk Copy Disk Copy Disk Copy

```
100 PRINT"■■■■BASIC - DATA - LADER FUER 'DISK COPY
110 INPUT"■■ANFANGSADRESSE";AD:ED=AD+135
120 FORI=ADTOED-1
130 READZ:POKEI,Z:NEXT
140 HB=INT(ED/256):LB=EDAND255
150 PRINT"■■■LADEN SIE NUN DAS PROGRAMM 'DISK COPY'.
160 PRINT"■GEBEN SIE NACH DEM LADEN EIN:":PRINT"■■■POKE 45,"LB":POKE 46,"HB"
170 PRINT"■■DAS PROGRAMM BEFINDET SICH JETZT VOLL-
180 PRINT"■STAENDIG IM SPEICHER UND KANN GE'SAVED' ■WERDEN.
190 DATA162,1,32,198,255,160,0,32,207,255,120,170,165,1,41,252,133,1,138,145
200 DATA252,165,1,9,3,133,1,88,32,183,255,201,64,240,11,201,0,208,7,200,208
210 DATA221,230,253,208,217,32,204,255,132,254,165,253,133,255,96,162,1,32
220 DATA201,255,160,0,120,165,1,41,252,133,1,177,252,170,165,1,9,3,133,1,88
230 DATA138,32,210,255,32,183,255,201,0,208,17,200,208,2,230,253,165,255,197
240 DATA253,208,217,196,254,144,213,240,211,76,204,255,162,3,32,198,255,32
250 DATA207,255,32,210,255,208,248,169,13,32,210,255,76,204,255,0,0,0
READY.
```

Bild 2. Basic Lader für »Disk Copy«

sant. Werden beide, Bit 0 und Bit 1, auf 0 gesetzt, ist zusätzlich auch noch der Ein-Ausgabebereich abgeschaltet. Eigentlich doch ganz einfach.

Der Teufel steckt wie fast immer im Detail: Wenn der Basic-Interpreter abgeschaltet ist, wie soll dann ein Basic-Programm laufen? Mehr noch, ohne sein Betriebssystem ist der Prozes-

Merge

Kleben per Software!

Wenn Sie ein Unterprogramm haben — zum Beispiel ein Formatierungsprogramm zum Erstellen von Tabellen — und sie möchten es an ein vorhandes Basicprogramm anhängen, ohne es extra eintippen zu müssen — mit Merge eine einfache Sache!

Sie müssen lediglich das Ende des Basic-Programms finden (3 Bytes 0) und die Adresse der zweiten Null in den Zeiger für Basic-Anfang einschreiben. Dann können Sie ein zweites Programm laden und modifizieren, ohne daß das erste Programm beeinflußt wird. Wenn sie anschließend wieder die ursprüngliche Startadresse in den Basic-Pointer schreiben, haben Sie ein einziges Programm.

Leider ist es eine langwierige Angelegenheit, das Ende eines Programms zu suchen, die gefundene Adresse in den Basic-Pointer einzupoken..... Das Programm Merge übernimmt diese Arbeit — und das Rücksetzen des Basic-Pointers auch.

Wenn Sie ein Programm geladen haben, so brauchen sie nur SYS 50000 einzugeben und das »Kuppelprogramm« Merge meldet sich mit

```
1 rem  .........................
2 rem  .                       .
3 rem  .  merge 1.1     11/83  .
4 rem  .                       .
5 rem  .   von heinz boeffel   .
6 rem  .    kantstrasse  12    .
7 rem  .   6680 neunkirchen  7 .
8 rem  .                       .
9 rem  .........................
10 print chr$(14);chr$(147)
30 print"          ... merge 1.1 ..."
40 print"          von heinz boeffel";chr$(13);chr$(13)
50 print"Das Programm MERGE setzt den Zeiger"
60 print"fuer Basic-Anfang unmittelbar hinter"
70 print"das im Speicher befindliche Programm.";chr$(13)
80 print"Somit kann ein weiteres Programm hinter"
90 print"das bestehende geladen werden.";chr$(13)
100 print"Geben sie hierzu den Befehl SYS 50000"
110 print"ein. Das nachgeladene Programm kann"
120 print"genau wie das erstgeladene behandelt"
130 print"werden.";chr$(13)
150 print"Um den Basic-Zeiger wieder zurueck-"
160 print"zusetzen, geben Sie wieder SYS 50000"
170 print"ein. Die beiden Programme ergeben nun"
180 print"zusammen ein einziges Programm.";chr$(13)
190 print "Zum Laden bitte Leertaste druecken!"
200 get gS:if gS()" " then 200
210 print chr$(147);chr$(142)
220 for i=50000 to 50264:read q:poke i,q:next i
230 new
10000 data 169,255,133,2,165,43,201,1,208,13
10010 data 133,251,165,44,201,8,208,5,133,252
10020 data 76,125,195,165,251,133,43,165,252,133
10030 data 44,162,0,189,0,196,240,6,32,22
10040 data 231,232,208,245,96,160,0,177,43,208
10050 data 12,200,177,43,208,7,200,177,43,208
10060 data 2,133,2,230,43,208,2,230,44,165
10070 data 2,208,228,162,0,189,167,195,240,6
10080 data 32,22,231,232,208,245,96,13,13,32
10090 data 32,32,32,32,32,32,32,32,32,42
10100 data 42,42,32,77,69,82,71,69,32,49
10110 data 46,49,32,42,42,42,13,32,32,32
10120 data 32,32,32,32,32,32,32,86,79,78
10130 data 32,72,69,73,78,90,32,66,79,69
10140 data 70,70,69,76,13,13,32,32,32,32
10150 data 32,32,32,32,32,32,32,80,82,79,71
10160 data 82,65,77,77,32,79,78,32,72,79
10170 data 76,68,33,13,0,0,13,13,32,32
10180 data 32,32,32,32,32,32,32,32,42,42
10190 data 42,32,77,69,82,71,69,32,49,46
10200 data 49,32,42,42,42,42,13,32,32,32,32
10210 data 32,32,32,32,32,32,86,79,78,32
10220 data 72,69,73,78,90,32,66,79,69,70
10230 data 70,69,76,13,13,32,32,32,32,32
10240 data 32,32,32,32,32,32,80,82,79,71,82
10250 data 65,77,77,83,32,77,69,82,71,69
10260 data 68,33,13,0,0,0,0,0,0,0

ready.
```

Disk Copy Disk Copy Disk Copy

sor hilfloser als ein Blinder im Nebel.

Die Lösung liegt einfach darin, zu verhindern, daß der Prozessor überhaupt auf die Idee kommt, in sein Betriebssystem oder ins Basic hineinzuspringen. Letzteres ist nicht schwierig, denn wir befinden uns ja in einem Maschinenprogramm, wenn wir das Basic abschalten.

Und bei der Bearbeitung eines solchen Programms kann nur dann etwas passieren, wenn der Prozessor das Programm verläßt. Das tut er allerdings jede 1/60 Sekunde, zum Beispiel um die interne Uhr weiterzustellen und nachzusehen, ob eine Taste gedrückt wurde etc. Das ist die sogenannte Interrupt-Routine. Wenn wir ihm die sperren, kann eigentlich gar nichts mehr schiefgehen. Und es funktioniert tatsächlich:

Im Leseteil des Maschinenprogramms wird zuerst der mit »OPEN« eröffnete Kanal als Eingabekanal gesetzt (FFC6). Dann wird ein Byte über diesen Kanal geholt. Jetzt sperrt der SEI (Set Interrupt) die Interruptroutine und wir können in aller Ruhe schalten und walten. Wir schieben das Byte ins X-Register, legen die beiden unteren Bits der Speicherstelle 1 auf 0, holen unser Byte aus dem X-Register und legen es im RAM ab. Jetzt setzen wir die Bits wieder auf 1, löschen die Interrupt-Sperre, und alles ist wieder in Ordnung. Nachdem unser Byte im RAM sicher untergebracht ist, fragen wir nun ab, ob vielleicht ein Fehler aufgetreten ist (FFB7) oder das Ende unseres Files erreicht ist. Wenn ja, springen wir ins Basic zurück und brechen im Fehlerfalle das Programm mit einer entsprechenden Meldung ab. Wenn nein, wenden wir uns dem nächsten Byte zu, das übertragen werden soll und behandeln es mit der gleichen Sorgfalt. Ganz zum Schluß müssen wir noch wieder die Kanäle zurücksetzen (Tastatur als Eingabe, Bildschirm als Ausgabe (FFCC)). Das alles klingt zwar umständlich und langwierig, geht aber in Wirklichkeit unglaublich schnell.

Das Schreiben auf Diskette bringt nichts grundsätzlich Neues, der ganze Vorgang läuft hier einfach andersherum ab. Unser Kanal 1 ist jetzt Ausgabekanal ($FFC9), und anstatt ein Byte von der Diskette zu holen, geben wir es aus ($FFD2).

Auch die Directory-Ausgabe folgt diesem Muster: Kanal 3 als Eingabe setzen ($FFC6), Zeichen für Zeichen holen ($FFCF) und — jetzt auf dem Bildschirm — ausgeben ($FFD2).

```
LESEN VON DISKETTE
,114D  A2 01     LDX #01
,114F  20 C6 FF  JSR FFC6
,1152  A0 00     LDY #00
,1154  20 CF FF  JSR FFCF
,1157  78        SEI
,1158  AA        TAX
,1159  A5 01     LDA  01
,115B  29 FC     AND #FC
,115D  85 01     STA  01
,115F  8A        TXA
,1160  91 FC     STA  (FC),Y
,1162  A5 01     LDA  01
,1164  09 03     ORA #03
,1166  85 01     STA  01
,1168  58        CLI
,1169  20 B7 FF  JSR FFB7
,116C  C9 40     CMP #40
,116E  F0 0B     BEQ 117B
,1170  C9 00     CMP #00
,1172  D0 07     BNE 117B
,1174  C8        INY
,1175  D0 DD     BNE 1154
,1177  E6 FD     INC  FD
,1179  D0 D9     BNE 1154
,117B  20 CC FF  JSR FFCC
,117E  84 FE     STY  FE
,1180  A5 FD     LDA  FD
,1182  85 FF     STA  FF
,1184  60        RTS
```

Bild 3. Für Interessierte: Maschinenroutine: »Lesen von Disk«

```
SCHREIBEN AUF DISKETTE
,1185  A2 01     LDX #01
,1187  20 C9 FF  JSR FFC9
,118A  A0 00     LDY #00
,118C  78        SEI
,118D  A5 01     LDA  01
,118F  29 FC     AND #FC
,1191  85 01     STA  01
,1193  B1 FC     LDA  (FC),Y
,1195  AA        TAX
,1196  A5 01     LDA  01
,1198  09 03     ORA #03
,119A  85 01     STA  01
,119C  58        CLI
,119D  8A        TXA
,119E  20 D2 FF  JSR FFD2
,11A1  20 B7 FF  JSR FFB7
,11A4  C9 00     CMP #00
,11A6  D0 11     BNE 11B9
,11A8  C8        INY
,11A9  D0 02     BNE 11AD
,11AB  E6 FD     INC  FD
,11AD  A5 FF     LDA  FF
,11AF  C5 FD     CMP  FD
,11B1  D0 D9     BNE 118C
,11B3  C4 FE     CPY  FE
,11B5  90 D5     BCC 118C
,11B7  F0 D3     BEQ 118C
,11B9  4C CC FF  JMP FFCC
```

Bild 4. In Assembler: Schreiben auf Diskette

```
DIRECTORY HOLEN
,11BC  A2 03     LDX #03
,11BE  20 C6 FF  JSR FFC6
,11C1  20 CF FF  JSR FFCF
,11C4  20 D2 FF  JSR FFD2
,11C7  D0 F8     BNE 11C1
,11C9  A9 0D     LDA #0D
,11CB  20 D2 FF  JSR FFD2
,11CE  4C CC FF  JMP FFCC
```

Bild 5. Laden des Directorys

Wichtige Bedienungshinweise

So, nun steht dem Eintippen des Programms nichts mehr im Wege. Noch ein paar wichtige Hinweise: Die beiden Teile des Programms müssen beim ersten Mal zusammengefügt werden. Dazu gehen wir folgendermaßen vor:

1. Tippen Sie das Programm »Disk Copy« ab und speichern Sie es auf Diskette.

2. Starten Sie das Programm mit »RUN« und drücken Sie die »RUN/STOP«-Taste, wenn das Menü erscheint.

3. Geben sie ein: »PRINT PR« und schreiben Sie sich die angezeigte Zahl auf.

4. Tippen Sie das Programm »Basic-Data-Lader« ein und starten Sie es. Auf die Frage nach der Anfangsadresse geben Sie Ihre aufgeschriebene Zahl ein.

5. Folgen Sie genau den Anweisungen des Programs und geben Sie die beiden »POKE«-Befehle ein.

6. Speichern Sie das vollständige Programm auf Diskette ab.

Jetzt haben Sie das Programm gebrauchsfähig auf Diskette. Sie können auch beliebige Änderungen am Programm durchführen, der Maschinensprach-Teil wird sich immer automatisch mitverschieben.

Anpassung auf VC 20:

Das Programm läuft auch auf dem VC 20, für den ich es ursprünglich geschrieben hatte. Nur Zeile 110 muß geändert werden:

110 POKE56,PEEK(46)+14: CLR:RB=PEEK(644)—PEEK(56):...

Wenn Sie mit einer 1541-Floppy arbeiten, sollten Sie noch einfügen:

115 OPEN1,8,15,"UI-": CLOSE1. Und nun viel Spaß beim Kopieren.

(Dietrich Weineck)

MERGE 1.1
VON HEINZ BOEFFEL
PROGRAMM ON HOLD!

Dann können Sie ein weiteres Programm (nur mit höheren Zeilennummern als das erste!) laden.

Das kann auch beispielsweise das Directory einer Diskette sein oder irgendwas, was Sie gerade ausprobieren möchten, sein. Mit NEW läßt sich dieses gleich wieder löschen.

Geben Sie dann wieder SYS 50000 ein und Merge meldet sich mit

MERGE 1.1
VON HEINZ BOEFFEL

PROGRAMMS MERGED!

Falls Sie das zweite Programm nicht durch NEW gelöscht haben, so ist aus den beiden Programmen ein einziges geworden. Haben Sie allerdings das zweite Programm vor dem Befehl SYS 50000 gelöscht, so erhalten Sie wieder den Ausgangszustand.

Merge ist vollständig in Maschinensprache geschrieben — daher beträgt die Ablaufzeit nur wenige Augenblicke.

Somit stellt Merge ein nützliches Hilfsmittel dar, das viele einzelne Eingaben (vor allem bei immer wieder verwendeten Unterprogrammen oder beim »Stricken« längerer Programme, die Sie zunächst abschnittweise abspeichern) erspart.

(Heinz Böffel)