

**Angerhausen · Englisch
Gerits · Thrun**

64

Tips & Tricks

BAND 1
Der Bestseller mit
über 70.000
verkauften
Exemplaren

**Eine Fundgrube für den
Commodore 64 Anwender**

EIN DATA BECKER BUCH

Angerhausen · Englisch
Gerits · Thrun

64

Tips & Tricks

BAND 1

**Eine Fundgrube für den
Commodore 64 Anwender**

EIN DATA BECKER BUCH

ISBN 3-89011-001-0

4. erweiterte und überarbeitete Auflage

Copyright (C) 1984 DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technische Angaben und Programme in diesem Buch wurden von dem Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

INHALTSVERZEICHNIS

1.	Vorwort	1
2.	Graphik für Fortgeschrittene	
2.1	Graphik auf dem Commodore 64	3
2.2	3D Graphik - BASIC-Programm	8
2.3	Farbige Balkengraphik	14
2.4	Definition eines eigenen Zeichensatzes	21
2.5	Modifikation des Zeichensatzes mit dem Joystick.	24
2.6	Der geteilte Bildschirm	32
2.7	Soft-Scrolling	38
2.8	Die Tastaturbelegung und ihre Änderung	40
3.	Dateneingabe mit Komfort	
3.1	Cursorpositionierung und Abfrage der Cursorposition	45
3.2	Cursor ein- und ausschalten	47
3.3	Repeatfunktion für alle Tasten	49
3.4	Der WAIT-Befehl: Warten auf einen Tastendruck	50
3.5	Die Belegung der Funktionstasten	52
3.6	Eine komfortable INPUT-Routine	59
3.7	Die "Maus" auf dem 64er: Simulation mit dem Joystick	67

4. BASIC für Fortgeschrittene

4.1	Oft versucht, selten gelungen:	
	Erzeugen einer BASIC-Zeile in BASIC	75
4.2	Kopieren des BASIC-Interpreters ins RAM	83
4.3	Keine negativen Zahlen mehr bei der FRE-Funktion .	85
4.4	Rückkehr ins BASIC-Programm nach LIST	87
4.5	GOTO, GOSUB und RESTORE	
	mit berechneten Zeilennummern	90
4.6	Der MID\$-Befehl	94
4.7	INSTR und STRING\$-Funktion	101
4.8	Automatische Zeilennummerierung	108
4.9	DEF FN - das unbekannte Wesen	112
4.10	Ihr Commodore 64 spricht deutsch	115
4.11	Verwendung einer HARDCOPY-Routine für	
	kommerzielle Programme	121
4.12	Mengenlehre auf dem CBM 64 am Beispiel	
	der Berliner Ku'damm-Uhr unter Verwendung	
	der Echtzeituhr und von Sprites	123
4.13	Ein kleiner Kopierschutz	130

5 Der CBM 64 kann nicht nur BASIC

5.1	Die Programmierung von FORTH	136
5.2	Vergleichsprogramm FORTH - BASIC	139
5.3	Weitere Sprachen: PASCAL, LOGO	148
5.4	ADA für den Commodore 64	150

6. CP/M auf dem Commodore 64

6.1	Das ist CP/M	152
6.2	Der Umgang mit den einzelnen CP/M Programmen . .	163

6.3	Die Anpassung von CP/M Standardsoftware an den 64er	168
6.4	Die Speicherverwaltung des Z80 Prozessors	169
6.5	Die Diskettenverwaltung unter CP/M	171
6.6	Die Zusammenarbeit der Prozessoren 6510 - Z80 . .	174
6.7	Kommentiertes BIOS-Listing	176
6.8	Implementierung eigener Ein-/ Ausgabefunktionen ins BIOS	221
6.9	Übertragung von Programmen und Daten vom CP/M ins Commodore-BASIC und umgekehrt	223
7.	Anschluß- und Erweiterungsmöglichkeiten des Commodore 64	
7.1	Eine sinnvolle Anwendung des USER-Port am Beispiel eines Centronics-Druckers	227
7.2	Datenübertragung von und zu anderen Rechnern mittels USER-Port	237
7.3	Der Expansionport: Eine Fallstudie mit der CP/M-Cartridge	245
7.4	Synthesizer in Stereo	253
8	Dateiverwaltung: Kein Buch mit sieben Siegeln	230
8.1	Cassette - Diskette	259
8.2	Das Prinzip der Dateiverwaltung: Sequentielle Dateien	266
8.3	Kopieren von Dateien mit einem und zwei Laufwerken	280
8.4	So geht's schneller: Relative Dateien	284
8.5	Eine andere Methode: Direktzugriff	304
8.6	Retten einer nicht ordnungsgemäß geschlossenen Datei	308
8.7	Der Blockverfolger	314

9. Poke's und andere nützliche Routinen

9.1	Der Kassettenpuffer als Programmspeicher	319
9.2	Sortieren von Stringfeldern	322
9.3	Minimum und Maximum von numerischen Feldern . .	327
9.4	DUMP - Ausgabe sämtlicher Variablen und ihrer Werte	334
9.5	Modifizierte PEEK-Funktion	339
9.6	Multitasking auf dem Commodore 64	343
9.7	POKE's und die Zeropage	353
9.8	Kontrolle von Texteingaben über die Tastatur . .	355
9.9	Formatiertes Programmlisting	359
9.10	Retten von Variablen und Warmstart	362

VORWORT

Tausende neuer Freunde gewinnt der Commodore 64 täglich weltweit. Das ist nicht weiter verwunderlich, denn der 64 bietet nicht nur hervorragende Leistungsdaten, sondern auch ein hervorragendes Preis-/Leistungsverhältnis. Für weit unter 2000 Mark bekommt man jetzt schon einen Commodore 64 einschließlich Diskettenlaufwerk. Wer noch das neue CP/M-Modul hinzurechnet, liegt immer noch unter 2000 Mark. Und das dann für ein komplettes Computersystem, das nicht nur die beiden gängigsten Prozessoren 6502 und Z-80 sondern auch die beiden meistverbreiteten Betriebssysteme CP/M und COMMODORE BASIC beinhaltet. Der Commodore 64 trägt zwar einen Einstiegspreis, doch nur zum Einsteigen und zum Spielen ist er viel zu schade. Vielmehr bietet er dem engagierten Hobbyprogrammierer ein fast grenzenloses Betätigungsfeld und läßt sich ohne weiteres auch für kleinere kommerzielle und technisch-wissenschaftliche Anwendungen einsetzen. Hier setzt 64 TIPS & TRICKS, unser zweites Buch zum Commodore 64 an. Unser bewährtes Autorenteam, bestehend aus Klaus Gerits, Lothar Englisch und Michael Angerhausen, hat wieder tief in die Trickkiste gepackt. Anhand zahlloser Beispiele und Musterprogramme möchten Ihnen die Autoren Anregungen für Ihre eigene Arbeit mit dem Commodore 64 geben. Auch dieses Buch soll Ihnen helfen, mehr aus Ihrem Commodore 64 zu machen. Alle drei Autoren kennen den 64 wie ihre Westentasche. Klaus Gerits z.B. hat IEC-BUS 64 und MAXI 64 entwickelt, von Lothar Englisch stammen PROFI-MON 64 und PROFI-ASS 64. Beide haben sich ausführlich mit technischen und systemnahen Eigenschaften des 64 befaßt. Zu den Ergebnissen gehören z.B. Projekte mit dem User Port und eine ausführliche Behandlung des CP/M Moduls, selbstverständlich mit ausführlich dokumentiertem BIOS-Listing. Michael Angerhausen, unser Datenbank-Spezialist, hat sich ausführlich mit dem 64 unter dem Aspekt der Dateiverwaltung befaßt. So zeigt er unter anderem, wie man auch mit BASIC 2.0 relative Dateien aufbauen kann.

Freuen würden sich unsere Autoren über Kommentare zu diesem Buch und Anregungen für weitere Arbeit. Hier auch gleich eine Antwort

auf die oft gestellte Frage: Kann man eigentlich an DATA BECKER Büchern mitarbeiten ? - Man kann ! Gerne nehmen wir neue Autoren auf, ganz gleich ob nur mit einem Beitrag oder gar mit einem komplett eigenen Buch.

Viel Spaß bei der Lektüre von 64 TIPS & TRICKS.

Vorwort zur erweiterten und überarbeiteten 2. Auflage

Innerhalb von wenigen Monaten waren die ersten 10.000 Exemplare von 64 Tips & Tricks vergriffen. Wir haben die Gelegenheit der Neuauflage genutzt, um dieses beliebte Buch zu überarbeiten und in wesentlichen Teilen zu erweitern.

Zu allen Maschinenprogrammen und -routinen von 64 TIPS & TRICKS finden Sie jetzt zusätzlich einen BASIC-Loader. Mit Hilfe dieser einfachen BASIC-Programme können jetzt auch alle Nicht-Maschinenprogrammierer die Programme in Maschinensprache leicht eingeben und voll nutzen. Der jeweilige BASIC-Loader überprüft das Programm auch auf Eingabefehler.

Vorwort zur erweiterten und überarbeiteten 4. Auflage

Wir haben durch die intensive Arbeit mit dem Commodore 64 noch einige interessante Dinge herausgefunden, die wir Ihnen nicht vorenthalten möchten. Neue Themenbereiche wie ADA, Kopierschutz und Block-Verfolger wurden von Herrn Frank Thrun hinzugefügt.

Düsseldorf, im Oktober 1984

Die Autoren

KAPITEL 2: GRAPHIK FÜR FORTGESCHRITTENE

2.1 GRAPHIK AUF DEM COMMODORE 64

Jeder Anwender des Commodore 64 hat natürlich den Wunsch früher oder später mit der eingebauten Graphik dieses Computers zu arbeiten. Leider erfährt er im Handbuch nur sehr kurz und knapp näheres über die Fähigkeiten und Möglichkeiten, die man mit dem Commodore 64 hat.

Wir wollen an dieser Stelle etwas ausführlicher auf die graphischen Möglichkeiten und Besonderheiten eingehen.

Zunächst muß man beim Commodore 64 zwischen der normalen Graphik, die über die Tastatur erreichbar ist (also die Block- oder Liniengraphik), der hochauflösenden Graphik und den Sprites unterscheiden. Blockgraphik und hochauflösende Farbgraphik bieten nur wenige Rechner an - aber die SPRITES sind eine wirkliche Neuerung auf dem Commodore 64. Diese SPRITES sind bisher nur von den vielen Spielautomaten bekannt. Man kennt dort ja die fantastischen Spiele, ob nun Abenteuer, SPACE-Games oder Action-Spiele. Und genau diese Möglichkeiten bietet uns heute auch der Commodore 64.

Auf den nächsten Seiten wollen wir nun auf die drei verschiedenen Grafik-Modi eingehen. Natürlich wird auch hier wieder die Theorie an Hand von Beispielen unterstützt.

1. Die Block- oder Liniengraphik der Tastatur.

Diese Möglichkeit der Erzeugung von Graphik auf dem Commodore 64 ist die einfachste und bequemste Art. Hier müssen nicht irgend welche Adressen berechnet oder auf verschiedene Register geachtet werden. Man kann im Programm von der Tastatur aus verschiedene Symbole erzeugen und diese direkt darstellen. Um diese Symbole zu erreichen sind jeweils zwei Tasten zu drücken. Wenn Sie sich die Tastatur des Commodore 64 genauer ansehen, so werden Sie feststellen, daß fast jede Taste des Gerätes zweifach belegt ist. Das Symbol, bzw. das Graphikzeichen auf der linken Seite der Taste wird durch gleichzeitiges Drücken der C= (CBM) Taste und der entsprechenden Taste, auf welcher sich das Symbol befindet, erreicht.

Man kann bei der Eingabe eines Programmes diese Zeichen immer innerhalb einer PRINT oder INPUT Anweisung eingeben. Dazu schreibt man zum Beispiel:

```
100 PRINT "
```

und drückt dann die Tasten C= und A. Sie sehen nun die obere rechte Ecke eines Rahmens auf dem Bildschirm. Um einen ganzen Rahmen zu erzeugen sind folgende weitere Eingaben notwendig (wir befinden uns immer noch in der Zeile 100):

Drücken Sie 38 mal die Tasten SHIFT und *. Sie sehen nun wie Sie von der Ecke des Rahmens aus eine gerade Linie auf der selben Höhe zeichnen. Außerdem haben Sie nun auch gelernt, daß durch drücken von SHIFT und einer anderen Taste das rechte Zeichen auf dieser Taste eingegeben werden kann. Nun drücken Sie noch einmal die Tasten C= und S und schon ist der obere Teil eines Rahmens fertiggestellt. Zum Abschluß der Zeile 100 geben Sie nun noch ein:

```
"; und drücken die RETURN-Taste
```

die nächste Zeile können Sie nun folgendermaßen eingeben:

```
110 PRINT "
```

Danach drücken Sie: 1 mal SHIFT und -, 38 mal die Leertaste und noch einmal SHIFT und -. Zum Abschluß geben Sie wieder ein:

```
"; und drücken die RETURN-Taste.
```

Schon ist die zweite Zeile des Rahmens fertig gestellt. Die dritte und letzte Zeile schreiben Sie nun folgendermaßen:

```
120 PRINT "
```

und danach 1 mal C= und Z, 38 mal SHIFT und * und 1 mal C= und X. Nun haben Sie den ganzen Rahmen, bestehend aus 3 Zeilen fertiggestellt. Geben Sie nun noch folgende Programmzeilen ein:

```
99 PRINT CHR$(147);: REM LOESCHEN DES BILDSCHIRMS
132 A$="
135 REM A$=38 MAL LEERTASTE DRÜCKEN
140 B$="DIESES ZEILE FUELLT UNSEREN RAHMEN AUS"
150 PRINT CHR$(19);
160 PRINT CHR$(17);CHR$(29);A$;
170 PRINT CHR$(19);
180 PRINT CHR$(17);CHR$(29);B$;
190 FOR I=1 TO 1000: NEXT
200 GOTO 150
```

Alle kommerziellen Programme werden heute durch die Anwendung solcher Rahmen sehr übersichtlich und benutzerfreundlich gestaltet. Wie Sie sehen kann auch der Commodore 64 programmiert werden wie die "Großen".

Natürlich gibt es auch eine andere Möglichkeit der Eingabe solcher Graphik-Symbole. Diese Symbole lassen sich allesamt auch über die CHR\$-Funktion erreichen. Zum Beispiel in unserem Programm:

```

100 A1$=CHR$(176): A2$=CHR$(174)
101 REM DIE LINKE UND RECHTE OBERE ECKE
102 A3$=CHR$(173): A4$=CHR$(189)
103 REM DIE LINKE UND RECHTE UNTERE ECKE
104 H1$=CHR$(96)
105 REM DER WAAGERECHTE STRICH
106 H2$=CHR$(125)
107 REM DER SENKRECHTE STRICH
108 H3$=CHR$(32)
109 REM DAS LEERZEICHEN
110 Z1$=A1$
111 FOR I=1 TO 38
112 Z1$=Z1$+H1$
113 NEXT I
114 Z1$=Z1$+A2$
115 REM 1. ZEILE DES RAHMEN
116 Z2$=H2$
117 FOR I=1 TO 38
118 Z2$=Z2$+H3$
119 NEXT I
120 Z2$=Z2$+H2$
121 REM 2. ZEILE DES RAHMEN
122 Z3$=A3$
123 FOR I=1 TO 38
124 Z3$=Z3$+H1$
125 NEXT I
126 Z3$=Z3$+A4$
127 REM 3. ZEILE DES RAHMEN
128 PRINT Z1$;
129 PRINT Z2$;
130 PRINT Z3$;

```

Wenn Sie diese neuen Zeilen eingeben und das Programm starten, werden Sie das selbe Ergebnis haben wie vorher. Der Vorteil liegt einfach darin, das Programme die mit solchen CHR\$-Funktionen geschrieben sind leichter zu lesen und zu ändern sind.

2. Die Benutzung von SPRITES

Ihr Commodore 64 kann aber mehr als nur einfache Linien zeichnen. Er bietet Ihnen wie schon zu Beginn gesagt Möglichkeiten, die bisher den großen Spielautomaten vorbehalten waren.

Der Commodore 64 ist in der Lage bis zu 8 verschiedene "Kleingraphiken" selbstständig zu verwalten. Diese Kleingraphiken nennen wir SPRITES. Jedes dieser maximal 8 Sprites können Sie unabhängig voneinander durch einfache POKES bewegen, löschen, kollidieren lassen und vieles mehr. Um aber mit SPRITES sinnvoll arbeiten zu können, müssen Sie sich ausführlich mit den entsprechenden Registern des Commodore 64 beschäftigen. Die kompletten Registerpläne finden Sie im Buch "64 intern". Wir wollen an dieser Stelle nur kurz auf die verschiedenen Möglichkeiten der Sprite-Manipulation eingehen.

Wie schon gesagt stehen uns für jedes Sprite verschiedene Register zur Verfügung. Es wäre vorteilhaft, wenn Sie zunächst einmal mit dem SPRITES experimentieren, welches Sie im Commodore Handbuch vorfinden.

Die wichtigste Adresse bei der Benutzung der Sprites ist die Zahl 53248. Diese Adresse gibt die den Start des Video-Controllers an. Der eingebaute VIC (Video-Controller) 6569 arbeitet also mit diesen Adressen. Um ein Sprite nun beispielsweise zu positionieren, müssen wir dem VIC-Chip mitteilen, an welcher Stelle das Sprite zu zeichnen ist. Das Register hierfür ist das Register 0 (oder genauer gesagt die Adresse 53248). In dieser Adresse finden wir die horizontale, und in Register 1 (53249) die vertikale Position des Sprites.

POKE 53248,160: POKE 53249,120

Es genügen also 2 POKES um eine ganze Kleingraphik an eine bestimmte Bildschirmposition zu setzen. Das Sprite in die Mitte des Bildschirms setzen. Die Register 0 und 1 dienen hierbei dem Sprite 1, die Register 2 und 3 dem Sprite 2 usw. Nach dem selben Prinzip arbeiten fast alle anderen Register. Genaueste Information über die Arbeit mit den Sprites finden Sie wie schon erwähnt in "64 intern". Auf den nächsten Seiten erfahren Sie nun wie man auch aufwendige Graphiken ohne große Mühe erstellen kann.

2.2 3D-GRAPHIK - BASIC PROGRAMM

Gleich zu Beginn unseres Buches wollen wir Ihnen ein BASIC-Programm vorstellen, das mit Hilfe der hochauflösenden Farbgraphik des Commodore 64 dreidimensionale Darstellungen auf den Bildschirm bringt. Das Programm benutzt die Befehle der Supergraphik 64; im Anschluß finden Sie jedoch die erforderlichen Änderungen, falls Sie nicht über dieses Programm verfügen.

Das Programm stellt eine Funktion dar, die Sie in Zeile 100 des Programms definieren können. Die Funktion kann in drei verschiedenen Arten dargestellt werden:

Erstens in einem normalen kartesischen (rechtwinkligen) Koordinatensystem, so wie Sie die Funktion auch auf Millimeterpapier zeichnen würden.

Zum zweiten ist die Darstellung in Polarkoordinaten (Angabe von Radius und Winkel) möglich.

Am interessantesten ist zweifellos die dreidimensionale Darstellung. Die Funktion wird dazu um die (senkrechte) Y-Achse gedreht. Wegen der Vielzahl der zu berechnenden Punkte braucht diese Art der Darstellung jedoch die längste Zeit. Die Parameter der Darstellung können Sie weitestgehend beeinflussen.

Nun zur Beschreibung des Programms. Zuerst können Sie die Darstellungsart auswählen (Zeile 40-70). Bei der kartesischen und Polardarstellung werden Sie dann nach dem Funktionsinkrement gefragt (Zeile 260). Das ist der Wert, um den der Parameter der waagerechten Achse nach jedem berechneten Punkt erhöht wird. Zeile 270 und 280 fragen nach dem Faktor für X- und Y-Achse. Damit können Sie das Maßstabsverhältnis der Achsen bestimmen. Für den Anfang können Sie beide Male 1 eingeben. Bei der horizontalen und vertikalen Verschiebung

(Zeile 370-410) können wir vorerst je Null eingeben. In Zeile 430 wird auf Graphikmodus umgeschaltet. Die Zeilen 450 bis 560 zeichnen das Achsenkreuz und Skalierung. Die Zeilen 680 bis 790 zeichnen die Polardarstellung, von Zeile 820 bis 970 steht die rechtwinklige Darstellung.

Ab Zeile 1010 steht die dreidimensionale Darstellung. Zuerst können Sie wieder Maßstab und Lage der Darstellung bestimmen; geben Sie für den Anfang die vorgeschlagenen Werte 20 bzw. 90 ein. Für die dreidimensionale Darstellung muß die Funktion in Zeile 100 mehrere zehntausend Male berechnet werden; das Programm benötigt dazu je nach Funktion ca. eine halbe bis mehrere Stunden.

Lassen Sie das Programm einmal mit verschiedenen Funktionen laufen; als Anregung hier ein paar Funktionen, die interessante Darstellungen ergeben.

```
100 DEF FN R(Q) = COS (2*Q) + COS ((Q + BB)/16)
100 DEF FN R(Q) = SQR (ABS(.5*(16-Q*Q)) + 1/(Q+4))
100 DEF FN R(Q) = COS (4*Q) + 20/(Q*Q + 3)
```

Haben Sie keine Supergraphik, müssen Sie folgende Änderungen und Ergänzungen im Programm durchführen:

Zeile 5 POKE 56,32 : CLR

Zeile 430 und 1400: Jeweils GOSUB 2000

Zeile 470 FOR A1=0 TO 199 : AX=F: AY=A1: GOSUB 3000 : NEXT

Zeile 480 FOR A1=0 TO 319 : AX=A1: AY=E: GOSUB 3000 : NEXT

Zeile 500 FOR A1=E-1 TO E+1: AX=XR: AY=A1: GOSUB 3000: NEXT

Zeile 520 FOR A1=E-1 TO E+1: AX=XL: AY=A1: GOSUB 3000: NEXT

Zeile 540 FOR A1=F-1 TO F+1: AX=A1: AY=YD: GOSUB 3000: NEXT

Zeile 560 FOR A1=F-1 TO F+1: AX=A1: AY=YU: GOSUB 3000: NEXT

Zeile 770 AX=XX: AY=YY: GOSUB 3000

Zeile 900 AX=G: AY=YY: GOSUB 3000

Zeile 1600 AX=X1: AY=Y1: GOSUB 3000

```

Zeile 1620 GOSUB 4000 : RETURN
Zeile 2000 FOR A1=8192 TO 16191 : POKE A1,0 : NEXT
Zeile 2010 FOR A1=1024 TO 2023 : POKE A1,16: NEXT
Zeile 2020 POKE 53248+17, 27+32 : POKE 53248+24, 16+8
Zeile 2030 RETURN
Zeile 3000 OY=320*INT(AY/8)+(AYAND7)
Zeile 3010 OX=8*INT(AX/8)
Zeile 3020 MA=2*((7-AX)AND7)
Zeile 3030 AV=8192+OY+OX
Zeile 3040 POKE AV,PEEK(AV) OR MA : RETURN
Zeile 4000 FOR A1=Y1+1 TO 199: AX=X1: AY=A1: GOSUB 5000: RETURN
Zeile 5000 OY=320*INT(AY/8)+(AYAND7)
Zeile 5010 OX=8*INT(AX/8)
Zeile 5020 MA=2*((7-AX)AND7)
Zeile 5030 AV=8192+OY+OX
Zeile 5040 POKE AV,PEEK(AV) AND NOT MA : RETURN

```

Durch die Programmierung der Graphikfunktionen in BASIC wird das Programm bedeutend langsamer gegenüber der Programmierung mit der Supergraphik 64.

```

10 PRINT"GRAFISCHE DARSTELLUNG VON FUNKTIONEN"
20 PRINT"DEFINIERT IN ZEILE 100"
40 PRINT"1 - KARTESISCHE DARSTELLUNG"
50 PRINT"2 - POLAR-KOORDINATEN"
60 PRINT"3 - 3D-DARSTELLUNG"
70 INPUT" AUSWAHL: 1 2 3 ";PL
100 DEF FNR(Q)=COS(Q)+COS(2*Q)+COS(5*Q)
210 IF PL=3 THEN 1010
250 PRINT:PRINT
260 INPUT"FUNKTIONS-INKREMENT = ";IK

```



```

270 INPUT "FAKTOR FUER X-ACHSE =";S1
280 INPUT "FAKTOR FUER Y-ACHSE =";S2
370 PRINT "URSPRUNG LINKS ODER RECHTS VER-
RSCHIEBEN"
380 INPUT "ZAHL VON -130 BIS 130 ";C
400 PRINT "URSPRUNG HOCH ODER RUNTER VER-
SCHIEBEN"
410 INPUT "ZAHL VON -90 BIS 90 ";D
430 GOTO 1
450 E=100+D:F=160+C
470 F=0TOF,199
480 GOTO E TO 319,E
490 FORXR=FTO319STEP19*S1
500 XR,E-1TOXR,E+1
510 FORXL=FTO0STEP-19*S1
520 XL,E-1TOXL,E+1
530 FORYD=ETO199STEP15*S2
540 F-1,YD TO F+1,YD
550 FORYU=ETO0STEP-15*S2
560 F-1,YU TO F+1,YU
580 IFPL=1THEN820
610 REM POLAR PLOT
690 RD=7/180:FORG=0TO360STEP1K:T=G*RD
710 X=FNR(T)*COS(T):Y=FNR(T)*SIN(T)
730 XX=X*(19*S1)+F:YY=-Y*(15*S2)+E
740 IFXX<0ORXX>319THEN780
750 IFYY<0ORYY>199THEN780
770 XX,YY
780 NEXT
790 END
820 REM CARTESIAN PLOT
830 FORG=0TO319STEP1K
840 X=(G-F)/(19*S1):Y=FNR(X)
850 YY=E-(Y*15*S2)
860 IFYY<0ORYY>199THEN960
900 G,YY
960 NEXT
970 END
1010 REM 3D-PLOT
1020 PRINT "VERTIKAL-MASSTAB"
1030 PRINT "40 BIS 40 20";IN
PUTN1
1040 PRINT "VERTIKALE LAGE"
1050 PRINT "50 BIS 150 90";IN
PUTN2

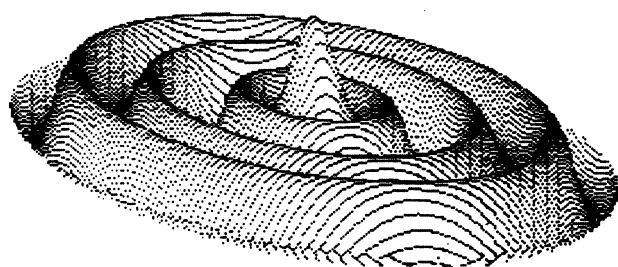
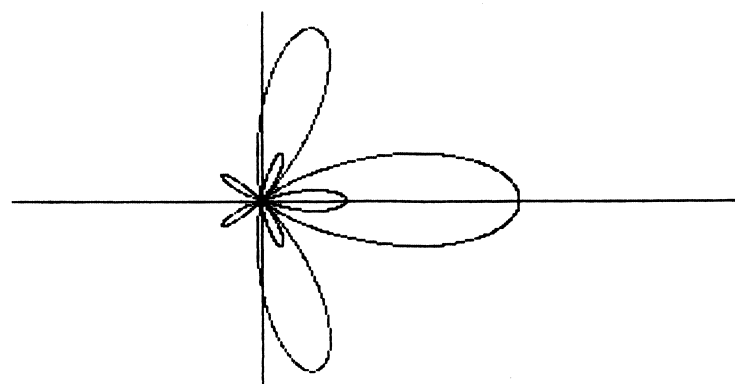
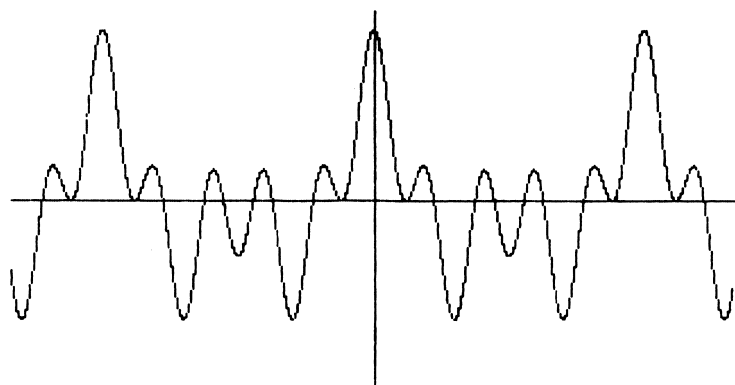
```

```

1260 REM KONSTANTEN A,B,C,D,E,F,G
1280 A=144:B=2.25:C=N1:D=.0327:E=160:F=N
2:G=199
1400 0,1:
1410 FORH=-ATOASTEPI
1420 AA=INT(.5+SQR(A*A-H*H))
1430 FORBB=-AATOA:CC=SQR(BB*BB+H*H)*D
1440 D1=FNR(CC):DD=D1*C:GOSUB1520:NEXT:N
EXT:END
1450 GOTO1450
1520 X=BB+H/B+E:Y=DD-H/B+F
1530 X1=INT(.85*X):Y1=INT(.9*(G-Y)):IFY<
00RY>199THENRETURN
1600 X1,Y1
1620 E X1,Y1+1TOX1,199:RETURN

```

Auf der nächsten Seite finden Sie die Ergebnisse dieses Programms.



2.3 FARBIGE BALKENGRAFIKEN

Das folgende Maschinenprogramm dient zum Zeichnen von waagerechten oder senkrechten farbigen Balkengrafiken. Damit lassen sich Kurven und Tabellen übersichtlich grafisch in Farbe auf dem Bildschirm darstellen. Da die Grafik mit normalen Bildschirmzeichen erstellt wird, läßt die Darstellung sich beliebig mit Texten mischen, z.B. zur Beschriftung der Grafiken. Die Balken sind jeweils 8 Bildpunkte gleich ein Zeichen breit und können horizontal 320 Punkte oder vertikal 200 Punkte auflösen.

Das Maschinenprogramm wurde so konzipiert, daß Länge bzw. Höhe und die Farbe angegeben werden können. Die Balkengrafik wird jeweils ab der aktuellen Cursorposition gezeichnet. Um die Darstellung kompletter Grafiken zur vereinfachen, wird der Cursor jeweils nach der Ausgabe eines vertikalen Balkens (der von unten nach oben gezeichnet wird) um eins nach rechts gesetzt, so daß sofort der nächste Balken (bei Bedarf in einer anderen Farbe) gezeichnet werden kann. Bei horizontalen Balkengrafiken rutscht der Cursor jeweils automatisch um eins nach unten.

Der Aufruf erfolgt durch einen erweiterten SYS-Befehl:

SYS H, L, C bzw.

SYS V, L, C

wobei H und V die Startadressen der Routinen für horizontale und vertikale Grafiken sind, L die Länge der Grafik bis 320 bzw. 200 und C der Farbcode ist (0 bis 15).

Im folgenden ist das Maschinenprogramm abgedruckt:

```

100:  c000                                .opt p1
                                           ; farbige balkengrafik
                                           ; hplot und vplot
130:  c000      getcor  =    $b7eb
140:  c000      scrout  =    $e716
150:  c000      lbyt   =    $14
160:  c000      hbyt   =    lbyt + 1
170:  c000      curcol  =    $d3
180:  c000      setcol  =    $ea24
190:  c000      setchar =    $ea1e
200:  c000      illquan =    $b248
205:  c000      chkcom  =    $aefd
210:  c000      code   =    $22
220:  c000      tmp    =    code + 1
230:  c000      xreg    =    tmp + 1
235:  c000      tmp1    =    xreg + 1
240:  c000      farb   =    $f3      ; zeiger auf farbram
250:  c000      curright =    $ab3b
260:  c000      adr     =    $fd
270:  c000      linelen =    $d5
280:  c000      charadr =    $d1
290:  c000      *=      $c000
300:  c000 20 fd ae hplot  jsr  chkcom  ; komma
310:  c003 20 eb b7      jsr  getcor
315:  c006 86 24      stx  xreg
320:  c008 a5 15      lda  hbyt
330:  c00a c9 02      cmp  #2
340:  c00c b0 42      bcs  ill
350:  c00e 0a      asl
350:  c00f 0a      asl
350:  c010 0a      asl
350:  c011 0a      asl
350:  c012 0a      asl

```

360:	c013 85 23	sta tmp
370:	c015 a5 14	lda lbyt
380:	c017 48	pha
390:	c018 4a	lsr
390:	c019 4a	lsr
390:	c01a 4a	lsr ; durch 8
400:	c01b 18	clc
410:	c01c 65 23	adc tmp
420:	c01e 65 d3	adc curcol ; cursorspalte
430:	c020 48	pha
440:	c021 a8	tay
450:	c022 c5 d3	cmp curcol
460:	c024 f0 13	beq t1
470:	c026 c9 27	cmp #39 ; < 40
480:	c028 90 02	bcc t2
490:	c02a a0 27	ldy #39
500:	c02c 20 24 ea t2	jsr setcol ; zeiger auf farbram
510:	c02f a9 a0	lda #" " + \$80 ; reverse blank
540:	c031 20 1e ea	jsr setchar ; zeichen und farbe setzen
550:	c034 88	dey
560:	c035 c4 d3	cpy curcol
570:	c037 10 f3	bpl t2
580:	c039 68 t1	pla
590:	c03a a8	tay
600:	c03b 68	pla
610:	c03c c0 28	cpy #40
620:	c03e b0 0b	bcs fertig
630:	c040 29 07	and #7
640:	c042 aa	tax
650:	c043 bd 53 c0	lda table,x
655:	c046 a6 24	ldx xreg
660:	c048 20 1e ea	jsr setchar
670:	c04b a9 11 fertig	lda #17 ; cursor down
680:	c04d 4c 16 e7	jmp scrout

```

690:  c050 4c 48 b2 ill      jmp  illquan
700:  c053 20 65 74 table     .byt  $20,$65,$74,$75,$61,$f6,$ea,$e7
1000: c05b 20 fd ae vplot     jsr  chkcom
1010: c05e 20 eb b7           jsr  getcor
1020: c061 a5 15             lda  hbyt
1030: c063 d0 eb            bne  ill
1040: c065 86 24            stx  xreg      ; farbe
1050: c067 a5 14            lda  lbyt
1060: c069 4a              lsr
1060: c06a 4a              lsr
1060: c06b 4a              lsr      ; durch 8
1070: c06c 85 23           sta  tmp
1080: c06e a5 14           lda  lbyt
1090: c070 29 07           and  #7
1100: c072 85 25           sta  tmp1
1110: c074 a5 d1           lda  charadr ; zeilenadresse
1120: c076 18              clc
1130: c077 65 d3           adc  curcol ; plus cursorspalte
1140: c079 85 fd           sta  adr
1150: c07b a5 d2           lda  charadr + 1
1160: c07d 69 00           adc  #0
1170: c07f 85 fe           sta  adr+1
1180: c081 a0 00           ldy  #0
1190: c083 a6 23           ldx  tmp
1200: c085 f0 20           beq  t3
1210: c087 20 c7 c0 t4     jsr  color ; farbadresse berechnen
1270: c08a a9 a0           lda  #" " + $80
1275: c08c 91 fd           sta  (adr),y
1280: c08e a5 24           lda  xreg ; farbe
1290: c090 91 f3           sta  (farb),y ; zeichen und farbe setzen
1300: c092 a5 fd           lda  adr
1310: c094 38              sec
1320: c095 e9 28           sbc  #40 ; nächste zeile
1330: c097 85 fd           sta  adr

```

```

1340: c099 b0 08          bcs t5
1350: c09b c6 fe          dec adr+1
1360: c09d a5 fe          lda adr+1
1370: c09f c9 04          cmp #4          ; oberste zeile erreicht ?
1380: c0a1 90 12          bcc t6
1390: c0a3 c6 23          t5      dec tmp
1400: c0a5 d0 e0          bne t4
1410: c0a7 20 c7 c0 t3     jsr color
1470: c0aa a6 25          ldx tmp1
1480: c0ac bd bf c0       lda tab2,x
1485: c0af 91 fd          sta (adr),y
1490: c0b1 a5 24          lda xreg          ; farbe
1500: c0b3 91 f3          sta (farb),y
1510: c0b5 a5 d3          t6      lda curcol
1520: c0b7 c5 d5          cmp linelen ; cursor in letzter spalte ?
1530: c0b9 f0 03          beq * + 5
1540: c0bb 4c 3b ab       jmp curright ; cursor right
1550: c0be 60              rts
1560: c0bf 20 64 6f tab2   .byt $20,$64,$6f,$79,$62,$f8,$f7,$e3
1570: c0c7 a5 fd          color   lda adr
1580: c0c9 85 f3          sta farb
1590: c0cb a5 fe          lda adr+1
1600: c0cd 4c 2a ea       jmp setcol + 6

```

Hier ein Ladeprogramm in BASIC für diejenigen, die keinen Assembler oder Monitor zur Verfügung haben.

```

100 for i = 49152 to 49359
110 read x : poke i,x : s=s+x : next
120 data 32,253,174, 32,235,183,134, 36,165, 21,201, 2
130 data 176, 66, 10, 10, 10, 10, 10,133, 35,165, 20, 72
140 data 74, 74, 74, 24,101, 35,101,211, 72,168,197,211
150 data 240, 19,201, 39,144, 2,160, 39, 32, 36,234,169

```



```

160 data 160, 32, 30,234,136,196,211, 16,243,104,168,104
170 data 192, 40,176, 11, 41, 7,170,189, 83,192,166, 36
180 data 32, 30,234,169, 17, 76, 22,231, 76, 72,178, 32
190 data 101,116,117, 97,246,234,231, 32,253,174, 32,235
200 data 183,165, 21,208,235,134, 36,165, 20, 74, 74, 74
210 data 133, 35,165, 20, 41, 7,133, 37,165,209, 24,101
220 data 211,133,253,165,210,105, 0,133,254,160, 0,166
230 data 35,240, 32, 32,199,192,169,160,145,253,165, 36
240 data 145,243,165,253, 56,233, 40,133,253,176, 8,198
250 data 254,165,254,201, 4,144, 18,198, 35,208,224, 32
260 data 199,192,166, 37,189,191,192,145,253,165, 36,145
270 data 243,165,211,197,213,240, 3, 76, 59,171, 96, 32
280 data 100,111,121, 98,248,247,227,165,253,133,243,165
290 data 254, 76, 42,234
300 if s <> 26696 then print "fehler in datas !!" : end
310 print "ok !"

```

Sehen wir uns nun eine mögliche Anwendung der Grafikroutinen an. Als erstes wollen wir eine Umsatzstatistik grafisch darstellen.

```

100 rem in den data statement sind die monatsumsatzzahlen
110 rem eines jahres gespeichert
120 dim u(12)
130 rem einlesen der daten
140 for i = 1 to 12 : read u(i) : next
150 rem bestimmung des maximalwertes
160 max = 0
170 for i = 1 to 12
180 if u(i) > max then max = u(i)
190 next
200 v = 12*4096+5*16+11 : rem adresse der maschinenroutine
220 print chr$(147) : rem bildschirm löschen
230 for i = 1 to 21 : print chr$(17); : next : rem cursor

```

```

240 rem grafik zeichnen
250 for i = 1 to 12
260 print spc(2); : sys v, u(i)/max * 180 , i
270 next
280 print : print
290 rem monatsnummern schreiben
300 for i=1 to 12
310 print right$(" "+str$(i),3);
320 next
330 get a$ : if a$ ="" then 330
400 rem umsatzdaten
410 data 12000, 13500, 11000, 8000, 14000, 9000
420 data 13800, 14000, 12750, 14000, 13800, 17200

```

Nach dem Beispiel für vertikale Balkengrafiken sehen wir uns eine Funktion mit der horizontalen Balkengrafik an.

In Zeile 100 wird der Farbcode eingegeben. Dann wird der Bildschirm gelöscht, der Hintergrund auf schwarz geschaltet und der Cursor in die zweite Spalte gebracht. In Zeile 120 und 130 wird die Funktion von -2.2 bis 2.2 berechnet, durch Multiplikation mit 300 in eine darstellbare Größenordnung gebracht und mit dem SYS-Befehl geplottet.

```

100 input "farbe ";c:if c<1 or c>15 then 100
110 h=12*4096 : print chr$(147)tab(2); : poke53281,0
120 for i=-2.2 to 2.2 step .2
130 sys h,exp(-i*i)*300,c : next

```

2.4 DEFINITION EINES EIGENEN ZEICHENSATZES

Eine der Besonderheiten des Commodore 64 ist die Möglichkeit, den Zeichengenerator in den RAM-Speicher zu legen. Dies bietet Ihnen die Möglichkeit, eigene Zeichen zu definieren.

Wie wird nun ein Zeichen definiert?

Jedes Zeichen wird durch seine sogenannte Zeichenmatrix bestimmt, die aus 8 * 8 Punkten besteht. Jeder Matrixpunkt wird durch ein Bit im Zeichengenerator bestimmt, insgesamt 64 Bits oder 8 Byte. Ist ein Bit Null, so ist der entsprechende Punkt der Matrix nicht gesetzt. Ein gesetztes Bit bedeutet einen gesetzten Punkt in der Zeichenmatrix. Das folgende kleine Programm zeigt die Matrix eines Zeichens auf dem Bildschirm an. Das Programm benutzt die modifizierte PEEK-Funktion aus Kapitel 9.5 - laden oder geben Sie deshalb zuerst dieses Programm ein bevor Sie das Programm eingeben.

```
100 PRINT "Zeichen"
110 INPUT "BITTE ZEICHEN EINGEBEN ":A$
120 PRINT "A$ :C=PEEK(1024)
130 PRINT:PRINT:PRINT:PRINT
140 CG = 13*4096 : REM START DES ZEICHEN
    GENERATORS
150 REM BESTIMMUNG VON KLEIN/GROSS ODER
    GRAFIKMODUS
160 B = (PEEK(53248+24) AND 2) * 1024
170 FOR I = 0 TO 7
180 Z =USR (CG+B+8*C+I) : REM: MATRIX ZE
    ILENWEISE HOLEN
190 FOR J=7 TO 0 STEP -1
200 A = Z AND 2↑J
210 IF A THEN PRINT "*": GOTO 230
220 PRINT " ";
230 NEXT:PRINT:NEXT
```

Das Programm fragt nach dem Zeichen, dessen Matrix Sie ansehen wollen. Der Bildschirmcode des Zeichens wird in Zeile 120 in die Variable B geholt. Zeile 140 prüft auf Groß/Klein- oder Groß/Grafik-Modus. In Zeile 160 wird die Startposition der Zeichenmatrix im Charactergenerator bestimmt. In Zeile 180 wird bestimmt, ob ein Matrixpunkt gesetzt ist oder nicht. In Abhängigkeit davon wird entweder ein Stern oder ein Punkt gezeichnet. Geben Sie ein "T" ein, erhalten Sie folgende Ausgabe:

```
.*****.
...**...
...**...
...**...
...**...
...**...
...**...
...**...
.....
```

Nachdem Sie sich nun die Matrix einzelner Zeichen angesehen haben, können wir daran gehen, eigene Zeichen zu definieren. Dazu müssen wir den Charactergenerator vom ROM ins RAM übertragen und dann dem Betriebssystem mitteilen, wo der neue Charactergenerator liegt. Dabei wird gleichzeitig der Bildschirmspeicher nach Adresse \$C400 (dezimal von 50176 bis 51175) verschoben. Das kann von BASIC aus mit einer POKE-Schleife geschehen. Wir benutzen dazu wieder die USR-Funktion aus Kapitel 9.5.

```
100 FOR I=13*4096 TO 14*4096-1
110 POKE I+4096,USR(I) : NEXT
120 POKE 53272,24 : POKE 56576,148 : POKE 648,196
```

Nachdem Sie dieses Programm ausgeführt haben, können wir mit folgendem Program unsere eigenen Zeichen definieren. Das Programm fragt nach dem Zeichen, das Sie ändern möchten. Dann können Sie die Zeichenmatrix eingeben; ein Stern bedeutet wieder einen gesetzten Punkt und ein Punkt heißt wieder kein Punkt gesetzt. Wollen Sie das Programm beenden, geben Sie bitte als Zeichen das Wort 'ENDE' ein.

```

100 REM ZEICHENDEFINITION
110 CG=14*4096: REM BASIS DES ZEICHENGEN
ERATORS
200 INPUT "Zeichenmatrix ZEICHEN ";A$:IFA$="END
E"THENEND
210 PRINT"A$
220 C=PEEK(12*4096+1024)
230 PRINT"Matrix:01234567"
300 FOR I=0 TO 7
310 PRINTI:INPUTA$(I):IFLEN(A$(I))<>8TH
ENPRINT" ":GOTO310
320 NEXT
400 B=(PEEK(53248+24)AND2)*1024 : REM GR
OSS/GRAFIK-MODUS
405 AD=CG+B+C*8
410 FOR I=0 TO 7 : Z=0
420 FOR J=0 TO 7
430 Z=Z-(MID$(A$(I),8-J,1)="*")*2↑J
440 NEXT
450 POKE AD+I,Z : REM ZEICHEN
460 POKE AD+1024+I,255-Z : REM RVS-ZEICH
EN
470 NEXT
480 GOTO 200

```

2.5 MODIFIKATION DES ZEICHENSATZES MITTELS JOYSTICK

Für bestimmte Zwecke ist es oft wünschenswert, besondere Zeichen und Symbole zur Verfügung zu haben, die auf Tastendruck sofort erscheinen. Darunter fallen z.B. griechische Buchstaben, die in mathematischen Formeln häufig benutzt werden.

Denkbar sind auch kleine Spielfiguren, die zwar auch mit Sprites erzeugt werden können, was allerdings etwas aufwendiger sowohl beim Erstellen als auch in der Handhabung ist.

Wenn Ihnen die geeigneten Verfahren geläufig sind, können Sie sich Ihre Figuren zunächst als Rasterdarstellung auf ein Blatt Papier malen und dann die ermittelten Werte in das Duplikat des Zeichengenerators poken, aber das ist bestimmt nicht jedermanns Sache.

Wir haben daher ein kleines Programm geschrieben, mit Hilfe dessen Sie das gewünschte Zeichen bequem am Bildschirm entstehen lassen und seine Entwicklung verfolgen können.

Voraussetzung hierzu ist lediglich ein Joystick am Controlport 1.

Die Wirkungsweise des Programmes ist die, daß zunächst zwei Duplikate des Zeichengenerators ins Ram kopiert werden.

Aus dem ersten Duplikat wird ein Zeichen entnommen und als Sprite dargestellt, und zwar einmal in Originalgröße und einmal in doppelter Größe, der besseren Lesbarkeit wegen.

In den Sprites erscheint ein blinkender Punkt, der sog. Mikrocursor, den Sie nun mit Hilfe des Joystick bewegen können.

Die gewünschte Aktion, nämlich Linie ziehen, Linie löschen oder nur positionieren, können Sie durch Druck auf den Knopf am Joystick auswählen.

Ist das Zeichen zu Ihrer Zufriedenheit ausgefallen, so

drücken Sie die Taste F1.

Das neue Zeichen wird nun an die angezeigte Position des Alternativzeichensatzes (2. Duplikat) übernommen.

Soll das dargestellte Zeichen unverändert bleiben, drücken Sie die Taste G.

Nun können Sie das nächstfolgende Zeichen bearbeiten.

Nachdem Sie alle 512 Zeichen bearbeitet haben, beendet sich das Programm.

Wieso eigentlich 512 Zeichen?

Nun, für eine Betriebsart, nämlich Groß/Grafik oder Groß/Kleinschrift, gibt es 128 darstellbare Zeichen. Dieselben Zeichen noch einmal in reverser Darstellung macht schon 256. Das ergibt für beide Betriebsarten zusammen 512.

Die den Zeichen zugeordneten Positionen im Zeichengenerator können Sie den Seiten 132ff im Bedienungshandbuch des C64 entnehmen.

Bevor wir das Programm im Einzelnen erörtern, sollten Sie die Bedeutung der verwendeten Variablen und Speicheradressen kennen.

Zunächst die Variablen:

C Basisadresse des 1.Duplikats des als nächstes anzuzeigenden Zeichens

CD Basisadresse des zu übertragenden Zeichens im 2.Duplikat

CP Zeichenpositionszeähler im Bereich 0-511

JB Zustand des Knopfes am Joystick

JR Knüppelstellung des Joystick

JS Adresse des Controlport 1

MA Zähler für den Betriebszustand

PO Mikrocursorposition innerhalb des adressierten Bytes

PP Adresse innerhalb der Spritedaten, wo sich gerade der Mikrocursor befindet

PV Augenblickswert des Bytes der Spritedaten, welches durch

den Joystick gerade adressiert ist
 SB Basisadresse der Spritedaten
 V Basisadresse des Videocontrollers
 X x-Position der Sprites auf dem Bildschirm
 XJ x-Position des durch den Joystick gesteuerten Mikrocursor
 Y y-Position der Sprites auf dem Bildschirm
 YJ y-Position des durch den Joystick gesteuerten Mikrocursor

Nun die Adressen:

56 Höherwertiges Byte des Pointer auf die Speicherobergrenze
 648 Pointer auf den Beginn des Videoram
 832 Startadresse des Kassettenpuffers.
 Da dieser innerhalb des Programmes nicht anderweitig benötigt wird, ist hier das Maschinenprogramm hinterlegt.
 50169 Pointer für Sprite 1
 50170 Pointer für Sprite 2
 53272 Pointer für Videoram und Zeichengenerator innerhalb des Videocontroller
 56576 Hier stehen die beiden Bits zur Festlegung des 16k-Bereiches für die mit dem Bildschirm zusammenhängenden Speicher.

Es folgt die schrittweise Einführung in das Programm:

10 Die Speicherobergrenze wird herabgesetzt, da hier das erste Duplikat des Zeichensatzes hinterlegt werden soll.
 30-70 Die Sprites 1+2 werden eingeschaltet und deren Farbe gesetzt.

Nun werden die Spritepointer geladen und dann Sprite 2 auf doppelt große Darstellung geschaltet.

Die Spritedaten werden zunächst gelöscht und dann die Sprites ungefähr in die Bildschirmmitte positioniert.

1000-1010 Hier steht das Maschinenprogramm.

Für den Interessierten nachstehend die Assemblerschreibweise:

sei	Interrupt verhindern
lda #\$33	Zeichengenerator zugänglich machen
sta 1	
lda #0	
sta \$5f	alter Blockanfang lo
sta \$5a	altes Blockende lo
sta \$58	neues Blockende lo
lda #\$d0	
sta \$60	alter Blockanfang hi
lda #\$f0	
sta \$59	neues Blockende hi
lda #\$e0	
sta \$5b	altes Blockende hi
jsr \$a3bf	Blockverschieberoutine
lda #\$37	
sta 1	
cli	Interrupt erlauben
rts	zurück nach Basic

1020-1040 Das Maschinenprogramm wird in den Kassettenuffpuffer gebracht und ausgeführt, das heißt von dem Zeichengenerator werden zwei Duplikate erzeugt.

1060 Dem Betriebssystem werden die veränderten

Verhältnisse mitgeteilt.

Verändert wurde die Lage von Zeichengenerator und (wegen der Hardware zwangsläufig) Videoram. Die Zeichen, die Sie jetzt auf dem Bildschirm sehen, stammen bereits aus dem 2. Duplikat.

2000-2360 Die Zeichen werden nun nacheinander in die Spritedaten überführt und können verändert werden.

2380 Nach Behandlung aller Zeichen wird die Möglichkeit zum Abspeichern des neuen Zeichensatzes angeboten.

4000-4070 Der Joystick wird abgefragt und abhängig von der Stellung des Knüppels und des Knopfes verzweigt.

5000-12040 Abhängig von der Stellung des Joystick wird die Cursorposition aktualisiert.

13000-13200 Die Position des Joystick wird auf die Punkteposition des Mikrocursor umgerechnet und dieser Punkt abwechselnd an- und ausgeschaltet.

20000-20060 Hier wird das für gut befundene Zeichen in das 2. Duplikat übertragen.

30000-30200 Hier kann der neu erzeugte Zeichensatz auf Diskette (8) oder Kassette (1) abgespeichert werden.

Als Besonderheit dieses Programmes ist noch anzumerken, daß der modifizierte Zeichengenerator keinen Basicspeicher belegt. Er liegt nämlich im Ram unter dem Betriebssystem (das ist wörtlich zu nehmen, denn bekanntlich überlappen sich beim C64 Ram und Rom).

Der Videoram hat nun die Startadresse 49152, was Sie bitte berücksichtigen wollen, wenn Sie hineinpoken. Durch die Hardware des C64 ist es leider gegeben, daß bei Verlegen des Zeichengenerators auch eine Verschiebung des Videorams

stattfindet. Diese Problematik finden Sie in unserem Buch
'64 intern' näher erläutert.

Abschließend das Programmlisting:

```
10 poke56,144:clr:v=53248
30 pokev+21,6:pokev+40,1:pokev+41,1
40 poke50169,16:poke50170,16:pokev+23,4:pokev+29,4
50 fori=0to62:poke50176+i,0:next
60 x=150:y=100:pokev+4,x:pokev+2,x-40:pokev+16,0
70 pokev+5,y:pokev+3,y+19
1000 data120,169,51,133,1,169,0,133,95,133,90,133,88,169
    ,208,133,96,169,240,133
1010 data89,169,224,133,91,32,191,163,169,55,133,1,88,96
1020 fori=832to832+33:reada:pokei,a:next
1040 sys832:poke850,160:sys832
1060 poke53272,8:poke56576,peek(56576)and252:poke648,192
1070 printchr$(147)
2000 c=9*4096
2020 forcp=0to511:printchr$(19)cp:sb=50176
2040 fori=0to7:pokesb+3*i,peek(c+i):nexti
2360 c=c+8:gosub4000:nextcp
2380 pokev+21,0:goto 30000
4000 xj=0:yj=0:js=56321:sb=50176:pp=832
4020 jr=(255-peek(js))and15:jb=(255-peek(js))and16
4030 ifjbthenma=ma+1:ifma>2thenma=0
4040 onjrgoto5000,6000,4020,7000,8000,9000,4020,
    10000,11000,12000
4045 ifpeek(203)<>4then4066
4050 printchr$(19)chr$(17)chr$(17)chr$(17)chr$(18)
    "save":gosub20000
4055 printchr$(19)chr$(17)chr$(17)chr$(17)chr$(17)"      "
    :return
4066 ifma=1thenprintchr$(19)chr$(17)chr$(17)chr$(18)" set"
```

```

4067 ifma=2thenprintchr$(19)chr$(17)chr$(17)chr$(18)" clr"
4068 ifpeek(203)=26thenreturn
4069 ifma=0thenprintchr$(19)chr$(17)chr$(17)"      "
4070 gosub13000:goto4020
5000 rem oben
5020 yj=yj-1:ifyj<0thenyj=0
5040 gosub13000:goto4020
6000 rem unten
6020 yj=yj+1:ifyj>7thenyj=7
6040 gosub13000:goto4020
7000 rem links
7020 xj=xj-1:ifxj<0thenxj=0
7040 gosub13000:goto4020
8000 rem links oben
8020 xj=xj-1:ifxj<0thenxj=0
8040 goto5000
9000 rem links unten
9020 xj=xj-1:ifxj<0thenxj=0
9040 goto6000
10000 rem rechts
10020 xj=xj+1:ifxj>7thenxj=7
10040 gosub13000:goto4020
11000 rem rechts oben
11020 xj=xj+1:ifxj>7thenxj=7
11040 goto5000
12000 rem rechts unten
12020 xj=xj+1:ifxj>7thenxj=7
12040 goto6000
13000 rem
13020 pp=sb+yj*3+int(xj/8):pv=peek(pp)
13040 po=xj-int(xj/8)*8
13060 ifpvand2^(7-po)thenpokepp,(pvand(255-2^(7-po)))
      :goto13100
13080 pokepp,(pvor(2^(7-po)))

```

```

13100 ifma=1thenpv=(pvor(2↑(7-po)))
13120 ifma=2thenpv=(pvand(255-2↑(7-po)))
13200 fori=0to50:next:pokepp,pv:return
20000 rem neues zeichen uebertragen
20010 cd=c+20472
20020 fori=0to7
20040 pokecd+i,peek(sb+3*i):pokecd+i-20480,peek(sb+3*i)
20060 nexti:return
30000 input"abspeichern ";r$
30020 if r$<>"j"then 30200
30040 input"name ";n$
30060 input"geraeteadresse ";ad
30080 poke186,ad:poke781,0:poke782,160
30120 poke251,0:poke252,144:poke780,251
30140 poke183,len(n$):poke187,680and255:poke188,680/256
30160 fori=1tolen(n$)
30180 poke679+i,asc(mid$(n$,i,1)):next:sys65496
30200 poke56,160:clr

```

Das nachfolgende kleine Programm ermöglicht es, den Zeichensatz wieder einzulesen und zu aktivieren.

```

100 rem Zeichensatz laden
110 input"name ";n$
120 input"geraeteadresse ";ad
130 poke186,ad:poke781,0:poke782,224
140 poke780,0:poke185,0:poke183,len(n$)
150 poke187,680and255:poke188,680/256
160 fori=1tolen(n$):poke679+i,asc(mid$(n$,i,1)):next
170 sys65493:rem load
180 poke53272,8:poke56576,peek(56576)and252:poke648,192
190 printchr$(147)

```

2.6 DER 'GETEILTE' BILDSCHIRM

Eine Eigenschaft des Videocontrollers im Commodore 64, die sehr schöne Effekte ermöglicht, ist noch wenig beschrieben und benutzt worden - der Rasterinterrupt.

Um Ihnen zu erklären, was es damit auf sich hat, müssen wir etwas weiter ausholen und uns klar machen, wie der Videocontroller Ihres Rechners ein Fernsehbild erzeugt.

Das Fernsehbild wird dazu aus einzelnen Zeilen aufgebaut, die Sie deutlich erkennen können, wenn Sie das Fernsehbild einmal näher betrachten. Sie können dann auch sicher erkennen, daß z.B. ein Buchstabe aus acht Zeilen aufgebaut ist. Der Videocontroller hat nun ein Register, das immer die Nummer der Fernsehzeile enthält, die gerade dargestellt wird. Dieses Register hat die Nummer 18, ist also unter der Adresse $53248+18 = 53266$ zu erreichen. Fragen Sie den Inhalt dieses Register ab,

```
PRINT PEEK(53266)
```

so erhalten Sie zufällige Werte, die die augenblicklich dargestellte Rasterzeile angeben. Da in einer Sekunde jedoch 25 Fernsehbilder dargestellt werden, können wir dies in BASIC nicht schnell genug auswerten. Wir müssen also zur Programmierung in Maschinensprache übergehen. Dabei kommt uns noch eine Eigenschaft des Videocontrollers zunutze: Der Videocontroller ist in der Lage, ein laufendes Programm dann zu unterbrechen, wenn gerade eine bestimmte Rasterzeile dargestellt wird. Die Nummer dieser Rasterzeile können Sie durch Schreiben in das gleiche Register des Videocontrollers bestimmen. Eine solche automatische Programmunterbrechung nennt man 'Interrupt'. Weiterhin muß der Interrupt durch den

Videocontroller zugelassen werden. Dies geschieht durch beschreiben des Interruptmaskenregisters. Eine solche Programmierung ist jedoch nur in Maschinensprache möglich. Wir haben deshalb ein kleines Programm für Sie geschrieben, das diese Möglichkeiten für Sie ausnutzt.

```

10: 033C          .OPT P1
20: 033C          IRQALT = $EA31
25: 033C          IRQVECT = $314
35: 033C          RASTER = $D012 ; Rasterzeile
40: 033C          IRQREG = $D019 ; Flag für Videointerrupt
50: 033C          MASKE = $D01A ; Interruptmaske des Videocontrollers
55: 033C          RAHMEN = $D020 ; Rahmenfarbe
60: 033C          FARBE = $D021 ; Hintergrundfarbe
100: 033C          ICR = $DCOD ; Flag für Timerinterrupt
110: 033C          RETIRQ = $FEBC ; Rückkehr vom Interrupt
120: 00FB          *= $FB
130: 00FC          ZEILE1 *= ++1
140: 00FD          ZEILE2 *= ++1
150: 00FE          FARBE1 *= ++1
160: 00FF          FARBE2 *= ++1

;

190: 033C          *= 828 ; Kassettenpuffer
200: 033C 78      SETUP SEI
210: 033D A9 5B    LDA #< IRQNEU
220: 033F 8D 14 03 STA IRQVECT
230: 0342 A9 03    LDA #> IRQNEU
240: 0344 8D 15 03 STA IRQVECT+1
260: 0347 A5 FB    LDA ZEILE1
270: 0349 8D 12 D0 STA RASTER ; Rasterzeile für Interrupt
275: 034C AD 11 D0 LDA RASTER-1
275: 034F 29 7F    AND #$7F ; Highbyte löschen
275: 0351 8D 11 D0 STA RASTER-1
280: 0354 A9 81    LDA #$81 ; IRQ durch Raster erlauben

```

```

300: 0356 8D 1A D0      STA MASKE
310: 0359 58            CLI
320: 035A 60            RTS

;
400: 035B AD 19 D0 IRQNEU LDA IRQREG
400: 035E 8D 19 D0      STA IRQREG ; Interruptflag löschen
410: 0361 29 01          AND #1
420: 0363 D0 07          BNE SCREEN ; Interrupt durch Rasterzeile ?
430: 0365 AD 0D DC      LDA ICR ; Interruptflag löschen
430: 0368 58            CLI ; Rasterinterrupt zulassen
435: 0369 4C 31 EA      JMP IRQALT

;
450: 036C AD 12 D0 SCREEN LDA RASTER ; Rasterzeile lesen
460: 036F C5 FC          CMP ZEILE2
470: 0371 B0 0D          BCS ZWEITER ; größer oder gleich 2. Wert ?
480: 0373 A5 FD          LDA FARBE1
490: 0375 8D 20 D0      STA RAHMEN ; Farbe 1 setzen
500: 0378 8D 21 D0      STA FARBE
510: 037B A5 FC          LDA ZEILE2 ; nächsten Interrupt bei 2. Wert
520: 037D 4C 8A 03      JMP EXIT
560: 0380 A5 FE ZWEITER LDA FARBE2
570: 0382 8D 20 D0      STA RAHMEN ; Farbe 2 setzen
580: 0385 8D 21 D0      STA FARBE
590: 0388 A5 FB          LDA ZEILE1 ; nächsten Interrupt
600: 038A 8D 12 D0 EXIT STA RASTER ; bei Zeile 1
610: 038D 4C BC FE      JMP RETIRQ

```

Hier wieder ein Ladeprogramm in BASIC. Geben Sie dieses Programm zuerst ein, speichern Sie es ab und starten Sie es mit 'RUN'.


```

100 FOR I = 828 TO 911
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 120,169, 91,141, 20, 3,169, 3,141, 21, 3,165
130 DATA 251,141, 18,208,173, 17,208, 41,127,141, 17,208
140 DATA 169,129,141, 26,208, 88, 96,173, 25,208,141, 25
150 DATA 208, 41, 1,208, 7,173, 13,220, 88, 76, 49,234
160 DATA 173, 18,208,197,251,176, 13,165,253,141, 32,208
170 DATA 141, 33,208,165,252, 76,138, 3,165,254,141, 32
180 DATA 208,141, 33,208,165,251,141, 18,208, 76,188,254
190 IF S <> 10677 THEN PRINT "FEHLER IN DATAS !!" : END
200 PRINT "OK"

```

Als Beispielprogramm hierzu haben wir ein Programm für Sie entwickelt, daß es Ihnen ermöglicht, einen Bildschirmbereich mit einer anderen Hintergrundfarbe zu versehen. Sie können dadurch eine oder mehrere Bildschirmzeilen hervorheben. Um das Programm möglichst universell zu halten, können Sie sowohl die Hintergrundfarben des hervorzuhebenden Bereichs als auch die des übrigen Bereichs durch POKE-Befehle bestimmen. Ebenso können Sie die Rasterzeile, bei der die Umschaltung auf die zweite Hintergrundfarbe geschehen soll bestimmen. Dies gilt natürlich auch für die Nummer der Rasterzeile, bei der wieder auf die erste Farbe zurückgeschaltet wird. Dies mag zwar etwas kompliziert klingen; probieren Sie jedoch das nachfolgende Programm einmal aus und Sie werden den Effekt sehen.

Mit diesem Programm können Sie einen farbigen Balken von der Breite einer Bildschirmzeile (8 Rasterzeilen) durch Drücken der Tasten 'Cursor hoch' und 'Cursor runter' um jeweils eine Zeile nach oben oder unten bewegen. Beachten Sie daß dies nicht auf einmal geschieht, sondern um 8 Rasterzeilen nacheinander. Die Funktionstasten können Sie benutzen um die Farbe des Balkens und des übrigen Hintergrunds zu ändern.

```

100 Z1=251:Z2=Z1+1:F1=Z2+1:F2=F1+1
110 Z=50:SYS828:REM INTERRUPT INITIALISIEREN
120 POKEZ1,Z:POKEZ2,Z+8:POKEF1,6:POKEF2,8
150 GETA$:IFA$=""THEN150
160 IFA$=CHR$(17)THENGOSUB200
170 IFA$=CHR$(145)THENGOSUB300
180 IFA$=CHR$(133)THENGOSUB400
190 IFA$=CHR$(134)THENGOSUB500
195 GOTO150
200 IFZ<240THENFORI=OTO7:Z=Z+1:POKEZ1,Z:POKEZ2,Z+8:NEXT
210 RETURN
300 IFZ<50THENFORI=OTO7:Z=Z-1:POKEZ1,Z:POKEZ2,Z+8:NEXT
310 RETURN
400 POKEF1,PEEK(F1)+1AND15:RETURN
500 POKEF2,PEEK(F2)+1AND15:RETURN

```

Sie können das Programm für Ihre Zwecke abändern, indem Sie in die Speicherzellen 251 bis 254 der Reihe nach die Nummer der Rasterzeile poken, bei der die Umschaltung auf die zweite Farbe stattfinden soll, dann die Rasterzeile, bei der wieder auf die ursprüngliche Farbe zurückgeschaltet wird. Die nächsten beiden Adressen 253 und 254 enthalten den Farbcode der ersten und zweiten Farbe. Für die Rasterzeilen gilt folgender Bereich:

Die Rasterzeile 50 entspricht der oberen Bildbegrenzung (dort wo der Rahmen beginnt), während der Beginn des unteren Rahmens der Rasterzeile 250 entspricht. Eine Bildschirmzeile ist in 8 Rasterzeilen aufgeteilt. Dadurch können Sie auch die Grenze zwischen zwei verschiedenen Farben mitten in eine Bildschirmzeile legen.

Die Umschaltung der Hintergrundfarbe ist jedoch nicht der einzige Effekt, den Sie mit dem Rasterinterrupt erreichen

können. Alle Parameter des Videocontrollers können Sie interruptgesteuert verändern. Hier ein paar Anregungen:

Blenden Sie in eine hochauflösende Grafik eine Textzeile ein oder Schalten Sie zwischen zwei Grafikseiten um. Sie sehen dann z.B. in der oberen Hälfte des Bildschirms ein Grafikbild und in der unteren Hälfte ein zweites. Durch stetiges Ändern der Rasterzeile, bei der umgeschaltet wird, können Sie so z.B. zwei Grafikbilder von oben nach unten überblenden, ähnlich wie in unserem Beispiel der farbige Balken. Ihrer Phantasie sind dabei keine Grenzen gesetzt; vielleicht probieren Sie auch einmal verschiedene Zeichensätze in verschiedenen Teilen des Bildschirms!

Mit dieser Technik können Sie auch Effekte erreichen, die der Commodore 64 von Hause aus eigentlich gar nicht kann, z.B. ist es mittels des Rasterinterrupts möglich, mehr als 8 Sprites gleichzeitig darzustellen. Dazu können Sie in der oberen Hälfte des Bildschirms acht Sprites darstellen. Beim Erreichen einer bestimmten Rasterzeile setzen Sie einfach die Spritepointer und -Koordinaten um und Sie können in der unteren Hälfte weitere 8 Sprites darstellen. Natürlich können Sie den Bildschirm dazu auch in mehr als zwei Teile aufteilen.

2.7 SOFT-SCROLL - EINE SACHE FÜR PERFEKTIONISTEN

Was Scrolling bedeutet, werden Sie sicher wissen: Der Bildschirm wird um eine Zeile nach oben gerollt. Dabei verschwindet die oberste Zeile und am unteren Bildschirmrand erscheint eine neue Zeile.

Unter Softscroll verstehen wir nun die Möglichkeit, eine neue Zeile langsam im Bild erscheinen zu lassen, während die alte Zeile langsam verschwindet. Diese Möglichkeit gibt uns der Videocontroller in die Hand. Dazu benutzen wir das Register 17. Die untersten drei Bit erlauben das Verschieben um insgesamt 8 Rasterzeilen, was genau einer Bildschirmzeile entspricht. Um nun eine neue Zeile auf dem Bildschirm erscheinen zu lassen, können wir den Videocontroller anweisen, lediglich 24 Zeilen darzustellen. Dies ist dann der Fall, wenn Bit 3 des Registers 17 gelöscht ist.

Wir gehen jetzt also folgendermaßen vor: Erst schalten wir den Bildschirm auf 24 Zeilen um und positionieren den restlichen Bildschirminhalt so, daß die obersten 24 Zeilen dargestellt werden. Nun können wir in die unsichtbare 25. Zeile etwas schreiben und den sichtbaren Anteil des Bildschirms um 8 Rasterzeilen gleich eine Zeile nach oben schieben, wobei gleichzeitig die oberste Zeile verschwindet.

Außer dem Scrollen nach oben (oder unten) unterstützt der Videocontroller noch das horizontale Softscrollen nach rechts oder links. Dazu dient das Register 22. Die untersten drei Bit sind wieder für das spaltenweise Verschieben zuständig, während Bit 3 die Darstellung von 38 Spalten ermöglicht.

Das Beispielprogramm scrollt den Bildschirm nach oben.

```

100 VIDEO = 53248
110 ZEILE = VIDEO + 17
115 X$=CHR$(19): FOR I=1 TO 24: X$=X$+CHR$(17): NEXT
120 POKE ZEILE ,PEEK(ZEILE) AND NOT 8
130 POKE ZEILE ,PEEK(ZEILE) AND 248 OR 7
140 N=N+1: A$="ZEILE"+STR$(N): GOSUB200: GOTO140
200 PRINT : PRINT X$ A$;
210 FOR I=7 TO 0 STEP-1
220 POKE ZEILE, PEEK(ZEILE) AND 248 OR I
230 FOR J=1 TO 100 : NEXT
240 NEXT: RETURN

```

115	Ein String aus einmal 'Cursor Home' und 24 mal 'Cursor Down' wird zur Cursorpositionierung in die 25. Zeile wird definiert.
120	Bit 3 im Register 17 des Videocontrollers wird gelöscht. Damit wird auf 24 Zeilendarstellung umgeschaltet.
130	Bit 0 bis 2 werden gesetzt. Damit werden die obersten 24 Zeilen des Bildschirms angezeigt, während die 25. Zeile am unteren Bildschirmrand unsichtbar bleibt.
140	Der Zähler N wird erhöht, in A\$ wird der auszugebende Text für das Unterprogramm 200 übergeben, daß anschließend aufgerufen wird.
200	Durch den PRINT-Befehl wird der Bildschirm um eine Zeile nach oben geschoben. Dann wird der auszugebende Text in die letzte Bildschirmzeile geschrieben.
210 -240	Diese Schleife schiebt den Bildschirm um acht Rasterzeilen nach oben. Die Verzögerungsschleife bestimmt das Tempo, mit dem nach oben gescrollt wird.

2.8 DIE TASTATURBELEGUNG UND IHRE ÄNDERUNG

Die Tastatur des Commodore 64 ist als eine Matrix mit 8 Zeilen und 8 Spalten angeordnet. Dabei sind die Leitungen der 8 Zeilen mit Port A (Adresse \$DC00 = 56320) von CIA 1 verbunden und die 8 Spalten liegen an Port B (Adresse \$DC01 = 56321) von CIA 1. Bei der Tastaturabfrage (Adresse \$FF9F = 65485) wird die Tastatur zeilenweise abgefragt, indem nacheinander auf jede Zeile über den Port A ein Signal angelegt wird. Ist nun eine Taste gedrückt, so kann man über Port B die Spalte der gerade gedrückten Taste abfragen. Aus Zeilen- und Spaltennummer wird nun die Tastennummer zwischen 0 und 63 berechnet. 64 bedeutet hier keine Taste gedrückt. Die Zuordnung ist in der obigen Tabelle dargestellt. Dieser Wert wird bei jeder Abfrage in Adresse \$CB gleich 203 abgelegt; die Nummer der letzten gedrückten Taste wird in \$C5 = 197 hinterlegt. Der Status der Sondertasten wird bei der Tastaturabfrage in Adresse \$028D = 653 gespeichert. Bit 0 bedeutet SHIFT, Bit 1 ist für COMMODORE reserviert und Bit 2 steht für gedrückte CTRL-Taste. Die Zuordnung eines bestimmten Zeichens zu einer Taste geschieht über verschiedene Tabellen, die den ASCII-Kode bestimmen, der einer bestimmten Taste zugeordnet wird. Da im Commodore 64 alle Tasten vierfach belegt sein können, existieren 4 solcher Tabellen. Beachten Sie auch die Unterscheidung zwischen linker und rechter Shift-Taste. Shift Lock ist mit der linken Shift-Taste verbunden.

Spalte	0	1	2	3	4	5	6	7
Zeile	-----							
0	DEL	RETURN	CURRIGHT	F7	F1	F3	F5	CURDOWN
1	3	W	A	4	Z	S	E	SHIFT LINKS
2	5	R	D	6	C	F	T	X
3	7	Y	G	8	B	H	U	V
4	9	I	J	O	M	K	O	N
5	+	P	L	-	.	:	@	,
6	PFUND	*	;	HOME	SHIFT	RECHTS	=	↑
7	1	PFL	CTRL	2	LEER	C=	Q	STOP

Die erste Tabelle gibt den ASCII-Kode, wenn die Taste allein gedrückt wurde. In der zweiten Tabelle steht der Kode mit gedrückter Shift-Taste, Tabelle 3 gilt für gedrückte Commodore-Taste und Tabelle 4 schließlich ist für die Control-Taste zuständig. Ein Eintrag von \$FF = 255 in dieser Tabelle markiert einen ungültigen Eintrag. Die Tasten 'Shift', 'Commodore' und 'CTRL' werden gesondert behandelt; die entsprechenden Einträge in der ersten Tabelle sind 1, 2 und 4. Dieser Status wird zusätzlich in \$28D = 653 gespeichert. Die Bits 0, 1 und 2 entsprechen diesen drei Tasten.

Wollen wir nun einer Taste einen anderen Kode zuordnen, müssen wir den entsprechenden Eintrag in der Tabelle ändern. Da die Tabelle im ROM liegt, ist dies nicht ohne weiteres möglich. Der Commodore 64 verfügt jedoch auch über RAM in diesem Speicherbereich, so daß wir das Betriebssystem in das 'darunter' liegende RAM kopieren und dort die Änderung vornehmen können. Dies kann mit einem kleinen BASIC-Programm geschehen. Gleichzeitig muß auch das BASIC ins darunterliegende RAM kopiert werden.

```

100 for i = 40960 to 49151: rem basic-rom kopieren
110 poke i, peek(i) : next
120 for i = 14*4096 to 65535 : rem betriebssystem kopieren
130 poke i, peek(i) : next
140 poke 1, 53

```

Die Zeilen 100 bis 130 kopieren das Betriebssystem und Basic vom ROM in das darunter liegende RAM. In Zeile 140 wird dann vom ROM auf RAM umgeschaltet, so daß das Betriebssystem jetzt im RAM läuft. Jetzt können wir hingehen und die Codes einzelner Tasten ändern.

Bei der deutschen Tastatur sind normalerweise Y und Z gegenüber der amerikanischen Tastatur vertauscht, d.h. Z liegt neben T und Y liegt neben X.

Dazu brauchen wir zuerst die Adressen der 4 Tabellen:

Tabelle 1	ungeshifted	\$EB81 = 60289
Tabelle 2	mit shift	\$EBC2 = 60354
Tabelle 3	mit Commodore	\$EC03 = 60419
Tabelle 4	mit Control	\$EC78 = 60536

Wollen wir jetzt einen Kode ändern, so müssen wir aus der Matrixtabelle die Nummer der Tasten ermitteln, die wir ändern wollen. Die Nummerierung läuft von 0 links oben bis 63 rechts unten. Y hat also die Nummer 25 und Z die Nummer 12. In diese Adressen müssen wir also nun den neuen Kode schreiben.

```

150 t1 = 60289 : rem tabelle 1 ohne shift
160 t2 = 60354 : rem tabelle 2 mit shift
170 t3 = 60419 : rem tabelle 3 mit commodore
180 t4 = 60536 : rem tabelle 4 mit control

```



```
200 poke t1+25, asc("z")
210 poke t1+12, asc("y")
```

Drücken Sie jetzt die Taste "Z", so erscheint ein "y" auf dem Bildschirm und umgekehrt. Um Klein- und Großbuchstaben zu ändern, müssen Sie die Einträge in Tabelle 2 ebenfalls ändern:

```
220 poke t2+25, asc("Z")
230 poke t2+12, asc("Y")
```

Werden die entsprechenden Controltasten benötigt, so können Sie auch diese ändern:

```
240 poke t4+25, asc("z")-64
250 poke t4+12, asc("y")-64
```

Mit Hilfe dieser 4 Tabellen können Sie also 4*64 gleich 256 verschiedene Zeichen definieren. Die RESTORE-Taste können Sie nicht umdefinieren; sie ist direkt mit der NMI-Leitung des Prozessors verbunden. Die Tastaturdefinition bleibt solange erhalten, bis Sie Stop/Restore drücken. Dabei wird wieder auf das ROM zurückgeschaltet. Aber auch dies können Sie verhindern, indem Sie den Wert für die Speicherkonfiguration im RAM ändern. Dies könnte in unserem obigen Programm z.B. in Zeile 190 geschehen:

```
190 poke 64982, 53
```

Anstatt die Nummer der Taste aus der Matrix zu ermitteln, kann man sie auch vom Programm selbst bestimmen lassen. Ein Programm mit automatischer Ermittlung der Nummer könnte z.B. so aussehen:

```
100 dim t(4): for i=1 to 4: read t(i): next
110 data 60289,60354,60419,60536
120 for i=14*4096 to 65535: poke i, peek(i): next
130 for i=40960 to 49151 : poke i, peek(i) : next
140 poke1, 53: poke64982, 53
1000 print "bitte druecken sie die zu aendernde taste ";
1010 get a$: if a$="" then1010
1020 print a$
1030 a = asc(a$)
1040 for j=1 to 4: t=t(j)
1050 for i=0 to 63: if peek(t+i) <> a then next: next
1060 print "bitte gewuenschte belegung angeben ";
1070 get a$: if a$="" then 1070
1080 print a$
1090 poke t+i, asc(a$): goto 1000
```

KAPITEL 3 DATENEINGABE MIT KOMFORT

3.1 CURSORPOSITIONIERUNG UND ABFRAGE DER CURSORPOSITION

Für eine komfortable Ein- und Ausgabe auf dem Bildschirm wäre es sehr nützlich, den Cursor direkt auf jede gewünschte Position des Bildschirms setzen zu können. Der Commodore 64 hat lediglich Befehle zur Positionierung des Cursors in einer Zeile, den TAB-Befehl sowie die Funktion POS zur Abfrage der Spalte. Mit dem TAB-Befehl ist zudem nur eine Positionierung nach vorne möglich.

Das Betriebssystem hält jedoch schon die Routinen zur Positionierung des Cursors bereit. In zwei Speicherstellen in der Zeropage stehen jeweils die Zeile und Spalte, in der sich der Cursor gerade befindet. Durch Abfrage dieser Werte mit PEEK können wir jederzeit die Position erfahren.

```
100 PRINT "DER CURSOR STEHT IN ZEILE " PEEK(214)
    "SPALTE" PEEK(211)
```

Wollen wir den Cursor setzen, so genügt es nicht, in Adresse 214 und 211 die entsprechenden Wert zu POKen. Das Betriebssystem muß noch aus der Cursorposition die erforderlichen Zeiger für Bildschirm- und Farb-RAM berechnen. Dafür steht jedoch eine Routine im Betriebssystem zur Verfügung.

```
100 REM CURSOR SETZEN
110 INPUT "ZEILE ";Z
120 INPUT "SPALTE";S
130 POKE 214,Z
140 POKE 211,S
150 SYS 58640
```

```
160 PRINT "TEST";
```

Durch den Aufruf dieser Routine wird der Cursor an die gewünschte Stelle gesetzt.

Durch Kombination dieser beiden Verfahren ergeben sich neue Möglichkeiten der Programmierung. Sie können in Ihrem Programm z.B. eine Statuszeile vorsehen, in der von Fall zu Fall Anweisungen an den Bediener ausgegeben werden. Um die normale Ausgabe dadurch in ihrem Verlauf nicht zu stören, merken Sie sich vorher die augenblickliche Position des Cursors und setzen dann den Cursor zu Ausgabe in die Statuszeile. Hier geben Sie nun die Meldung aus. Anschließend setzen Sie den Cursor wieder an die ursprüngliche Stelle und fahren im Programmablauf fort. Ein Programmausschnitt sieht dann etwa so aus:

```
...  
300 Z=PEEK(214): REM ZEILE  
310 S=PEEK(211): REM SPALTE  
320 POKE 214,0: REM CURSOR IN ZEILE 0  
330 POKE 211,10: REM CURSOR IN SPALTE 10  
335 SYS 58640  
340 PRINT "BITTE DISKETTE EINLEGEN"  
350 POKE 214,Z: REM ZEILE WIEDER HERSTELLEN  
360 POKE 211,S: REM SPALTE  
370 SYS 58640  
...
```

Die Nummerierung der Zeilen und Spalten geht von 0 bis 24 bzw. von 0 bis 39.

3.2 CURSOR EIN- UND AUSSCHALTEN

Der Cursor zur Markierung der momentanen Bildschirmposition wird beim Commodore 64 automatisch immer dann eingeschaltet, wenn der Rechner eine Eingabe erwartet. Dies ist z.B. dann der Fall, wenn ein INPUT-Befehl benutzt wird oder wenn Sie im Direktmodus arbeiten. Machen Sie aber eine Eingabe mit GET, so erscheint kein Cursor. Dies möchte man manchmal vermeiden, damit der Benutzer durch den blinkenden Cursor auf die erforderliche Eingabe hingewiesen wird.

Der Commodore 64 hat jedoch eine Speicherstelle, die als Flag für den Cursor funktioniert. Enthält die Speicherstelle den Wert eins (oder einen anderen Wert als Null), so weiß der Rechner, daß der Cursor ausgeschaltet ist und die entsprechende Programmstelle im Betriebssystem (während des Interrupts) wird übersprungen. Ein Wert von Null hingegen bedeutet für den Rechner, daß der Cursor blinkt.

Diese Tatsache können wir uns zunutze machen, wenn wir den Cursor programmgesteuert ein- und ausschalten wollen. Wir können z.B. vor einem GET-Befehl den Cursor einschalten, dann auf den Tastendruck warten und anschließend den Cursor wieder ausschalten.

```
100 POKE 204,0: REM CURSOR EIN
110 GET A$: IF A$ = "" THEN 110: REM WARTEN AUF TASTENDRUCK
120 POKE 204,1: REM CURSOR AUS
130 PRINT A$;
```

Hierbei kann es passieren, daß der Cursor gerade dann ausgeschaltet wird, wenn er in der Blinkphase ist. Auf dem Bildschirm bleibt dann ein weißes Quadrat stehen. Dies kann

man vermeiden, wenn man vor dem Ausschalten abfragt, ob der Cursor gerade in der Blinkphase ist. Auch dafür gibt es eine Speicherstelle in der Zeropage. Wir fügen dazu noch die Zeile 115 ein.

```
115 IF PEEK(207) THEN 115: REM WARTEN AUF CURSOR AUS
```

Eine Anwendung dieser Technik finden Sie in Kapitel 3.6

3.3 REPEAT-FUNKTION FÜR ALLE TASTEN

Beim Arbeiten mit Ihrem Commodore 64 werden Sie sicher schon festgestellt haben, daß die Cursorsteuertasten sowie die Leertaste mit einer Repeatfunktion belegt sind. Drücken Sie diese Tasten, so werden sie automatisch solange wiederholt, bis Sie die Tasten wieder loslassen. Dies ist besonders zur Cursorpositionierung und beim Editieren von Programmen nützlich. Mit einem POKE-Befehl kann der Commodore 64 jedoch alle Tasten mit einer Repeatfunktion belegen. Dies ist zum Beispiel bei Textverarbeitung sehr nützlich. Die Umschaltung kann beliebig im Programm oder im Direktmodus durchgeführt und auch wieder rückgängig gemacht werden. Die Adresse zum Umschalten ist 650. Ein Wert von 0 bedeutet, daß nur die Cursortasten automatisch wiederholt werden. Schreiben Sie mit POKE den Wert 128 in die Speicherstelle 650, dann werden alle Tasten wiederholt. Es ist auch möglich, die Repeatfunktion vollkommen abzuschalten. Dazu muß der Wert 64 in Adresse 650 stehen.

```
100 POKE 650,128: REM REPEAT FUER ALLE TASTEN
```

```
...
```

```
200 POKE 650,0 : REM REPEAT NUR FUER CURSOR
```

```
...
```

```
300 POKE 650,64 : REM REPEAT GANZ ABSCHALTEN
```

Die Werte bis zum Ansprechen der Taste (Repeatverzögerung) und die Wiederholrate stehen in den Speicherstellen 651 und 652. Diese Werte werden jedoch von Betriebssystem immer neu gesetzt, so daß Änderungen nur durch Verlegen des Betriebssystems ins RAM möglich sind (siehe dazu auch Kapitel 2.8 und 4.2).

3.4 DER WAIT-BEFEHL: WARTEN AUF EINEN TASTENDRUCK

Ein BASIC-Befehl, der nur äußerst selten angewandt wird, ist der WAIT-Befehl. Wie funktioniert dieser Befehl?

WAIT A,B

Bei diesem Befehl wird der Inhalt der Speicherzelle A geholt (wie bei der PEEK-Funktion). Dieser Wert wird über AND mit B verknüpft. Ist das Ergebnis Null, dann wird das Ganze nochmal wiederholt. Ist der Wert ungleich Null, wird im Programmablauf fortgefahren. Aus dieser Beschreibung geht schon hervor, daß der Wert A entweder die Adresse eines I/O-Ports oder eines anderen Peripheriebausteines sein muß oder daß der Wert von A von der Interruptroutine geändert werden muß. Ansonsten würde der Befehl gar nicht oder unendlich lange warten. Eine Anwendung finden Sie bei der Datenübertragung zwischen zwei Rechnern im Kapitel 7.2

Die interessanteste Anwendung ist sicher das Warten auf bestimmte Tastendrücke. Die Speicherzelle 653 enthält z.B. die Information darüber ob die drei Sondertasten SHIFT, CONTROL oder COMMODORE gedrückt sind. Sie können mit dem WAIT-Befehl solange warten, bis eine der drei Tasten gedrückt wurde.

```
100 PRINT "BITTE DRUECKEN SIE DIE CONTROL-TASTE"  
110 WAIT 653,4: REM WARTEN AUF CTRL  
120 ...
```

Das Programm wartet in Zeile 110 solange, bis die Control-Taste gedrückt wird. Auf die Tasten SHIFT und COMMODORE können Sie mit folgenden WAIT-Befehlen warten:

WAIT 653,1: REM WARTEN AUF SIHFT

WAIT 653,2: REM WARTEN AUF COMMODORE

Wollen Sie auf einen beliebigen Tastendruck warten, können Sie die Speicherstelle 203 abfragen. Ist keine Taste gedrückt, enthält die Adresse den Wert 64, ansonsten steht dort die Matrixnummer der gedrückten Taste (siehe auch kapitel 2.5). Mit

WAIT 203,64

wartet das Programm solange, wie eine Taste gedrückt ist. Mit

WAIT 203,63

wird auf einen beliebigen Tastendruck gewartet. Dies kann man z.B. beim GET-Befehl benutzen, um die Abfrage auf den Leerstring zu sparen.

100 WAIT 203,63

110 GET A\$: PRINT A\$;

Der WAIT-Befehl wird nur dann beendet, wenn zu diesem Zeitpunkt eine Taste gedrückt wird. Standen eventuell schon Daten im Tastaturpuffer, können Sie auch die Anzahl der gedrückten Tasten zur Basis der WAIT-Abfrage machen.

100 WAIT 198,255: REM DATEN IM TASTATURPUFFER ?

110 GET A\$: PRINT A\$;

120 GOTO 100

3.5 DIE BELEGUNG DER FUNKTIONSTASTEN DES COMMODORE 64

Der Commodore 64 hat neben seinen alphanumerischen Tasten noch 4 Funktionstasten, die doppelt belegt sind. Diese Tasten kann man z.B. bei der Menuesteuerung benutzen, um bestimmte Programmteile auszuwählen. Dazu kann man mit GET die Tastatur abfragen und abhängig von der gedrückten Taste die Verzweigung vornehmen. Die Funktionstasten haben folgende ASCII-Kodes:

```
F1 => 133
F3 => 134
F5 => 135
F7 => 136
```

Bei gleichzeitig gedrückter Shifttaste erhöht sich der ASCII-Kode um 4:

```
F2 => 137
F4 => 138
F6 => 139
F8 => 140
```

Eine Funktionstastenabfrage kann man z.B. so verwirklichen:

```
100 GET A$ : IF A$="" THEN 100
110 A = ASC (A$)
120 IF A = 133 THEN 1100 : REM F1
130 IF A = 134 THEN 1200 : REM F3
140 IF A = 135 THEN 1300 : REM F5
150 IF A = 136 THEN 1400 : REM F7
160 IF A = 137 THEN 1500 : REM F2
170 IF A = 138 THEN 1600 : REM F4
```

```

180 IF A = 139 THEN 1700 : REM F6
190 IF A = 140 THEN 1800 : REM F8
200 GOTO 100

```

Abhängig von der gedrückten Funktionstaste wird in die entsprechende Routine verzweigt. Eleganter kann man dies mit einem ON .. GOTO Sprungverteiler lösen.

```

100 GET A$ : IF A$="" THEN 100
110 A = ASC (A$) : IF A<133 OR A>140 THEN 100
120 ON A-132 GOTO 1100,1200,1300,1400,1500,1600,1700,1800

```

Diese Technik können Sie innerhalb von Programmen benutzen. Wir wollen Ihnen jetzt jedoch ein Programm vorstellen, daß es erlaubt, jeder Funktionstaste einen Text zuzuweisen, der beim Drücken der Taste auf dem Bildschirm erscheint. Damit können Sie sich die Arbeit mit dem Rechner komfortabler gestalten. Meist wird man die Funktionstasten mit BASIC-Befehlsworten belegen, z.B. RUN, LIST, PRINT, GOTO. Ebenso wird es möglich sein, einer Funktionstaste ein Wort einschließlich der abschließenden RETURN-Taste zuzuweisen. Damit werden die obigen Befehle dann direkt ausgeführt. Steht z.B. auf der Taste F1 der Text LIST und der Kode für RETURN, so wird durch Drücken der Taste F1 das Programm gelistet, ohne daß eine weitere Tastenbetätigung erforderlich wäre. Die maximale Länge eines Textes bei der Belegung der Funktionstasten ist 10 Zeichen, die Länge des Tastatureingabepuffers.

Als Besonderheit können Sie mit unserem Programm nicht nur 8 verschiedene Texte auf die Funktionstasten legen (doppelte Belegung - mit und ohne Shift-Taste), sondern sogar 16. Zur Unterscheidung dienen dabei außer der Shifttaste noch die Commodore- und die Control-Taste. Wir wählen folgende Zuordnung zu den Tasten:

```

F1 => F1
F2 => F3
F3 => F5
F4 => F7
F5 => F1 plus SHIFT
F6 => F3 plus SHIFT
F7 => F5 plus SHIFT
F8 => F7 plus SHIFT
F9 => F1 plus COMMODORE
F10 => F3 plus COMMODORE
F11 => F5 plus COMMODORE
F12 => F7 plus COMMODORE
F13 => F1 plus CTRL
F14 => F3 plus CTRL
F15 => F5 plus CTRL
F16 => F7 plus CTRL

```

Wir stellen Ihnen nun ein Maschinenprogramm vor, daß die Belegung der Funktionstasten mit den beliebigen Texten gestattet. Diese Texte sind als DATA's abgelegt und können bis zu 10 Zeichen lang sein.

```

;
; Funktionstasten für cbm 64
;

190: 033c      *= 828      ; Kassettenpuffer
200: 033c      keyvec  = $28f ; Vector für tastaturdekodierung
210: 033c      keypnt  = $f5  ; Zeiger auf Dekodiertabelle
220: 033c      buffer  = $277 ; Tastaturpuffer
230: 033c      nmbkeys = $c6  ; Anzahl der Zeichen im Tastaturpuffer
240: 033c      shift   = $28d ; Flag für SHIFT/ CBM/ CTRL
250: 033c      keyno   = $cb  ; Matrixnummer der gedrückten Taste
260: 033c      lastkey = $c5  ; Nummer der letzten Taste

```

```

265: 033c      temp      =      lastkey
270: 033c      fmin      =      $85      ; Code der niedrigsten Funktionstaste
280: 033c      fmax      =      $88      ; Code der höchsten Funktionstaste
285: 033c      setflag   =      $eb26
287: 033c      oldkey    =      $eb48      ; alte Tastaturauswertung

;

300: 033c a9 47      init      lda      #<funktion ; Vector auf neue Routine setzen
300: 033e a0 03              ldy      #>funktion
310: 0340 8d 8f 02              sta      keyvec
310: 0343 8c 90 02              sty      keyvec+1
310: 0346 60              rts

;

330: 0347 a4 cb      funktion ldy      keyno ; Nummer der Taste
335: 0349 c4 c5              cpy      lastkey ; gleiche wie vorher ?
335: 034b f0 0a              beq      nofunc ; ja
340: 034d b1 f5              lda      (keypnt),y ; Ascii-Code
350: 034f c9 89              cmp      #fmax+1 ; mit höchster Funktionstaste
                                vergleichen

360: 0351 b0 04              bcs      nofunc ; keine Funktionstaste ?
370: 0353 c9 85              cmp      #fmin
380: 0355 b0 03              bcs      function
390: 0357 4c 48 eb nofunc      jmp      oldkey ; zur alten Tastaturauswertung
400: 035a e9 85      function      sbc      #fmin
400: 035c 85 c5              sta      temp
410: 035e 0a              asl
410: 035f 0a              asl      ; mal 10
410: 0360 65 c5              adc      temp
410: 0362 0a              asl
420: 0363 ae 8d 02          ldx      shift ; Flag SHIFT/CBM CTRL
430: 0366 e0 01              cpx      #1 ; SHIFT ?
430: 0368 f0 0e              beq      shiftkey
440: 036a e0 02              cpx      #2 ; CBM ?
440: 036c f0 07              beq      cbmkey
450: 036e e0 04              cpx      #4 ; CTRL ?

```

```

450: 0370 d0 09          bne nospec
500: 0372 18      ctrlkey clc
500: 0373 69 28          adc #40
510: 0375 18      cbmkey  clc
510: 0376 69 28          adc #40      ; Zeiger auf nächste Tabelle
520: 0378 18      shiftkey clc
520: 0379 69 28          adc #40
530: 037b aa      nospec  tax          ; Zeiger als Index
540: 037c a0 00          ldy #0
550: 037e bd 00 cf getkey lda tabel,x ; Belegung aus Tabelle holen
550: 0381 f0 09          beq endkey
560: 0383 99 77 02      sta buffer,y ; und in Tastaturpuffer schreiben
570: 0386 e8          inx
570: 0387 c8          iny
570: 0388 c0 0a          cpy #10      ; schon 10 Zeichen ?
580: 038a d0 f2          bne getkey
590: 038c 84 c6      endkey sty nmbkeys ; Anzahl der Zeichen merken
600: 038e a2 ff          ldx #$ff      ; Flag für Tastaturcode ungültig
610: 0390 4c 26 eb      jmp setflag ; Flags aktualisieren
615: 0393          tabel  =   $cf00    ; Tabelle mit Tastenbelegungen

```

Das folgende BASIC-Programm generiert das Maschinenprogramm und legt die Texte für die 16 Funktionstasten im Speicher ab. Die Texte selbst sind als DATA's ab Zeile 300 im Programm hinterlegt und können natürlich nach Ihren Wünschen geändert werden. Beachten Sie dabei jedoch, daß Texte mit mehr als 10 Zeichen nicht zulässig sind. Wollen Sie daß ein Befehl durch Drücken der Funktionstaste direkt ausgeführt wird, so muß das abschließende 'RETURN' mit im Text enthalten sein. Schreiben Sie dazu als letztes Zeichen ein '<' (Pfeil nach links). Dieses Zeichen wird beim Laden dann gegen den RETURN-Kode ausgetauscht (Zeile 250). Wollen Sie Anführungszeichen innerhalb des Textes benutzen, so verwenden Sie anstelle

dessen ein Apostroph ' : LOAD '\$',8 (Zeile 260). Kommt innerhalb eines Textes ein Komma vor, so muß der komplette Text in Hochkommas eingeschlossen werden (siehe Zeile 330).

```
100 for i = 828 to 914
110 read x : poke i,x : s=s+x : next
120 data 169, 71,160, 3,141,143, 2,140,144, 2, 96,164
130 data 203,196,197,240, 10,177,245,201,137,176, 4,201
140 data 133,176, 3, 76, 72,235,233,133,133,197, 10, 10
150 data 101,197, 10,174,141, 2,224, 1,240, 14,224, 2
160 data 240, 7,224, 4,208, 9, 24,105, 40, 24,105, 40
170 data 24,105, 40,170,160, 0,189, 0,207,240, 9,153
180 data 119, 2,232,200,192, 10,208,242,132,198,162,255
190 data 76, 38,235
200 if s <> 10591 then print "fehler in datas !!" : end
210 sys 828
```

```
220 rem belegungen im speicher ablegen
230 ad = 12*4096+15*256 : rem $cf00
240 for i=0 to 7 : read x$ : for j=1 to len(x$)
250 a=asc(mid$(x$,j,1)) : if a=95 then a=13 :rem return
260 if a= 39 then a=34 : rem anführungszeichen
270 poke ad+10*i+j-1,a : next
280 if j<>11 then poke ad+i*10+j-1,0 : rem endekriterium
290 next
300 data list←, run←, goto, chr$(
310 data ?fre(0)←, save, print, then
320 data poke, peek(, print#, input#
330 data "load '$',8", next, gosub, return
```

Die Texte der Funktionstasten werden im freien RAM-Bereich ab Adresse \$CF00 abgelegt. Ist dieser Bereich bei Ihnen belegt oder wollen Sie einen anderen Speicherbereich für die Tasten-

texte benutzen, müssen Sie diese Adresse in Zeile 230 einsetzen und zusätzlich die Adresse in den DATA-Statements in Zeile 170 ändern. Ersetzen Sie dazu das fünft- und vierletzte Element (0 und 207) durch Lo- und High-Byte der neuen Adresse. Wollen Sie eine andere Adresse benutzen, so müssen dort 160 Bytes zur Verfügung stehen (16 Tasten mal 10 Zeichen).

Wollen Sie die Funktionstastenbelegung ändern, so genügt es, das Programm mit entsprechend geänderten Datas ab Zeile 220 laufen zu lassen.

Durch Drücken von RUN/STOP-RESTORE wird die Funktionstastendefinition wieder aufgehoben. Mit

SYS 828

können Sie in einem solchen Falle die Belegungen wieder aktivieren.

3.6 EINE KOMFORTABLE INPUT-ROUTINE

Sicher standen Sie schon oft vor dem Problem, daß Ihr Programm nach einer Eingabe von der Tastatur "ausstieg". Hierfür waren bestimmt in erster Linie zwei Gründe vorhanden:

Eingabe in der Form INPUT A

Hierbei tritt ein Programmabbruch auf, wenn die Eingabe nicht ausschließlich numerische Zeichen beinhaltet.

Eingabe in der Form INPUT A\$

Das Programm wird abgebrochen, wenn die RETURN-Taste ohne vorherige alphanumerische Eingabe betätigt wird.

Die Eingabe mittels GET A\$ ist zwar syntaktisch sicherlich unverfänglicher, jedoch scheuen viele den hierbei erforderlichen Zusammenbau eines Strings, falls die Eingabe mehr als ein Zeichen umfassen soll. Außerdem erscheint hierbei kein blinkender Cursor auf dem Bildschirm, was jedoch als Aufforderung zur Eingabe wünschenswert wäre.

Haben Sie alle Hindernisse bei der Eingabe genommen, so können sich dennoch bei der weiteren Verarbeitung der so gewonnenen Daten Fehler einschleichen, die zwar nicht zum Programmabbruch führen müssen, jedoch das gewünschte Ergebnis in Frage stellen.

Ein typisches Beispiel hierfür ist die numerische Eingabe über INPUT A\$.

Sie verlangen also Zahlen über die Tastatur, die Sie mittels VAL(A\$) in eine numerische Variable zu überführen gedenken. Wohl haben Sie hier den bei INPUT A möglichen Konflikt vermieden, dennoch hat eine falsche Eingabe den folgenden Effekt:

Sie beantworten obigen INPUT mit 123R56. Bei der folgenden Überführung mit A=VAL(A\$) erhalten Sie in A den Wert 123, was sicherlich nicht in Ihrem Sinne lag.

Sie mögen nun einwenden, daß solche Eingabefehler nicht die Regel sind und darüberhinaus falsche Rechenergebnisse in den gemeinhin von Hobbyprogrammierern (was keinerlei Wertung bedeutet) erstellten Programmen keine weitreichenden Folgen zeitigen.

Wir meinen jedoch, daß es Ihr Bestreben sein sollte, "sichere" Programme zu erstellen, auch im Hinblick darauf, daß Sie durchaus einmal in die Verlegenheit kommen könnten, Programme nicht nur für Ihren Eigenbedarf zu schreiben, und was dann Ihr Auftraggeber zu einem absturzgefährdeten Programm sagen wird, können Sie sich ausmalen.

Wir wollen Ihnen hier nun ein gebrauchsfertiges Unterprogramm vorstellen, welches die oben umrissenen Probleme weitgehend, was Programmabbrüche betrifft sogar vollständig eliminiert.

Wir werden dieses Programm im einzelnen besprechen, was Ihnen ermöglichen soll, bestimmte Teile Ihren besonderen Bedürfnissen anzupassen.

Zunächst soll die Bedeutung der im Programm verwendeten Variablen und Speicherzellen erklärt werden.

Folgende Variable müssen beim Ansprung der Routine versorgt sein:

MN=0 Es wird eine rein numerische Eingabe verlangt
MN=1 Die Eingabe darf alphanumerisch sein
ML=0 Die in IL angegebene Länge ist zwingend
ML=1 In IL ist die maximale Länge angegeben
IL Zwingende oder maximale Länge der Eingabe

Des weiteren werden in der Routine verwendet:

CC Anzahl der gültigen Zeichen in IN\$
CS Augenblickliche Spaltenposition des Cursor
CZ Augenblickliche Zeilenposition des Cursor
CP Durch INSERT erzeugte Länge des Eingabefeldes
MS Höchste Spaltenposition des Cursor während der Eingabe
G\$ Enthält Zeichen des letzten GET
IN\$ Hierin wird die vollständige Eingabe zurückgeliefert

Verwendete Speicherstellen:

204=0 Cursor einschalten
204=1 Cursor ausschalten
205 Zähler für Blinkfrequenz des Cursor
207=0 Cursor in AUS-Phase
207=1 Cursor in EIN-Phase
211 Spaltenposition des Cursor
214 Zeilenposition des Cursor

Weitere Voraussetzung für die Funktion dieser INPUT-Routine ist die Öffnung des Bildschirms durch 'OPEN 1,3', da in Zeile 35680 die erzeugte Eingabe vom Bildschirm mittels 'GET#1' in eine Stringvariable gelesen wird.

Nun zu den einzelnen Programmschritten:

35020 Hier werden die benötigten Variablen initialisiert und die Cursorposition für den abschließenden GET#1 in Zeile 35680 gerettet.
35060 Ein Zeichen wird von der Tastatur geholt.
35080 Falls dieses Zeichen ein RETURN war, wird die Eingabe zunächst als beendet angesehen und abhängig vom Wert in ML die Länge überprüft.
35100-35130 Wurde die DELETE-Taste betätigt, so müssen hier

die Längen- und Positionszähler aktualisiert werden in Abhängigkeit davon, ob das Eingabefeld nur gültige Zeichen enthält (CP=0) oder ob es durch INSERT aufgebläht wurde (CP<>0).

35140 INSERT wird nur ausgeführt, wenn dadurch nicht die in IL vorgegebene Länge überschritten wird.

35160-35180 Bei CRSR RIGHT und LEFT muß dafür gesorgt werden, daß der Cursor nicht das Eingabefeld verläßt.

35200 Ansprung des Datenfilters abhängig von MN.

35220-35240 Falls sich der Cursor innerhalb des Datenfeldes befindet und ein reiner Zahlenwert eingegeben wurde, wird das Zeichen akzeptiert. Den Gültigkeitsbereich, hier durch die Werte 47 und 58 festgelegt, können Sie nach eigenen Vorstellungen variieren.

In unserem Beispiel grenzen diese Werte den durch sie repräsentierten Zahlenbereich 0-9 ein. Grundsätzlich erhalten Sie die Werte zur Abgrenzung des Gültigkeitsbereiches, indem Sie sie einfach aus dem Handbuch des C64, Seite 135ff ablesen.

Für unser Beispiel bedeutet das, daß die 47 unter dem Zeichen 0 und die 58 über dem Zeichen 9 steht. Alle Zeichen, die dazwischenstehen, also 0-9, sind gültig.

35300-35380 Hier trifft das gleiche zu wie oben, jedoch ist der Gültigkeitsbereich auch auf Buchstaben erweitert worden.

35400 Falls die Eingabe nicht die vorgeschriebene Länge hat (ML=0), kann sie nicht beendet werden.

35600-35690 Das Eingabefeld wird nun vom Bildschirm nach IN\$ übernommen, und zwar bis entweder die in IL angegebene Länge erreicht ist, oder sich keine

weiteren Daten mehr auf dem Bildschirm befinden. Vorher wird allerdings der Cursor wieder auf die Position gebracht, die er am Beginn der Routine hatte, damit der GETÜ1 auch am Feldanfang startet.

36000-36060 Der Cursor wird ausgeschaltet und das Zeichen in G\$ auf dem Bildschirm zur Anzeige gebracht.

Die Zeilennumerierung der Routine ist willkürlich gewählt. Selbstverständlich kann die Routine auch mit Zeilennr. 1000, 50000 oder einer anderen Nummer anfangen. Wir empfehlen Ihnen jedoch, Subroutinen in allen Programmen stets an die gleiche Stelle zu legen. Dies erleichtert wesentlich die Programmierung und den Aufbau neuer Programme aus einer Baustein-Bibliothek.

Hier nun die INPUT-Routine:

```
35000 rem eingabe tastatur
35020 in$="":cc=0:cs=peek(211):cz=peek(214):cp=0:ms=0
35040 poke204,0:rem cursor ein
35060 getg$:ifg$=""then35060
35080 g=asc(g$):ifg=13thenonml+1goto35400,35600
35100 ifg=20andcp>0thencp=cp-1:gosub36000:goto35060
      :rem delete
35120 ifg=20andcc>0andpeek(211)>csthencc=cc-1:ms=ms-1
      :cp=cp-1:gosub36000:goto35060
35130 ifg=20andpeek(211)>csthensms=ms-1:gosub36000:goto35060
35140 ifg=148andcp+ms<ilthencp=cp+1:ms=ms+1:gosub36000
      :goto35060:rem insert
35160 ifg=29andpeek(211)<cs+il-1thengosub36000
      :goto35060:rem cursor right
35180 ifg=157andpeek(211)>csthengosub36000:goto35060
      :rem cursor left
```

```

35200 onmngoto35300
35220 ifg>47andg<58andcc<ilandpeek(211)<=cs+il-1thencc=cc+1
      :gosub36000:goto35360
35230 goto35360
35240 ifg>47andg<58andpeek(211)<=cs+ilthengosub36000
      :goto35360
35300 ifg<48or(g>57andg<65)or(g>90andg<193)org>218then35060
35320 ifcc<ilandpeek(211)<=cs+il-1thencc=cc+1
      :gosub36000:goto35360
35340 ifpeek(211)<cs+ilthengosub36000
35360 ifcp>0thencp=cp-1
35380 goto35060
35400 ifcc<>ilthen35060
35600 poke205,2
35620 ifpeek(207)<>0then35620
35640 poke204,1
35660 poke211,cs:poke214,cz
35670 ifcc=0thenreturn
35680 get#1,g$:ifg$=chr$(13)then
      in$=left$(in$+"                               ",il):return
35682 in$=in$+g$
35684 iflen(in$)<ilthen35680
35690 return
36000 poke205,2
36020 ifpeek(207)<>0then36020
36040 printg$;:ifpeek(211)>msthenms=peek(211)
36060 return

```

Wie wenden Sie nun das Programm an?

Angenommen, Sie verlangen die Eingabe z.B. einer Artikelnummer für eine Lagerbestandsführung. Diese Nummer muß eine feste Länge von 6 Stellen haben und soll ausschließlich aus numerischen Zeichen bestehen.

Zu diesem Zweck ist folgendes zu programmieren:

10 open1,3

.

.

100 il=6:mn=0:ml=0

110 print"artikelnummer ";;gosub35000

120 in=val(in\$)

Die gewünschte Artikelnummer steht jetzt in IN zur Verfügung und Sie dürfen sicher sein, daß sie exakt 6-stellig ist und ausschließlich Ziffern enthält.

Zu der zuvor eingegebenen Artikelnr. benötigen Sie bei der Neuanlage auch die Bezeichnung des Artikels. Da Sie eine Datei mit Sätzen fester Länge eingerichtet haben, darf diese Bezeichnung max. 10 Stellen umfassen, die jedoch nicht obligatorisch sind.

So sollten die dazugehörenden Programmzeilen aussehen:

.

.

200 il=10:mn=1:ml=1

210 print"bezeichnung ";;gosub35000

Die verlangte Bezeichnung steht jetzt in IN\$ und ist, sofern kürzer, auf die Länge von 10 Zeichen mit Leerzeichen aufgefüllt.

Der Preis ist natürlich auch wichtig. Dieser hat natürlich auch eine variable Länge, nehmen wir einmal an bis zu 8 Stellen, und enthält ausschließlich numerische Zeichen, also:

.

.

300 il=8:mn=0:ml=1

310 print"preis ";;gosub35000

320 in=val(in\$)

Die verlangte, max. 8-stellige Zahl steht jetzt in IN und

Sie können nun an die Eingabe der Menge gehen, usw.

Wir hoffen, Ihnen hiermit eine hübsche kleine Routine an die Hand gegeben zu haben, die Sie Ihren Ärger mit syntaktisch falschen Daten vergessen läßt.

Die hierin verwendeten Besonderheiten hinsichtlich der Cursorpositionierung und der Eingabe vom Bildschirm (GET#1) dürften Sie auch für weitere Programme inspiriert haben.

3.7 DIE "MAUS" AUF DEM CBM64

Ein neues Schlagwort hat die Welt der Personalcomputer heimgesucht: Die "MAUS"

Was verbirgt sich hinter diesem für die Branche etwas ungewöhnlichen Ausdruck?

Sicher kennen Sie von den Videospielautomaten den sogenannten Spinball. Dieses ist, anstelle von Joysticks oder Paddles, ein Betätigungselement, mit dem Sie irgendwelche Figuren über den Bildschirm bewegen können.

Im Gegensatz zu Joysticks, deren Hebel Sie nur in acht Richtungen bewegen können, erlaubt der Spinball ein Drehen in alle Richtungen, da es sich hierbei um eine frei gelagerte Kugel handelt, deren Bewegung auf zwei Winkelcodierer übertragen wird, und zwar auf je einen für die X- und die Y-Achse.

Bei den Spielautomaten wird diese Kugel in der Regel mit der Handfläche in Bewegung gesetzt.

Die "Maus" besteht aus einem ebensolchen Spinball, jedoch ist dieser in ein handliches Gehäuse eingebaut, welches man, Kugel zuunterst, auf den Tisch legt und nun mit der Hand hin- und herrollen kann.

Durch das Bewegen des Gehäuses auf dem Tisch wird die Kugel durch die Reibung auf der Tischfläche in Rotation versetzt.

Durch ein geeignetes Interface wird der jeweilige Stand der beiden Winkelcodierer in den Rechner übertragen.

Die sinnige Bezeichnung dieser Anordnung mag wohl aus der Ähnlichkeit mit einer Aufziehmaus herrühren.

Wie wendet man nun die Maus in durchaus als ernsthaft zu bezeichnenden Programmen an?

Dazu müssen wir ein wenig ausholen:

Wenn ein Programm, welches einem breiten Anwenderkreis zugänglich ist, das Prädikat 'bedienungsfreundlich' zu Recht

tragen soll, wird es in aller Regel nach der sog. Menütechnik aufgebaut sein. Dieses Verfahren hat den Vorteil, daß es nahezu von jedem Bediener gehandhabt werden kann.

Hierbei werden die vom Anwender verlangten Aktivitäten diesem, entweder einzeln oder zur Auswahl aus einer Gruppe von Möglichkeiten, über den Bildschirm mitgeteilt.

Die Auswahl erfolgt entweder durch Eingeben eines Kennbuchstabens oder durch Bewegen des Cursor in ein zu diesem Zweck am Bildschirm aufgebautes Feld.

Nun haben Fachleute, die etwas von Ergonomie verstehen, herausgefunden, daß der Vorgang der Auswahl bequemer und auch sicherer erfolgen kann, wenn nicht auf der Tastatur nach dem Adlersystem der passende Buchstabe herausgesucht werden muß, sondern das Ganze im weichen Sessel zurückgelehnt geschehen kann.

Das setzt natürlich voraus, daß das gesamte Programm derart gehandhabt werden kann, d.h. sämtliche erforderlichen Eingaben müssen auf dieses System abgestimmt sein.

Bei der Maus nun korrespondieren die Bewegungen des Cursor am Bildschirm mit denen des Gerätes auf der Tischfläche. Das Erreichen des gewünschten Feldes wird durch Drücken eines an der Maus befindlichen Knopfes dem Rechner signalisiert.

Soviel zu Anwendung und Entstehungsgeschichte der "Maus".

Um Ihnen die Möglichkeit zu geben, mit diesem reizvollen Tierchen eigene Versuche anzustellen ohne die Anschaffung eines kostspieligen Spinball, haben wir das folgende kleine Programm entwickelt, welches mit einem gewöhnlichen Joystick am Controlport 2 Ihres CBM64 arbeitet.

Natürlich läßt sich damit nicht derselbe Komfort erreichen, wie mit der echten Maus, aber zur Erprobung des Prinzips ist es äußerst zweckdienlich.

Nach bewährter Methode wollen wir Ihnen zunächst die

benutzten Variablen und Speicheradressen vorstellen und anschließend das Programm im Einzelnen besprechen.

Zunächst die Variablen:

RO\$ Zeichen für reverse on
RF\$ Zeichen für reverse off
A\$ jeweils eingegebenes Zeichen
B\$ enthält nach RETURN alle bis dahin eingegebenen Zeichen
A Zweidimensionales Variablenfeld, welches für jede Position der ersten vier Bildschirmzeilen den ASC-Wert des dort abgebildeten Zeichens enthält.
DR ursprünglicher Wert des Datenrichtungsregisters in 56322. Dieser Wert muß beim ordnungsgemäßen Beenden des Programmes wieder dorthin zurückgepoked werden.
J vom Joystick hervorgerufener Wert an Controlport 2
JS Spaltenposition des Joystick
JZ Zeilenposition des Joystick
PS Spaltenposition für PRINT
PZ Zeilenposition für PRINT
S Spaltenposition des Joystickcursor zur Indizierung von A(y,x)
Z wie oben, jedoch Zeilenposition

Nun die Speicheradressen:

56322 Datenrichtungsregister für Controlport 2
58643 Betriebssystemroutine zur Abfrage der Cursorposition
58636 Betriebssystemroutine zur Positionierung des Cursor
781 Inhalt des Prozessorregisters X, welches bei SYS-Befehlen von dort geladen, bzw. dort hinterlegt wird

782 wie oben, jedoch Prozessorregister Y
 204 =0 Cursor einschalten
 =1 Cursor ausschalten
 205 Zähler für Blinkfrequenz des Cursor
 207 =0 Cursor in AUS-Phase
 <>0 Cursor in EIN-Phase

Im folgenden wird das Programm Schritt für Schritt besprochen:

- 1 Da Controlports und Tastatur diegleichen Peripheriebausteine im C64 belegen, wird hier die Tastatur abgeschaltet. Am Ende Ihres Programmes müssen Sie den Wert in DR wieder nach 56322 poken, da der Rechner sonst nicht mehr auf die Tastatur reagiert. Aus unserem Beispielpogramm rettet Sie nur STOP/RESTORE.
- 10-50 Hier wird das Menüfeld am Bildschirm aufgebaut.
- 60-560 Das Array wird mit den ASC-Werten der in den Bildschirmzeilen 1-4 stehenden Zeichen gefüllt. Wird das Array mit der Zeilen- und Spaltenposition indiziert, so liefert es den Wert des an dieser Bildschirmstelle stehenden Zeichens zurück.
- 680 Das von dem Unterprogramm 5000 gelieferte Zeichen wird auf den Bildschirm gebracht.
- 700 Die aus dem PRINT resultierende Cursorposition wird gerettet, da in 720 das Menüfeld neu aufgebaut wird. Das ist notwendig, da bei vollem Bildschirm die Zeilen hochgescrollt werden und das Feld zerstört werden kann.
- 760 Der durch den Neuaufbau des Feldes versetzte Cursor wird wieder an die alte Position gebracht.
- 780 Falls das letzte Zeichen RETURN war, wird die

Eingabe einer Zeile abgebrochen. Wenn Sie eine Weiterverwendung der Daten wünschen, sollten Sie sie hier aus B\$ entnehmen.

800 Andernfalls wird das eingegebene Zeichen an B\$ angehängt.

5020-5140 Der Cursor wird gerettet und ausgeschaltet. Dann wird der Cursor auf die Position 0 gesetzt und wieder eingeschaltet.

5160 Der vom Joystick am Controlport 2 gelieferte Wert wird nach J gebracht.

5170 Verzögerungsschleife. Hierdurch wird der Cursor leichter kontrollierbar.

5180-5340 Entsprechend der Stellung des Joystickhebels wird der Cursor bewegt.

Wurde der Knopf am Joystick gedrückt (5260), so wird das unter dem Cursor stehende Zeichen (ab 6010) nach A\$ übernommen.

6010-6160 Mit der Cursorposition des Joysticks wird das Array A(y,x) adressiert und das daraus resultierende Zeichen nach A\$ gebracht (6060).

Da der C64 eine Doppelzeilenorganisation besitzt, d.h. der Spaltenzähler kann 80 Positionen umfassen, obwohl eine Bildschirmzeile nur 40 Zeichen faßt, wird in 6050 der Spaltenwert zur korrekten Indizierung des Arrays korrigiert.

Die Anwendung dieses Programms ist denkbar einfach.

Nach RUN erscheint in der linken oberen Ecke des Bildschirms der Cursor.

Diesen können Sie nun mittels des Joystick, den Sie hoffentlich in Controlport 2 gesteckt haben, bewegen.

Deckt er sich nun mit einem Zeichen, das Sie gerne nach B\$ übernehmen möchten, so rücken Sie einfach den Knopf.

Zur Quittung erscheint das ausgewählte Zeichen einige Zeilen darunter. Nun suchen Sie das nächste Zeichen aus.

Nach Anwahl von RETURN ist die Zeile komplett in B\$, aus dem Sie die Daten jetzt entnehmen können.

Anschließend geht es auf die gleiche Weise mit der nächsten Zeile weiter.

Wir hoffen, daß Ihnen das Programmchen gefällt und Sie zu eigenen Versuchen anregt.

Abschließend nun das Programmlisting:

```
1 dr=peek(56322):poke56322,224:ro$=chr$(18):rf$=chr$(146)
5 printchr$(147):gosub10:goto60
10 printchr$(19)" , - . / 0 1 2 3 4 5 6 7 8 9
";
20 print" ① a b c d e f g h i j k l ";
30 print" m n o p q r s t u v w x y z ";
40 print" "ro$"ret"rf$" "ro$"del"rf$" "ro$"f1"rf$" "
    ro$"f3"rf$" "ro$"f5"rf$;
45 print" "ro$"f7"rf$" "
50 return
60 dima(4,40)
100 fori=0to13
120 a(0,i*2+1)=i+44
140 nexti
180 fori=0to12
200 a(1,i*2+2)=i+64
220 nexti
260 fori=0to13
280 a(2,i*2+1)=i+77
300 nexti
340 fori=1to3
```

```

360 a(3,i)=13
380 nexti
420 fori=5to7
440 a(3,i)=20
460 nexti
500 fori=0to3
520 a(3,i*2+9)=i+133
540 a(3,i*2+9)=i+133
560 nexti
580 print:print
600 b$="":x=fre(0)
640 gosub5000:rem zeichen holen
680 printa$;
700 sys58643:pz=peek(211):ps=peek(214)
720 gosub10
760 poke211,pz:poke214,ps:remsys58636
780 ifasc(a$)=13then600
800 b$=b$+a$
820 goto640
5000 rem
5001 rem ***** joystick-abfrage *****
5002 rem
5020 sys58643:rem printcursor retten
5060 pz=peek(781):ps=peek(782)
5070 poke205,3
5080 ifpeek(207)then5080
5090 poke204,1
5100 poke781,0:poke782,0:jz=0:js=0
5120 sys58636:rem joystickcursor setzen
5140 poke204,0:rem cursor einschalten
5160 j=peek(56320):rem joystick abfragen
5170 fori=0to50:nexti
5180 if(j and1)=0thenjz=jz-1
5200 if(j and2)=0thenjz=jz+1

```

```

5220 if(j and4)=0thenjs=js-1
5240 if(j and8)=0thenjs=js+1
5260 if(j and16)=0then6000
5280 ifjz<0thenjz=0
5281 ifjs<0thenjs=0
5282 ifjs>30thenjs=30
5283 ifjz>3thenjz=3
5285 poke205,3
5290 ifpeek(207)then5290
5295 poke204,1
5300 poke781,jz:poke782,js:sys58636
5340 goto5140
6000 rem
6001 rem ***** zuordnung zu einem zeichen *****
6002 rem
6010 poke205,3
6015 ifpeek(207)then6015
6017 poke204,1
6020 sys58643:rem cursorposition holen
6040 z=peek(781):s=peek(782)
6050 ifs>39thens=s-40
6060 a$=chr$(a(z,s))
6100 poke781,pz:poke782,ps
6120 sys58636:rem printposition laden
6160 return

```


KAPITEL 4 BASIC FÜR FORTGESCHRITTENE

4.1 ERZEUGEN EINER BASIC-ZEILE IN BASIC

Haben Sie schon einmal versucht, ein wirklich universelles Rechenprogramm zu schreiben?

Unter universell verstehen wir hier ein Programm, mit dem sich beliebige arithmetische Operationen mit beliebigen Variablen oder Konstanten durchführen lassen. Die Betonung liegt auf "beliebig".

Natürlich nicht, werden Sie antworten, denn das Wesen eines Programmes liegt ja darin, nach vorher festgelegten Algorithmen mit den Daten zu verfahren.

Sicher, aber stellen Sie sich einmal vor, Sie wollen eine rechnende Textverarbeitung (sowas gibt es) schreiben.

In Ihrem Text haben Sie numerische Felder, die arithmetisch miteinander verknüpft werden sollen. Das ist z.B. beim Schreiben einer Rechnung der Fall, wo die Preisfelder der Einzelposten addiert werden sollen, die Summe anschließend mit dem Mehrwertsteuersatz multipliziert wird und auch noch Verpackung und Porto hinzukommen.

Sie können selbstverständlich eine Version des oben skizzierten Programmes schreiben, die speziell auf die Bedürfnisse des Rechnungschreibens abgestimmt ist.

Ein solches Programm ist sicher recht nützlich, wenn Sie die Preise der einzelnen Posten kennen und entsprechend eingeben können.

Ein Teppichhändler jedoch, der Auslegeware von der Rolle verkauft, tut sich mit diesem Programm schon wesentlich schwerer, da der Preis seiner Ware von der Quadratmeterzahl abhängt. Diese muß er zunächst berechnen und hat erst nach

der Multiplikation mit dem Quadratmeterpreis den Wert, den er dem beschriebenen Rechnungsprogramm zur Verfügung stellen kann.

Kein Problem, werden Sie sagen, schreiben wir eben eine weitere Version für den Teppichhändler.

Klar, aber wieviele Versionen für wieviele Händler sollen denn dann entstehen?

Wäre es nicht viel praktischer, eine einzige Version zu haben, mit der sämtliche anfallenden Berechnungsarten abgedeckt sind?

Genau das wollen wir Ihnen hier vermitteln, nämlich ein Verfahren, mit dem Sie nachträglich, d.h. während des Programmlaufes festlegen können, welche Variablen wie miteinander verknüpft werden sollen.

Das ist nur möglich, wenn es gelingt, einen Basiczeile mit der gewünschten Formel beliebig oft bei laufendem Programm zu generieren.

Wie das gemacht wird, wollen wir Ihnen jetzt zeigen.

Das im folgenden besprochene Programm enthält auch einen Maschinenspracheteil, aber keine Angst, auch der wird per Basic gehandhabt.

Bevor wir näher auf die einzelnen Programmschritte eingehen, sollen Ihnen zunächst die verwendeten Variablen und Speicheradressen vorgestellt werden.

Als erstes die Variablen:

TM Adresse des Speicherendes

VL niederwertiges Byte der Adresse 'Variablenstart'

VH wie oben, jedoch höherwertiges Byte

VT wie oben, aber Gesamtwert

BU Adresse aus dem Bereich des Zeileneingabepuffers

BC Laufvariable zum Füllen des Puffers

CA\$ eingegebene Rechenvorschrift

RE hierin steht nach Durchlauf der Routine das Ergebnis

Die benutzten Speicherstellen:

45-46 Zeiger auf Beginn der Variablen

47-48 Zeiger auf Start der Arrays

49-50 Zeiger auf Beginn der Strings

56 höherwertiges Byte des Zeigers auf Ende des Basicram

40448 hier steht die erzeugte Basiczeile 50100

40704 ab hier steht die Routine, die aus dem Inhalt des Eingabepuffers eine Basiczeile erzeugt und nach 40448 bringt

Nun das Programm Schritt für Schritt

- 1 Die Speicherobergrenze wird auf 40448 heruntergesetzt, da oberhalb dieser Adresse die Maschinenroutine und später auch die erzeugte Basiczeile steht.
- 2-6 Der Zeiger für Variablenstart wird heraufgesetzt, da hier die Verbindungszeile 50099 eingefügt wird.
- 10-14 Hier werden vorab die Zeilen 50100 und 50110 eingerichtet, damit diese auf jeden Fall vorhanden sind, wenn sie angesprungen werden, ohne daß vorher eine Operation dort hinterlegt worden wäre. Die Zeilen enthalten einfach PRINT und RETURN.
- 20-30 Die Verbindungszeile 50099 wird gleich hinter das Basicprogramm gesetzt.
Die Fortsetzungsadresse dieser Zeile zeigt auf die Zeile 50100 an der Adresse 40449.
- 32-50 Diese Zeilen enthalten das Maschinenprogramm, welches später etwas näher behandelt wird.

60-70 Das Maschinenprogramm wird in den Speicher ab 40704 gebracht.

50040 Die Basiczeile wird von der Tastatur in CA\$ eingelesen.

Berücksichtigen Sie bei der Eingabe bitte, daß nur noch Funktionen einzugeben sind. Stellen Sie sich vor der Eingabe noch die Anweisung 'RE=' als gegeben vor.

50050 Die Eingabe wird aus CA\$ in den Zeileneingabepuffer gebracht (bis 50075).

50080 Das Maschinenprogramm zur Erzeugung der Basiczeile wird angesprungen.

50095 Hier wird die eigentliche Berechnung angesprungen. Das Ergebnis wird in RE zurückgeliefert.

Für diejenigen, die es interessiert, das Maschinenprogramm:

```
lda $7a          Basicpointer retten
sta $9fff
lda $7b
sta $9ffe
lda $14
sta $9ffd
lda $15
sta $9ffc
lda #$0b         Offset auf Inputbuffer
sta $7a
jsr $a579        Aufruf der Routine 'CRUNCH'
ldx #0
XX lda $0200(x)   Zeile nach 40453 transferieren
beq YY           Aussprung wenn Ende
sta $9e05(x)
inx
bne XX
```

```

YY lda #$3a          : hinter die Zeile
   sta $9e05(x)
   lda #$8e          RETURN anhängen
   sta $9e06(x)
   lda #0            Zeilenende markieren
   sta $9e07(x)
   sta $9e08(x)
   sta $9e09(x)
   lda $9fff         Basicpointer zurückladen
   sta $7a
   lda $9ffe
   sta $7b
   lda $9ffd
   sta $14
   lda $9ffc
   sta $15
   ret               zurück nach Basic

```

Das Programm, welches nachstehend gelistet ist, besteht aus zwei Teilen:

Der erste Teil von Zeile 1 bis 70 muß nur einmal beim Programmstart angesprungen werden. Es ist wichtig, daß es auch wirklich am Beginn Ihres Programmes steht, also nicht auf andere Zeilennummern übertragen wird, da sonst in den Zeilen 1 und 6 Ihre Variablen zerstört werden.

Der zweite Teil schließlich produziert aus der in CA\$ eingegebenen Rechenvorschrift eine Basiczeile und führt diese aus. Das Ergebnis steht in RE.

Falls sich die Operation nicht laufend ändert, können Sie natürlich auch die Zeile 50100 mehrmals hintereinander anspringen.

Ihr Programm, in welches diese Routinen eingebunden sind, darf nur bis Zeile 49999 gehen. Die Zeilen ab 50000 müssen

unbedingt die letzten im Programm sein.

Zur Anwendung sei nur gesagt, daß Sie auf die Eingabeanforderung (INPUT CA\$) nur Funktionen eingeben brauchen, z.B. '75/2*V1-V2+SQR(V3)'.

Die Zuordnung des Resultats zu RE ist bereits in Zeile 50050 gegeben.

ACHTUNG! Wenn das Programm einmal bereits gestartet wurde, darf es nicht mehr verändert werden. Sie sollten in dem Falle NEW eingeben, das Programm neu laden und dann ändern. Das rührt daher, daß die erzeugten Zeilen nicht unmittelbar hinter dem übrigen Programm stehen, sondern ganz oben im Speicher. Eine derartige Anordnung kann das Betriebssystem bei Einfügen oder Löschen einer Zeile nicht verwalten.

Hier nun das Programmlisting:

```
1 poke56,158:clr
2 ifpeek(45)+2>255thenpoke45,2-(256-peek(45)):
   poke46,peek(46)+1:goto6
4 poke45,peek(45)+2
6 poke47,peek(45):poke48,peek(46):poke49,peek(45):
   poke50,peek(46)
8 tm=40448
10 poketm,0:poketm+1,7:poketm+2,158:poketm+3,180
   poketm+4,195
12 poketm+5,153:poketm+6,0:poketm+7,13:poketm+8,158:
   poketm+9,190
14 poketm+10,195:poketm+11,142:poketm+12,0:
   poketm+13,0:poketm+14,0
20 v1=peek(45):vh=peek(46):vt=vh*256+v1
30 pokevt-4,1:pokevt-3,158:pokevt-2,179:pokevt-1,175
```

```

32 data165,122,141,255,159,165,123,141,254,159,165,20,
    141,253,159,165,21
33 data141,252,159,169,11,133,122,32,121,165
34 data162,0,189,0,2,240,6,157,5,158,232,208,245,169,
    58,157,5,158
36 data169,142,157,6,158,169,0,157,7,158,157,8,158,
    157,9,158
40 data173,255,159,133,122,173,254,159
50 data133,123,173,253,159,133,20,173,252,159,133,21,96
60 fori=40704to40785
70 read mc:pokei,mc:nexti
50000 rem      calculator *****
50040 bu=523:input"calc";ca$
50050 pokebu,asc("r"):pokebu+1,asc("e"):
    pokebu+2,asc("="):bu=bu+2
50060 fori=1tolen(ca$)
50070 pokebu+i,asc(mid$(ca$,i,1)):nexti
50073 bc=len(ca$)+1
50075 pokebu+bc,0:pokebu+bc+1,0:pokebu+bc+2,0
50080 sys40704
50095 gosub50100
50097 return

```

Zum (hoffentlich) guten Schluß noch eine Anregung zum 'Bebrüten':

Stellen Sie sich vor, Sie haben ein Verfahren gefunden, welches aus einer speziell zu formulierenden Problematik die zur Lösung erforderliche Basiczeile erzeugt. Stellen Sie sich weiter vor, dieses Verfahren sei so universell, daß es in der Lage ist, aus einer Kette von Aufgaben auch eine Reihe von Basiczeilen zu generieren einschließlich aller Schleifen und Sprünge.

Bleibt nur noch das Problem, diese Zeilen hintereinander in

den Speicher zu bringen. Das oben beschriebene Verfahren kann zwar nur eine Basiczeile erzeugen, jedoch ließe es sich so erweitern, daß es auch für mehrere Zeilen taugt.

Zu beachten ist hierbei, daß das Maschinenprogramm die jeweils erzeugte Zeile nicht immer an dieselbe Stelle transferiert, sondern die Startadresse entsprechend der Länge der zuvor erzeugten Zeile heraufsetzen muß.

Außerdem müßten die Zeilenfolgepointer (das sind die ersten beiden Bytes am Beginn jeder Basiczeile) entsprechend versorgt werden, was wir in unserem Beispiel vernachlässigt haben, da der Rücksprungbefehl in der erzeugten Zeile selbst steht.

Vielleicht schaffen Sie es, mit dieser Anregung einen wirklich guten Programmgenerator zu bauen, denn um einen solchen handelt es sich im Prinzip. Programmgeneratoren sind nämlich Programme, die mit bestimmten Aufgaben aus einem speziellen Themenkreis betraut werden und daraus ein Programm in einer bekannten Programmiersprache (es muß nicht immer Basic sein) erzeugen, also eine Art Macrointerpreter.

4.2 KOPIEREN DES BASIC-INTERPRETERS INS RAM

Der Commodore 64 hat gegenüber anderen Commodore-Rechnern den Vorteil, daß der komplette Adressraum des Prozessors - 64 KByte - mit RAM ausgestattet ist. Dadurch kann man also den Commodore 64 mit einem vollkommen neuen Betriebssystem und einem neuen BASIC-Interpreter ausstatten, ohne irgendwelche ROM-Bausteine auszuwechseln. Sie brauchen das neue oder geänderte Betriebssystem oder den BASIC-Interpreter nur in das RAM zu laden oder zu übertragen und dann dem Rechner mitteilen, daß das ROM abgeschaltet und das entsprechende RAM aktiviert werden soll. Dies läßt sich mit POKE-Befehlen erreichen.

Will man kein komplett neues BASIC laden, sondern nur einige Eigenschaften des BASIC-Interpreters ändern, z.B. Implementierung eigener Funktionen oder Abänderung von bestehenden Funktionen oder Befehlen, so kopiert man einfach den BASIC-Interpreter ins RAM, führt dort die Änderungen durch und schaltet dann auf RAM um.

Dazu kurz noch etwas zum Speichermanagment des Commodore 64. Wenn Sie den Rechner einschalten, werden das Betriebssystem-ROM und das BASIC-ROM eingeschaltet und gestartet. Wenn Sie jetzt mit einem PEEK-Befehl diesen Speicherbereich auslesen, so bekommen Sie den Wert aus dem ROM zurück (Wie Sie doch noch an das RAM rankommen, erfahren Sie in Kapitel 9.5). Schreiben Sie dagegen mit POKE in den Speicherbereich, so wird immer in das RAM geschrieben, egal ob ROM oder RAM selektiert ist. Diese Eigenschaft können wir ausnutzen, wenn wir das komplette Betriebssystem oder das BASIC-ROM in das darunter liegende RAM kopieren wollen, um es im RAM für unsere Zwecke zu manipulieren. Das Kopieren läßt sich so mit

einer BASIC-Schleife erledigen.

```
FOR I=A TO E : POKE I, PEEK(I) : NEXT
```

Dabei bedeuten A die Anfangsadresse und E die Endadresse. Für das BASIC sind dies die Adressen 40960 (\$A000) und 49151 (\$BFFF); das Betriebssystem liegt von 57344 (\$E000) bis 65535 (\$FFFF).

Mit dieser POKE-Schleife wird der Inhalt des ROMs in das darunter liegende RAM kopiert. Das BASIC läuft jedoch immer noch im ROM ab. Wir müssen dem Rechner erst noch mitteilen, daß das RAM selektiert werden soll.

Dazu dient die Speicherstelle 1, der Prozessorport, der die Speicheraufteilung bedient. Im Normalfall enthält diese Speicherzelle den Wert 55. Wollen Sie das BASIC ins RAM legen, müssen Sie mit POKE 1, 54 das RAM selektieren. Achtung: Sie dürfen diesen POKE erst dann durchführen, wenn Sie mit der obigen Schleife das BASIC von 40960 bis 49151 ins RAM kopiert haben, da Ihr Rechner sonst 'abstürzt'! Wollen Sie auch das Betriebssystem ins RAM legen, so ist dies nur zusammen mit dem BASIC-ROM möglich, da bei der Auswahl dieses RAMs automatisch auch das RAM unter dem BASIC-ROM selektiert wird (siehe dazu auch Kapitel 2.8). Der POKE-Befehl hierzu lautet POKE 1,53. Wollen Sie Ihren BASIC-Interpreter so manipulieren, so kopieren Sie zuerst das ROM, machen die gewünschten Änderungen und speichern das Programm, mit dem Sie die Änderungen gemacht haben, erst ab und schalten dann mit POKE 0,... um. Haben Sie nämlich einen Fehler gemacht, so kann sich Ihr Rechner 'verabschieden' und Sie müssen wieder von vorne anfangen. So können Sie das Programm wieder laden, die Fehler korrigieren und neu laufen lassen bis die Sache so läuft, wie Sie es sich vorgestellt haben.

4.3 KEINE NEGATIVEN ZAHLEN MEHR BEI DER FRE-FUNKTION

Ist es Ihnen auch schon aufgefallen, daß sich Ihr Commodore 64 beim Einschalten zwar mit 38911 Bytes free meldet, auf den Befehl

```
PRINT FRE(0)
```

aber mit -26627 antwortet ?

Hat man eine negative Zahl erhalten, muß man erst 2 hoch 16 gleich 65536 addieren, um den richtigen Wert zu erhalten. Dies ist zwar nicht tragisch, aber lästig. Woran liegt dies nun ?

Dazu müssen wir uns im ROM-Listing die entsprechende Stelle ansehen. (Adresse \$B37D: 64 intern). Dort wird, nachdem die nicht mehr gebrauchten Strings entfernt und deren Speicherplatz wieder freigegeben wird (Garbage Collection) der freie Speicherplatz berechnet: Beginn der Strings (\$33/\$34) minus Ende der Arrays (\$31/\$32). Dieser 16-Bit Integerwert wird nun ins Fließkommaformat gewandelt und zurückgegeben. Hier liegt nun der Fehler. Der Integerwert wird als Zahl mit Vorzeichen behandelt analog den Integervariablen (%), die nur Werte von -32768 bis 32767 haben können. Werden diese Zahlen jedoch als positive Werte betrachtet, können sie im Bereich von 0 bis 65535 liegen. Bei den frühen Commodore-Rechnern war der freie Speicherplatz nie größer als 32767 Bytes, so daß dieser Fehler nie auftrat. Wir müssen jetzt die FRE-Routine so ändern, daß bei der Umwandlung in Fließkommazahlen von einem positiven Integerwert ausgegangen wird, wie dies zum Beispiel auch bei der Ausgabe von Zeilennummern der Fall ist, die ja bekanntlich

größer als 32767 sein können.

Dies sind die Änderungen, die erforderlich sind. Hier sind wir hingegangen und haben den zusätzlichen Kode in einen unbenutzten Bereich des BASIC-Interpreters gelegt.

B38D	4C 55 BF	JMP \$BF55
B390	EA	NOP
BF55	A5 34	LDA \$34
BF57	E5 32	SBC \$32
BF59	A2 00	LDX #\$00
BF5B	86 0D	STX \$0D
BF5D	85 62	STA \$62
BF5F	84 63	STY \$63
BF61	A2 90	LDX #\$90
BF63	4C 49 BC	JMP \$BC49

Die Werte lassen sich mit einer kleinen POKE-Schleife setzen.

```
100 FOR I=40960 TO 49151
110 POKE I, PEEK(I) : NEXT
120 A=11*4096+3*256+8*16+13
130 FOR I=A TO A+3
140 READ X : POKE I,X :NEXT
150 A=11*4096+15*256+5*16+5
160 FOR I=A TO A+16
170 READ X : POKE I,X : NEXT
180 POKE 1,54
200 DATA 76,85,191,234
210 DATA 165,52,229,50,162,0,134,13,133,98,132,99
220 DATA 162,144,76,73,188
```

4.4 RÜCKKEHR INS BASIC-PROGRAMM NACH EINEM LIST-BEFEHL

Wenn man in ein BASIC-Programm einen LIST-Befehl einbaut, so springt das Programm nach der Ausführung des LIST-Befehl immer in den Direkt-Modus zurück. Dies ist dann störend, wenn Sie z.B. Zeilen ausgeben wollen, die etwa eine Funktionsdefinition mit DEF FN enthalten um die jeweilige Funktion auf den Bildschirm anzuzeigen. Ebenso könnten Sie kein Programm-listing von einem Programm aus mehrfach ausgeben, ebenso nicht im Direkt-Modus in einer Schleife, z.B.

```
FOR I=1 TO 2 : LIST : NEXT
```

Hier läßt sich folgendermaßen Abhilfe schaffen: Sie ersetzen am Ende der LIST-Funktion den Sprung zum BASIC-Warmstart durch einen Return-Befehl. Zusätzlich müssen Sie vor dem Aufruf der LIST-Funktion den Zeiger auf den Programmtext retten, da er während der LIST-Funktion verändert wird.

Dazu benötigen wir eine kleine Routine, die diese Aufgaben durchführt und wieder zum BASIC-Interpreter zurückspringt. Da dies sowieso ein kleines Maschinenprogramm erfordert, packen wir gleich noch das Kopieren des BASIC-ROMs ins RAM mit hinein. Wir haben uns dadurch die langsame BASIC-POKE-Schleife erspart.

Wir haben die kleine Routine in den Kassettenpuffer gelegt. Nach dem Eingeben oder Laden wird sie einmal mit SYS 828 aufgerufen und ab sofort läßt sich der LIST-Befehl ohne Programmabbruch verwenden.

```

105: 033c          *= 828      ; Kassettenpuffer
110: 033c      chrptr = $7a      ; Programmzeiger
112: 033c      chrgot = $79      ; holt letztes Zeichen
114: 033c      list  = $a69c     ; LIST-Routine
115: 033c      lstvec = $a042     ; Zeiger auf LIST-Routine
116: 033c      nextst = $a8f8     ; setzt Programmzeiger
                                ; auf nächstes Statement
117: 033c      crlf   = $aad7     ; gibt CR aus
119: 033c a2 20      ldx #32      ; 32 Pages kopieren
120: 033e a9 a0      lda #>$a000 ; Zeiger auf BASIC-Anfang
120: 0340 a0 00      ldy #0
120: 0342 84 22      sty $22
120: 0344 85 23      sta $23
121: 0346 b1 22      loop lda ($22),y ; Transferschleife
121: 0348 91 22      sta ($22),y
121: 034a c8         iny
122: 034b d0 f9      bne loop
122: 034d e6 23      inc $23
122: 034f ca         dex
122: 0350 d0 f4      bne loop
123: 0352 a9 60      lda #$60      ; RTS-Kode
123: 0354 8d 14 a7   sta $a714
123: 0357 a9 ea      lda #$ea      ; NOP-Kode
123: 0359 8d bb a6   sta $a6bb
123: 035c 8d bc a6   sta $a6bc
125: 035f a9 6d      lda #<newlst-1 ; Zeiger auf neue LIST-Funktion
125: 0361 8d 42 a0   sta lstvec
126: 0364 a9 03      lda #>newlst-1
126: 0366 8d 43 a0   sta lstvec+1
127: 0369 a9 36      lda #$36      ; Speicher auf RAM schalten
127: 036b 85 01      sta 1
127: 036d 60         rts
129: 036e a5 7a      newlst lda chrptr

```

```

130: 0370 48          pha          ; Programmzeiger retten
140: 0371 a5 7b       lda  chrptr+1
150: 0373 48          pha
160: 0374 20 79 00     jsr  chrgot  ; letztes Zeichen holen
170: 0377 20 9c a6     jsr  list    ; LIST ausführen
175: 037a 20 d7 aa     jsr  crlf    ; 'CR' ausgeben
180: 037d 68          pla
190: 037e 85 7b       sta  chrptr+1
200: 0380 68          pla          ; Programmzeiger zurückholen
210: 0381 85 7a       sta  chrptr
220: 0383 20 f8 a8     jsr  nextst  ; Zeiger auf nächstes Statement
230: 0386 4c 79 00     jmp  chrgot  ; letztes Zeichen holen

```

Hier wieder das Ladeprogramm in BASIC.

```

100 for i = 828 to 904
110 read x : poke i,x : s=s+x : next
120 data 162, 32,169,160,160, 0,132, 34,133, 35,177, 34
130 data 145, 34,200,208,249,230, 35,202,208,244,169, 96
140 data 141, 20,167,169,234,141,187,166,141,188,166,169
150 data 109,141, 66,160,169, 3,141, 67,160,169, 54,133
160 data 1, 96,165,122, 72,165,123, 72, 32,121, 0, 32
170 data 156,166, 32,215,170,104,133,123,104,133,122, 32
180 data 248,168, 76,121, 0
190 if s <> 9613 then print "fehler in datas !!" : end
200 print "ok !"

```

Wenn Sie folgendes kleine BASIC-Programm einmal ohne und dann nach Aufruf von SYS 828 laufen lassen, sehen Sie den Unterschied.

```

100 PRINT "LIST-TEST"
110 LIST 120
120 GOTO 100

```

4.5 BERECHNETE ZEILENNUMMERN BEI GOTO, GOSUB UND RESTORE

Eine sehr nützliche Änderung des BASIC-Interpreters wollen wir uns jetzt ansehen.

Wollen Sie eine Programmverzweigung oder ein Unterprogramm aufrufen, so müssen Sie die Zeilennummer als Zahl fest eingeben. Für eine komfortablere Programmierung wäre es jedoch wünschenswert, wenn die Zeilennummer berechnet werden könnte, z. B. wie es folgendes kleine Programm annimmt.

```
100 INPUT "ZEILENNUMMER ";Z
110 GOTO Z
...
```

Dies könnte eine umfangreiche ON .. GOTO Reihe ersetzen. Gleiches gilt für den GOSUB-Befehl. Auch hier wollen wir berechnete Zeilennummern zulassen.

Einen bedeutenden Fortschritt wäre auch ein RESTORE Zeilennummer. Hat man z.B. mehrere verschiedene zusammen hängende Datenblöcke, die man mehrfach lesen möchte, so konnte man bisher den READ-DATA Zeiger immer nur zum Programmanfang zurücksetzen und mußte dann evtl. große Mengen an Daten überlesen, bis man zu den gewünschten Daten kam. Ein RESTORE Zeilennummer erlaubt Ihnen nun, den DATA Zeiger gezielt auf jede beliebige Zeile zu setzen.

Die Änderung des GOTO-Befehls läßt sich mit ein paar POKes bewerkstelligen. Wir brauchen lediglich den Aufruf zum Holen der Zeilennummer durch eine Routine ersetzen, die einen beliebigen numerischen Ausdruck holt und diesen in das erforderliche Adressformat umwandelt. Den GOSUB-Befehl haben

wir damit gleich mit modifiziert, da von GOSUB die GOTO-Routine aufgerufen wird.

```
A8A0 20 C0 02 JSR $02C0
```

```
02C0 20 8A AD JSR $AD8A
```

```
02C3 4C F7 B7 JMP $B7F7
```

Hier haben wir den zusätzliche Kode an die Adresse \$02C0 (704) gelegt, die normalerweise frei ist, solange Sie nicht Sprite 11 benutzen.

Der Kode läßt sich mit ein paar Pokes in den Speicher bringen

```
100 FOR I=40960 TO 49151
110 POKE I, PEEK(I) : NEXT
120 A=10*4096+8*256+10*16
130 FOR I=A TO A+2
140 READ X : POKE I,X :NEXT
150 A=704
160 FOR I=A TO A+5
170 READ X : POKE I,X :NEXT
200 DATA 32,192,2
210 DATA 32,138,173,76,247,183
```

Damit haben wir GOTO- und GOSUB-Befehl abgehandelt. Beim RESTORE-Befehl ist etwas mehr erforderlich, da normalerweise gar keine Zeilennummer erlaubt ist. Wir müssen also unterscheiden, ob es sich um einen RESTORE-Befehl mit oder ohne Zeilennummer handelt. Dies ist jedoch nicht schwierig.

```
02C6 D0 03 BNE $02CB ; folgen weitere Zeichen ?
02C8 4C 1D A8 JMP $A81D ; zum alten RESTORE-Befehl
02CB 20 C0 02 JSR $02C0 ; Zeilennummer holen
```

```

02CE 20 13 A6   JSR $A613   ; Adresse der Zeilennummer berechnen
02D1 38         SEC
02D2 A5 5F      LDA $5F     ; Adresse low
02D4 E9 01      SBC #$01    ; eins abziehen
02D6 A4 60      LDY $60     ; Adresse high
02D8 4C 24 A8   JMP $A824   ; weiter wie alter RESTORE-Befehl

```

Den Kode können wir wieder durch Pokes eingeben.

```

300 A=2*256+12*16+6
310 FOR I=A TO A+20
320 READ X : POKE I,X : NEXT
330 DATA 208,3,76,29,168,32,192,2,32,19,166
340 DATA 56,165,95,233,1,164,96,76,36,168

```

Jetzt müssen wir dem Interpreter noch mitteilen, wo die neue RESTORE-Routine steht:

```

400 POKE 40996, 197 : POKE 40997, 2
410 POKE 1,54

```

Zeile 410 schaltet wieder auf RAM um. Ab sofort können Sie den RESTORE-Befehl in drei Varianten benutzen:

Einmal ohne Zeilennummer wie bisher, dann mit einer Zeilennummer sowie mit einem Ausdruck, der ausgewertet wird und die Zeilennummer ergibt. In diesen Fall wird beim nächsten READ-Befehl das DATA-Element gelesen, das als nächstes ab dieser Zeilennummer steht. Existiert die Zeilennummer nicht, erfolgt keine Fehlermeldung 'undef'd statement', sondern der Zeiger wird automatisch auf die nächste Zeile gesetzt. Jetzt wäre z.B. folgende Programmierung möglich.

```

100 GOTO 200
...

```

200 RESTORE 10

...

500 RESTORE

...

800 GOSUB A*2+100

...

900 RESTORE X*100+500

4.6 DER MID\$-BEFEHL

Sie kennen sicher alle die MID\$-Funktion, mit der ein Teil einer Zeichenkette isoliert werden kann. Das folgende Programmbeispiel

```
100 A$ = "TESTSTRING"  
110 B$ = MID$(A$,5,3)  
120 PRINT B$
```

führt zum Ausdruck

STR

Nachfolgend finden Sie nun eine Erweiterung des MID\$-Befehls, die nicht nur das AUSLESEN von Teilen eines Strings ermöglicht, sondern auch das ERSETZEN von Teilen eines Strings. Mit dem neuen Befehl ist z.B. folgende Programmierung möglich:

```
100 A$ = "TESTSTRING"  
110 MID$ (A$,5,3) = "123"  
120 PRINT A$
```

Hier werden 3 Zeichen ab Position 5 durch die Zeichenfolge "123" ersetzt; als Ergebnis entsteht

TEST123ING

Ohne MID\$-Befehl müssten Sie den String in einzelne Teile zerlegen und neu zusammen setzen:

```
100 A$ = "TESTSTRING"  
110 A$ = LEFT$(A$,4) + "123" + MID$(A$,7,3)
```

120 PRINT A\$

Dieser Befehl ist z.B. dann sehr nützlich, wenn Sie in einem Datensatz einzelne Teile (Datenfelder) ersetzen wollen. Dazu definieren Sie einen String in der Länge des Datensatzes und ersetzen mit dem neuen MID\$-Befehl die Werte der einzelnen Datenfelder.

Der Befehl wird wieder mit einem kleinen Maschinenprogramm realisiert.

```

90:   033c           .opt p1
           ; mid$ als pseudovariable
           ;
           ; mid$(stringvariable,position,laenge) = stringausdruck
           ; mid$(stringvariable,position) = stringausdruck
           ;

200:  033c   midcode =   $ca
210:  033c   execut  =   $308   ; vector für statement ausführen
220:  033c   chrget  =   $73
230:  033c   chrgot  =   chrget + 6
240:  033c   execold  =   $a7e7
250:  033c   varnam  =   $45
255:  033c   varadr  =   $49
260:  033c   descrpt =   $64
270:  033c   teststr =   $ad8f
280:  033c   getvar  =   $b08b
290:  033c   setstr  =   $aa52
300:  033c   chkauf  =   $aefa   ; klammer auf
310:  033c   chkzu   =   $aef7   ; klammer zu
320:  033c   chkcom  =   $aefd   ; komma
325:  033c   test    =   $aeff
330:  033c   getbyt  =   $b79e
340:  033c   frmevl  =   $ad9e
350:  033c   illquan =   $b248

```

```

352: 033c      frestr  =   $b6a3
355: 0003              *=   3
360: 0004      laenge  *=  ++1
370: 0005      position *=  ++1
372: 0007      varstr  *=  ++2
375: 0007      gleich  =   $b2
378: 0007      zeig2   =   $50

;

390: 033c              *=   828
400: 033c a9 47      init   lda  #<midtest
410: 033e a0 03              ldy  #>midtest
420: 0340 8d 08 03      sta  execut
430: 0343 8c 09 03      sty  execut+1
440: 0346 60              rts
450: 0347 20 73 00 midtest jsr  chrget
460: 034a c9 ca              cmp  #midcode ; code für mid$ ?
470: 034c f0 06              beq  midstr ; ja
480: 034e 20 79 00      jsr  chrgot
490: 0351 4c e7 a7      jmp  execold ; normales statement ausführen
500: 0354 20 73 00 midstr jsr  chrget ; nächstes zeichen
505: 0357 20 fa ae      jsr  chkaufr ; klammer auf
510: 035a 20 8b b0      jsr  getvar ; variable holen
520: 035d 85 64      sta  descrpt
530: 035f 84 65      sty  descrpt+1
535: 0361 85 49      sta  varadr
535: 0363 84 4a      sty  varadr+1
540: 0365 20 a3 b6      jsr  frestr
545: 0368 a0 00      ldy  #0
545: 036a b1 64      lda  (descrpt),y
545: 036c 48      pha           ; länge
545: 036d f0 2e      beq  ill
550: 036f 20 52 aa      jsr  setstr ; string in ram übertragen
560: 0372 a0 01      ldy  #1
560: 0374 b1 49      lda  (varadr),y

```

560:	0376 85 05	sta varstr ; variablenadresse merken
570:	0378 c8	iny
570:	0379 b1 49	lda (varadr),y
570:	037b 85 06	sta varstr+1
600:	037d 20 fd ae	jsr chkcom
610:	0380 20 9e b7	jsr getbyt ; position holen
620:	0383 8a	txa
630:	0384 f0 17	beq ill
650:	0386 ca	dex
650:	0387 86 04	stx position
660:	0389 20 79 00	jsr chrgot
660:	038c c9 29	cmp #")" ; ausdruck zu ende ?
665:	038e d0 04	bne next
665:	0390 a9 ff	lda #\$ff ; max. länge
665:	0392 d0 0c	bne store
670:	0394 20 fd ae next	jsr chkcom
670:	0397 20 9e b7	jsr getbyt ; länge holen
680:	039a 8a	txa
690:	039b d0 03	bne *+5
700:	039d 4c 48 b2 ill	jmp illquan
710:	03a0 85 03 store	sta laenge
715:	03a2 68	pla
715:	03a3 38	sec
715:	03a4 e5 04	sbc position
717:	03a6 c5 03	cmp laenge
717:	03a8 b0 02	bcs ok
717:	03aa 85 03	sta laenge
720:	03ac 20 f7 ae ok	jsr chkzu ; klammer zu
730:	03af a9 b2	lda #gleich
770:	03b1 20 ff ae	jsr test
780:	03b4 20 9e ad	jsr frmevl ; ausdruck holen
790:	03b7 20 a3 b6	jsr frestr
800:	03ba a0 02	ldy #2
800:	03bc b1 64	lda (descript),y

```

800: 03be 85 51      sta  zeig2+1
800: 03c0 88        dey
800: 03c1 b1 64      lda  (descript),y
800: 03c3 85 50      sta  zeig2
810: 03c5 88        dey
810: 03c6 b1 64      lda  (descript),y
820: 03c8 f0 d3      beq  ill      ; null, dann fehler
840: 03ca c5 03      cmp  laenge
850: 03cc b0 02      bcs  ok1
860: 03ce 85 03      sta  laenge
870: 03d0 a5 05      ok1  lda  varstr
880: 03d2 18        clc
880: 03d3 65 04      adc  position
910: 03d5 85 05      sta  varstr
910: 03d7 90 02      bcc  *+4
920: 03d9 e6 06      inc  varstr+1
940: 03db a4 03      ldy  laenge
950: 03dd 88        loop  dey
950: 03de b1 50      lda  (zeig2),y ; zeichen aus stringausdruck
960: 03e0 91 05      sta  (varstr),y ; in stringvariable übertragen
970: 03e2 c0 00      cpy  #0
970: 03e4 d0 f7      bne  loop
980: 03e6 4c ae a7   jmp  $a7ae ; zur interpreterschleife

```

Nach dem Eingeben des Programms können Sie mit dem Befehl
SYS 828

Ihre Befehlserweiterung initialisieren. Das nachfolgend
wiedergegebene BASIC-Ladeprogramm macht diese Initialisierung
automatisch.


```

100 for i = 828 to 1000
110 read x : poke i,x : s=s+x : next
120 data 169, 71,160, 3,141, 8, 3,140, 9, 3, 96, 32
130 data 115, 0,201,202,240, 6, 32,121, 0, 76,231,167
140 data 32,115, 0, 32,250,174, 32,139,176,133,100,132
150 data 101,133, 73,132, 74, 32,163,182,160, 0,177,100
160 data 72,240, 46, 32, 82,170,160, 1,177, 73,133, 5
170 data 200,177, 73,133, 6, 32,253,174, 32,158,183,138
180 data 240, 23,202,134, 4, 32,121, 0,201, 41,208, 4
190 data 169,255,208, 12, 32,253,174, 32,158,183,138,208
200 data 3, 76, 72,178,133, 3,104, 56,229, 4,197, 3
210 data 176, 2,133, 3, 32,247,174,169,178, 32,255,174
220 data 32,158,173, 32,163,182,160, 2,177,100,133, 81
230 data 136,177,100,133, 80,136,177,100,240,211,197, 3
240 data 176, 2,133, 3,165, 5, 24,101, 4,133, 5,144
250 data 2,230, 6,164, 3,136,177, 80,145, 5,192, 0
260 data 208,247, 76,174,167
270 if s <> 19273 then print "fehler in datas !!" : end
280 sys 828 : print "ok !"

```

Als Beispiel und Test für die neue Funktion kann folgendes Programm dienen:

```

100 DIM A$(10)
110 FOR I = 1 TO 10
120 A$(I) = "TESTSTRING"
130 NEXT
140 FOR I = 1 TO 10
150 MID$(A$(I),I,1) = MID$("1234567890",I,1)
160 NEXT
170 FOR I = 1 TO 10
180 PRINT A$(I)
190 NEXT

```

Als Ergebnis erhalten Sie 10 String, wobei im ersten String das erste Zeichen durch eine "1" ersetzt wurde, im zweiten String steht als 2. Zeichen einen "2" usw.:

1ESTSTRING
T2STSTRING
TE3TSTRING
TES4STRING
TEST5TRING
TESTS6RING
TESTST7ING
TESTSTR8NG
TESTSTRI9G
TESTSTRINO

4.7 INSTR UND STRING\$-FUNKTION

Viele andere Basic-Rechner haben zwei sehr nützliche Stringfunktionen, die dem 64er fehlen. Die eine Funktion, meist STRING\$ genannt, erzeugt mit einem Befehl einen String beliebiger Länge mit einem vorgegebenen ASCII-Kode, die andere Funktion, oft INSTR genannt, prüft ob ein String in einem anderen enthalten ist.

Mit der Kenntnis des Basicinterpreters und der Stringverwaltung des Commodore 64 ist es nun möglich, auf dem 64er auch diese Funktionen zu implementieren. Als Bezeichnung benutzen wir die schon vorhandenen Befehlsworte 'POS' und 'STR\$' des 64ers, die wir zur Unterscheidung durch ein vorangestelltes "!" kennzeichnen.

Die INSTR-Funktion hat dann folgende Syntax:

```
I = !POS( A$,B$,P)
```

Dabei ist A\$ der String, den Sie durchsuchen wollen, B\$ ist der String, dessen Vorkommen in A\$ Sie überprüfen wollen und P ist die Position, ab der im String A\$ gesucht wird. Das Ergebnis wird der Variablen I zugewiesen und ist gleich null, wenn der gesuchte String nicht gefunden wurde. Kommt der zweite String im ersten vor, so enthält I die Position, an der er gefunden wurde. Die Angabe der Position P ist wahlfrei; wird sie nicht angegeben, so wird von Anfang an gesucht. Anstelle von Variablen können Sie überall auch Ausdrücke verwenden oder die Funktion selbst in Ausdrücken, z.B. einer Print-Liste benutzen.

Hier einige Beispiel zur Anwendung:

```
PRINT !POS("ABCDEFGHIJK","D")
```

4

```
IF !POS(A$,"J") THEN PRINT "GEFUNDEN"
```

```
A$ = "TESTSTRING"
```

```
PRINT !POS(A$,"T")
```

1

```
X = !POS(A$,"T",5) : PRINT X
```

6

Die STRING\$-Funktion wird folgendermaßen benutzt:

```
A$ = !STR$ (L,B)
```

oder

```
A$ = !STR$ (L,B$)
```

Hierbei ist A\$ der String, den Sie erzeugen wollen. L ist die Länge, die der erzeugte String erhalten soll und B ist der ASCII-Kode des Strings. Wird anstelle von B ein String verwendet, wird der ASCII-Kode des ersten Zeichen dieses String benutzt. Folgende Beispiele demonstrieren die Anwendung:

```
PRINT !STR$(20,65)
```

```
AAAAAAAAAAAAAAAAAAAAA
```

```
A$ = !STR$(10,"*") : PRINT A$
```

```
*****
```

Das Maschinenprogramm ist im freien RAM-Bereich abgelegt und beginnt bei Adresse 51200.

```

100:  c800                      .opt p1
                                ; instring-funktion
                                ;

135:  c800                      *=  $c800

140:  c800      chkauf  =  $aefa

150:  c800      chkzu   =  $aef7

160:  c800      chkcom  =  $aefd

170:  c800      frmevl  =  $ad9e

180:  c800      chkstr  =  $ad8f

190:  c800      frestr  =  $b6a3

200:  c800      yfac    =  $b3a2

205:  c800      chrget  =  $73

210:  c800      chrgot  =  chrget+6

220:  c800      getbyt  =  $b79b

226:  c800      integer =  $b1aa

229:  c800      descrpt =  $64

230:  c800      stradr  =  $62

231:  c800      adr2    =  $fb

232:  c800      adr1    =  $fb+2

233:  c800      len1    =  3

234:  c800      len2    =  4

235:  c800      anzahl  =  5

236:  c800      start   =  6

237:  c800      typflag =  13

238:  c800      strcode =  $c4

240:  c800      illquan =  $b248

241:  c800      syntax  =  $af08

242:  c800      poscode =  $b9

243:  c800      vector  =  $30a

245:  c800      temp    =  len1

248:  c800 a9 0b          lda  #<testin

248:  c802 a0 c8          ldy  #>testin

248:  c804 8d 0a 03       sta  vector

248:  c807 8c 0b 03       sty  vector+1

248:  c80a 60             rts

250:  c80b a9 00      testin  lda  #0

```

250:	c80d 85 0d	sta typflag
250:	c80f 20 73 00	jsr chrget
251:	c812 c9 21	cmp #"!"
251:	c814 f0 06	beq test2
251:	c816 20 79 00	jsr chrgot
251:	c819 4c 8d ae	jmp \$ae8d
252:	c81c 20 73 00 test2	jsr chrget
252:	c81f c9 b9	cmp #poscode
252:	c821 f0 0a	beq instr
252:	c823 c9 c4	cmp #strcode
252:	c825 d0 03	bne *+ 5
253:	c827 4c b1 c8	jmp string
253:	c82a 4c 08 af	jmp syntax
300:	c82d 20 73 00 instr	jsr chrget
300:	c830 20 fa ae	jsr chkaufr ; klammer auf
310:	c833 20 9e ad	jsr frmevl ; ausdruck holen
320:	c836 20 8f ad	jsr chkstr ; string testen
330:	c839 a5 64	lda descrpt
330:	c83b 48	pha ; stringadresse auf stack
340:	c83c a5 65	lda descrpt+1
340:	c83e 48	pha
350:	c83f 20 fd ae	jsr chkcom ; komma
360:	c842 20 9e ad	jsr frmevl ; zweiter string
380:	c845 20 a3 b6	jsr frestr
390:	c848 f0 64	beq ill ; länge = 0
400:	c84a 85 04	sta len2
410:	c84c 86 fb	stx adr2
420:	c84e 84 fc	sty adr2+1
430:	c850 68	pla
430:	c851 a8	tay
440:	c852 68	pla ; erste stringadresse
450:	c853 20 aa b6	jsr frestr+7
460:	c856 f0 56	beq ill
470:	c858 85 03	sta len1
480:	c85a 86 fd	stx adr1
490:	c85c 84 fe	sty adr1+1

500:	c85e a2 00	ldx #0	; default für 3. parameter
510:	c860 20 79 00	jsr chrgot	
520:	c863 c9 2c	cmp #", "	
530:	c865 d0 07	bne l1	
540:	c867 20 9b b7	jsr getbyt	
550:	c86a 8a	txa	; startposition
560:	c86b f0 41	beq ill	
570:	c86d ca	dex	
580:	c86e 86 06 11	stx start	; startposition im string
590:	c870 20 f7 ae	jsr chkzu	
600:	c873 a5 03	lda len1	
610:	c875 38	sec	; len2 > len1
620:	c876 e5 04	sbc len2	
630:	c878 90 28	bcc end	; ergebnis 0 ?
640:	c87a 69 00	adc #0	
650:	c87c 85 05	sta anzahl	; der verschiebungen
660:	c87e a5 06	lda start	
660:	c880 18	clc	; adresse 1 plus startposition
670:	c881 65 fd	adc adr1	
670:	c883 85 fd	sta adr1	
680:	c885 90 02	bcc *+4	
690:	c887 e6 fe	inc adr1+1	
700:	c889 a0 00 12	ldy #0	
710:	c88b b1 fb 13	lda (adr2),y	
720:	c88d d1 fd	cmp (adr1),y	; zeichen vergleichen
730:	c88f d0 0b	bne l5	; ab nächster position suchen
740:	c891 c8	iny	
740:	c892 c4 04	cpy len2	; schon alle zeichen string2 getestet ?
750:	c894 90 f5	bcc l3	
760:	c896 a4 06	ldy start	
770:	c898 c8	iny	
780:	c899 4c a2 b3 14	jmp yfac	; ergebnis
790:	c89c e6 06 15	inc start	
800:	c89e c6 05	dec anzahl	
810:	c8a0 d0 04	bne l6	; noch nicht fertig ?
820:	c8a2 a0 00 end	ldy #0	; nicht gefunden, null

```

830:  c8a4 f0 f3          beq  14
840:  c8a6 e6 fd      16   inc  adr1
850:  c8a8 d0 df          bne  12      ; string2 adresse erhöhen
860:  c8aa e6 fe          inc  adr1+1
870:  c8ac d0 db          bne  12
880:  c8ae 4c 48 b2 ill    jmp  illquan

;
; string$-funktion
;

900:  c8b1 20 73 00 string jsr  chrget
900:  c8b4 20 fa ae          jsr  chkaufr ; klammer auf
910:  c8b7 20 9e b7          jsr  getbyt+3
920:  c8ba 8a              txa
920:  c8bb 48              pha          ; länge merken
930:  c8bc 20 fd ae          jsr  chkcom
940:  c8bf 20 9e ad          jsr  frmevl
950:  c8c2 24 0d          bit  typflag
960:  c8c4 30 0c          bmi  str      ; string
970:  c8c6 20 aa b1          jsr  integer
980:  c8c9 a5 64          lda  descrpt ; highbyte
990:  c8cb d0 e1          bne  ill      ; > 255
1000: c8cd a5 65          lda  descrpt+1 ; low-byte, länge
1010: c8cf 4c db c8        jmp  str2
1020: c8d2 20 82 b7 str    jsr  $b782 ; setstr, typflag auf numerisch
1030: c8d5 f0 d7          beq  ill      ; länge null
1040: c8d7 a0 00          ldy  #0
1050: c8d9 b1 22          lda  ($22),y ; erstes zeichen
1060: c8db 85 03      str2 sta  temp
1070: c8dd 68              pla          ; länge
1080: c8de 20 7d b4          jsr  $b47d ; frestr
1090: c8e1 a8              tay
1100: c8e2 f0 07          beq  str3
1110: c8e4 a5 03          lda  temp
1120: c8e6 88      loop    dey
1120: c8e7 91 62          sta  (stradr),y ; string erzeugen
1130: c8e9 d0 fb          bne  loop

```



```

1140: c8eb 20 ca b4 str3      jsr $b4ca ; string in descriptorstack bringen
1150: c8ee 4c f7 ae           jmp chkzu

```

Hier wieder für 'Nicht-Maschinen-Programmierer' ein Ladeprogramm in BASIC, das auch bereits die Initialisierung übernimmt.

```

100 for i = 51200 to 51440
110 read x : poke i,x : s=s+x : next
120 data 169, 11,160,200,141, 10, 3,140, 11, 3, 96,169
130 data 0,133, 13, 32,115, 0,201, 33,240, 6, 32,121
140 data 0, 76,141,174, 32,115, 0,201,185,240, 10,201
150 data 196,208, 3, 76,177,200, 76, 8,175, 32,115, 0
160 data 32,250,174, 32,158,173, 32,143,173,165,100, 72
170 data 165,101, 72, 32,253,174, 32,158,173, 32,163,182
180 data 240,100,133, 4,134,251,132,252,104,168,104, 32
190 data 170,182,240, 86,133, 3,134,253,132,254,162, 0
200 data 32,121, 0,201, 44,208, 7, 32,155,183,138,240
210 data 65,202,134, 6, 32,247,174,165, 3, 56,229, 4
220 data 144, 40,105, 0,133, 5,165, 6, 24,101,253,133
230 data 253,144, 2,230,254,160, 0,177,251,209,253,208
240 data 11,200,196, 4,144,245,164, 6,200, 76,162,179
250 data 230, 6,198, 5,208, 4,160, 0,240,243,230,253
260 data 208,223,230,254,208,219, 76, 72,178, 32,115, 0
270 data 32,250,174, 32,158,183,138, 72, 32,253,174, 32
280 data 158,173, 36, 13, 48, 12, 32,170,177,165,100,208
290 data 225,165,101, 76,219,200, 32,130,183,240,215,160
300 data 0,177, 34,133, 3,104, 32,125,180,168,240, 7
310 data 165, 3,136,145, 98,208,251, 32,202,180, 76,247
320 data 174
330 if s <> 30119 then print "fehler in datas !!" : end
340 sys 51200 : print "ok !"

```

4.8 AUTOMATISCHE ZEILENNUMMERIERUNG

In diesem Abschnitt wollen wir Ihnen einen nützlichen Befehl für Ihren Commodore 64 vorstellen, der die Eingabe von Programmen sehr erleichtern kann. Der Befehle 'AUTO' gibt nach jeder Programmzeile, die Sie eingegeben haben, automatisch die Nummer der nächsten Zeile vor. Dabei können Sie das Inkrement, also die Zahl, um die die laufende Zeilennummer jeweils erhöht wird, selbst bestimmen. Die Anwendung des neuen Befehls ist ganz einfach:

Wollen Sie die automatische Zeilennummerierung einschalten, so geben Sie folgenden Befehl ein:

```
SYS 828, Startnummer, Inkrement
```

z.b. SYS 828, 100, 10

Das Inkrement darf dabei Werte bis zu 255 annehmen. Nach der Eingabe des SYS-Befehls wird die Startnummer ausgegeben und der Cursor steht hinter der Zeilennummer. Sie können nun direkt die Programmzeile eingeben und mit RETURN übernehmen. Jetzt wird automatisch die nächste Zeilennummer vorgegeben, in unserem Beispiel 110.

```
100 INPUT "EINGABE"; A$  
110
```

Um den AUTO-Modus zu beenden, Drücken Sie einfach nur RETURN ohne eine weitere Eingabe. Wollen Sie später weitere Programmzeilen eingeben, so genügt die Eingabe von

```
SYS 828
```

ohne weitere Angaben. Es wird dann automatisch bei der nächsten folgenden Zeilennummer weitergemacht, bei der vorher abgebrochen wurde. Natürlich können Sie jederzeit Startwert und Inkrement ändern, indem Sie diese Werte beim SYS-Befehl mit angeben.

Es folgt nun das Listing des Maschinenprogramms, das im Kassettenpuffer abgelegt ist sowie anschließend wieder ein Ladeprogramm in BASIC.

```

150: 033C          *= 828      ; Kassettenpuffer
152: 033C          LO      = $14
154: 033C          HI      = LO+1
156: 033C          FAC     = $62      ; Fließkommaakku
158: 033C          CR      = 13      ; carriage return
160: 033C          LINE    = 251     ; Zeile
162: 033C          INCR    = LINE +2 ; Inkrement
164: 033C          INTFLOAT = $BC49  ; Integer nach Fließkomma
166: 033C          FLOATASCI= $BDDD  ; Fließkomma nach ASCII
170: 033C          VECTOR  = $302    ; ZEILENEINGABE
172: 033C          INPUT   = $FFCF
174: 033C          PRINT   = $FFD2
176: 033C          BUFFER1 = $101
178: 033C          BUFFER2 = $200
180: 033C          MAINLOOP = $A486
182: 033C          GOON    = $A569
183: 033C          WEITER  = $A576
184: 033C          CHRGOT  = $79
186: 033C          OLDVEC  = $A483
187: 033C          GETPAR  = $B7EB
189: 033C          CHKCOM  = $AEFD
190: 033C 20 79 00      JSR CHRGOT ; weitere Zeichen ?
190: 033F F0 10        BEQ LO      ; nein

```

200:	0341 20 FD AE	JSR	CHKCOM ; Komma ?
200:	0344 20 EB B7	JSR	GETPAR ; Parameter holen
210:	0347 86 FD	STX	INCR
220:	0349 A5 14	LDA	LO ; und abspeichern
220:	034B 85 FB	STA	LINE
230:	034D A5 15	LDA	HI
230:	034F 85 FC	STA	LINE+1
240:	0351 A9 5C LO	LDA	#< AUTO
240:	0353 8D 02 03	STA	VECTOR ; Eingabevektor setzen
250:	0356 A9 03	LDA	#> AUTO
250:	0358 8D 03 03	STA	VECTOR+1
260:	035B 60	RTS	
			;
300:	035C 20 62 03 AUTO	JSR	AUTONUM
310:	035F 4C 86 A4	JMP	MAINLOOP
			;
330:	0362 A5 FB AUTONUM	LDA	LINE
330:	0364 A6 FC	LDX	LINE+1
340:	0366 85 63	STA	FAC+1 ; Zeilennummer
340:	0368 86 62	STX	FAC
340:	036A A2 90	LDX	#\$90
340:	036C 38	SEC	
350:	036D 20 49 BC	JSR	INTFLOAT; nach Fließkomma
360:	0370 20 DD BD	JSR	FLOATASCI ; nach ASCII
370:	0373 A2 00	LDX	#0
380:	0375 BD 01 01 L1	LDA	BUFFER1,X ; Ziffern holen
390:	0378 FO 09	BEQ	L2
400:	037A 9D 00 02	STA	BUFFER2,X ; in BASIC-Puffer
400:	037D 20 D2 FF	JSR	PRINT ; und ausgeben
410:	0380 E8	INX	
410:	0381 D0 F2	BNE	L1
420:	0383 A5 FB L2	LDA	LINE ; Zeilennummer
420:	0385 18	CLC	; erhöhen
420:	0386 65 FD	ADC	INCR

```

420: 0388 85 FB          STA LINE
420: 038A 90 02          BCC L3
420: 038C E6 FC          INC LINE+1
425: 038E A9 20      L3   LDA #" "      ; Leerzeichen ausgeben
425: 0390 20 D2 FF          JSR PRINT
425: 0393 20 CF FF          JSR INPUT
430: 0396 C9 0D          CMP #CR      ; leere Eingabe
440: 0398 F0 03          BEQ L4      ; ja
450: 039A 4C 69 A5          JMP GOON      ; weitermachen
460: 039D A5 FB      L4   LDA LINE
460: 039F E5 FD          SBC INCR      ; Zeilennummer für nächsten
460: 03A1 85 FB          STA LINE      ; Zugriff erniedrigen
470: 03A3 B0 02          BCS L5
470: 03A5 C6 FC          DEC LINE +1
480: 03A7 A9 83      L5   LDA #< OLDVEC
480: 03A9 A0 A4          LDY #> OLDVEC
490: 03AB 8D 02 03          STA VECTOR ; alten Vektor
490: 03AE 8C 03 03          STY VECTOR+1 ;wieder herstellen
500: 03B1 4C 76 A5          JMP WEITER

```

```

100 FOR I = 828 TO 947

```

```

110 READ X : POKE I,X : S=S+X : NEXT

```

```

120 DATA 32,121, 0,240, 16, 32,253,174, 32,235,183,134

```

```

130 DATA 253,165, 20,133,251,165, 21,133,252,169, 92,141

```

```

140 DATA 2, 3,169, 3,141, 3, 3, 96, 32, 98, 3, 76

```

```

150 DATA 134,164,165,251,166,252,133, 99,134, 98,162,144

```

```

160 DATA 56, 32, 73,188, 32,221,189,162, 0,189, 1, 1

```

```

170 DATA 240, 9,157, 0, 2, 32,210,255,232,208,242,165

```

```

180 DATA 251, 24,101,253,133,251,144, 2,230,252,169, 32

```

```

190 DATA 32,210,255, 32,207,255,201, 13,240, 3, 76,105

```

```

200 DATA 165,165,251,229,253,133,251,176, 2,198,252,169

```

```

210 DATA 131,160,164,141, 2, 3,140, 3, 3, 76,118,165

```

```

220 IF S <> 15495 THEN PRINT "FEHLER IN DATAS !!" : END

```

```

230 PRINT "OK"

```

4.9 DEF FN X(Y) - DAS UNBEKANNTE WESEN

Viele Programmierer ziehen es vor, anstelle dieser eigentlich komfortablen Funktion - zugegeben, sie ist in den Handbüchern gewöhnlich unvollkommen (bisweilen auch unverständlich) beschrieben - einige Programmzeilen mehr, oft auch Subroutinen einzugeben.

Außer der erwähnten schlechten Dokumentation muß es dafür noch einen weiteren Grund geben: Die Funktion scheint nicht sehr mächtig zu sein, jedenfalls nicht auch den ersten Blick (der leider oft entscheidet über Brauchbarkeit und Unbrauchbarkeit), da offensichtlich nur ein Argument an die Funktion übergeben werden kann.

Wir wollen Ihnen deshalb in diesem Abschnitt zum einen den Umgang mit dieser Funktion veranschaulichen, zum anderen zeigen, daß auch recht komplexe Formelgebilde mit mehreren Variablen durch Schachtelung mehrerer DEF FN X(Y) (das ist tatsächlich möglich) gelöst werden können.

Eine Funktionsdefinition ist folgendermaßen aufgebaut:

DEF FN Name (Funktionsvariable) = arithmetischer Ausdruck

z.B.

DEF FN A(X) = 2*X + B

Unsere Funktion hat den Namen A und die Funktionsvariable X. Wird die Funktion nun aufgerufen, so wird anstelle des X der Wert verwendet, der beim Aufruf übergeben wird. Eventuell existierende Variable mit dem Namen X und A bleiben dabei

unverändert. Die Funktionsvariable X dient nur als Platzhalter für den aktuellen Wert beim Funktionsaufruf. Im Gegensatz dazu bezieht sich B auf die Variable B, die vorher definiert wurde. Die Funktionsvariable wird auch oft als Dummy-Variable bezeichnet. Das Ergebnis einer Funktion muß immer numerisch sein, ein Stringausdruck ist dabei nicht erlaubt.

Zur Einstimmung ein einfaches, oft benötigtes Beispiel, nämlich die kaufmännische Rundung. Wie Ihnen sicher bekannt ist, wird hierbei die dritte Dezimalstelle eines DM-Betrages derart behandelt, daß bei Werten von kleiner 5 abgerundet und bei größer 4 aufgerundet wird.

Die meisten Programmierer führen diese Rundung direkt am Ort des Entstehens in der Form

```
A = INT (B*100+.5)/100
```

durch. Geschieht das öfter, kann man bei Ersatz dieser Formel durch

```
A = FNK(B)
```

ganz schön Platz sparen. Konkret sieht das Ganze so aus:

```
10 DEF FN K(B) = INT(B*100+.5)/100
...
100 A = FNK(B)
```

Den Befehl in Zeile 100 können Sie nun sooft wie nötig anstelle der längeren Formel einsetzen, wobei die Variable B den ungerundeten Wert enthalten sollte.

Nun ein Beispiel für die Schachtelung von Funktionen. Hier soll anhand des Netto-Einkaufspreises der Verkaufspreis unter Zugrundelegung der gewünschten Handelsspanne und des Mehrwertsteuersatzes ermittelt werden.

C ist der Einkaufspreis

H die Handelsspanne in % "von oben"

M der Mehrwertsteuersatz in %

```
10 DEF FN V(C) = (C/(1-H/100))*(1+M/100)
```

```
20 DEF FN K(B) = INT (FN V(C)*100+.5)/100
```

```
...
```

```
100 A = FN K(C)
```

A enthält nach Zeile 100 den Verkaufspreis, gerundet und inklusive Mehrwertsteuer. Natürlich können Funktionen noch tiefer geschachtelt werden. Dabei ist eine maximal 10fache Schachtelung möglich. Versucht man noch tiefer zu schachteln, so erhält man einen 'OUT OF MEMORY ERROR', da die Zwischenergebnisse im Stack gespeichert werden, der nur eine begrenzte Größe hat. Sind während des FN-Aufruf noch Unterprogramme (GOSUB) oder Schleifen (FOR-NEXT) aktiv, die ebenfalls den Stack benötigen, so ist nur eine entsprechend geringere Schachtelung möglich, die jedoch für praktische Anwendungen immer noch genügend sein dürfte.

4.10 IHR COMMODORE 64 SPRICHT DEUTSCH

Stört es Sie eigentlich, daß Ihr Commodore 64 englisch mit Ihnen 'spricht' ? - Dann bringen Sie ihm doch Deutsch bei !

Wenn Sie das folgende Programm auf Ihrem 64er starten, wird Ihr Rechner dazu veranlaßt, sämtliche Fehlermeldungen in deutscher Sprache auszugeben. Im Programm steht eine Liste der Fehlermeldungen in deutscher Sprache. Die Fehlermeldungen sind von 1 bis 29 nummeriert, die erste Fehlermeldung wäre z.B. 'too many files'. Diese Tabelle finden Sie in '64 intern'. Wir ersetzen einfach die Fehlermeldungen dieser Tabelle durch eigene deutsche Texte. Besitzen Sie einen Assembler für den 64er und wollen eventuell selbst andere Texte eingeben, so beachten Sie bitte, daß das Ende eines Textes jeweils durch ein Zeichen mit gesetztem Bit 7 gekennzeichnet ist. Sie können das einfach dadurch erreichen, indem Sie den letzten Buchstaben eines jeden Textes geshiftet eingeben. Für die 'Nur-BASIC-Programmierer' haben wir wieder ein Ladeprogramm abgedruckt.

```
100: c000                                .opt p1
                                           ;
                                           ; deutsche fehlermeldungen c64
                                           ;
140: c000      errvec = $300      ; vector für fehlermeldungen
150: c000      erradr = $22       ; zeiger auf fehlermeldungen
160: c000      olderr = $a445
                                           ;
180: c000                                *= $c000
190: c000 a9 0b  init  lda #<newerr
200: c002 a0 c0      ldy #>newerr
```

```

210:  c004 8d 00 03      sta  errvec
220:  c007 8c 01 03      sty  errvec+1
230:  c00a 60             rts

                ;
250:  c00b 8a      newerr txa          ; fehlernummer
260:  c00c 10 03      bpl  err1
270:  c00e 4c 43 c0    jmp  ready
280:  c011 0a      err1  asl  a          ; mal 2
290:  c012 aa      tax          ; als index
300:  c013 bd 48 c0    lda  errtab-2,x
310:  c016 85 22      sta  erradr
320:  c018 bd 49 c0    lda  errtab-1,x
330:  c01b 85 23      sta  erradr+1
340:  c01d 20 cc ff    jsr  $ffcc      ; clrhc
350:  c020 a9 00      lda  #0
360:  c022 85 13      sta  $13          ; i/o-flag
370:  c024 20 d7 aa    jsr  $aad7      ; neue zeile
380:  c027 20 45 ab    jsr  $ab45      ; "?" ausgeben
390:  c02a a0 00      ldy  #0
400:  c02c b1 22      loop lda  (erradr),y
400:  c02e 48      pha
400:  c02f 29 7f      and  #$7f
410:  c031 20 47 ab    jsr  $ab47
420:  c034 c8      iny
420:  c035 68      pla
420:  c036 10 f4      bpl  loop
430:  c038 20 7a a6    jsr  $a67a      ; cont sperren
440:  c03b a4 3a      ldy  $3a
440:  c03d c8      iny
440:  c03e f0 03      beq  ready
450:  c040 20 c2 bd    jsr  $bdc2      ; in zeilennummer
460:  c043 a9 bb      ready lda  #< fertig
470:  c045 a0 c2      ldy  #> fertig
480:  c047 20 78 a4    jsr  $a478      ; wie gehabt

```

```

490:  c04a 86 c0 96 errtab      .word f1,f2,f3,f4,f5,f6,f7,f8
500:  c05a 13 c1 2b              .word f9,f10,f11,f12,f13,f14,f15
510:  c068 89 c1 9f              .word f16,f17,f18,f19,f20,f21,f22
520:  c076 15 c2 29              .word f23,f24,f25,f26,f27,f28,f29,f30

;
540:  c086 5a 55 20 f1          .asc "zu viele dateien"
550:  c096 44 41 54 f2          .asc "datei offen"
560:  c0a1 44 41 54 f3          .asc "datei nicht offen"
570:  c0b2 44 41 54 f4          .asc "datei nicht gefunden"
580:  c0c6 47 45 52 f5          .asc "geraet nicht vorhanden"
590:  c0dc 4b 45 49 f6          .asc "keine eingabedateI"
600:  c0ee 4b 45 49 f7          .asc "keine ausgabedateI"
610:  c100 46 45 48 f8          .asc "fehlender dateiname"
620:  c113 55 4e 47 f9          .asc "ungueltige geraetenummer"
630:  c12b 27 4e 45 f10         .asc "'next' ohne 'for'"
640:  c13b a7                   .byt "'"+$80
650:  c13c 53 59 4e f11         .asc "syntax fehler"
660:  c149 27 52 45 f12         .asc "'return' ohne 'gosub'"
670:  c15d a7                   .byt "'"+$80
680:  c15e 5a 55 20 f13         .asc "zu wenig 'datas'"
690:  c16d a7                   .byt "'"+$80
700:  c16e 55 4e 5a f14         .asc "unzulaessiger wert"
710:  c180 55 45 42 f15         .asc "ueberlauf"
720:  c189 5a 55 20 f16         .asc "zu wenig speicherplatz"
730:  c19f 46 45 48 f17         .asc "fehlende zeile"
740:  c1ad 55 4e 47 f18         .asc "ungueltiger index"
750:  c1be 46 45 4c f19         .asc "feld bereits dimensioniert"
760:  c1d8 44 49 56 f20         .asc "division durch null"
770:  c1eb 44 49 52 f21         .asc "direkt-modus verboten"
780:  c200 46 41 4c f22         .asc "falscher variabeltyp"
790:  c215 5a 45 49 f23         .asc "zeichenkette zu lang"
800:  c229 46 45 48 f24         .asc "fehlerhafte daten aus dateI"
810:  c244 53 54 52 f25         .asc "stringausdruck zu kompliziert"
820:  c261 46 4f 52 f26         .asc "fortfuehren nicht moeglich"

```

```

830:  c27b 4e 49 43 f27      .asc "nicht definierte funktion"
840:  c294 4b 45 49 f28      .asc "keine uebereinstimmung"
850:  c2aa 4c 41 44 f29      .asc "ladefehler"
860:  c2b4 41 42 42 f30      .asc "abbruch"
870:  c2bb 0d      fertig    .byt 13      ; crlf
880:  c2bc 46 45 52      .asc "fertig."
890:  c2c3 0d 00      .byt 13,0

```

```

100 for i = 49152 to 49860
110 read x : poke i,x : s=s+x : next
120 data 169, 11,160,192,141, 0, 3,140, 1, 3, 96,138
130 data 16, 3, 76, 67,192, 10,170,189, 72,192,133, 34
140 data 189, 73,192,133, 35, 32,204,255,169, 0,133, 19
150 data 32,215,170, 32, 69,171,160, 0,177, 34, 72, 41
160 data 127, 32, 71,171,200,104, 16,244, 32,122,166,164
170 data 58,200,240, 3, 32,194,189,169,187,160,194, 32
180 data 120,164,134,192,150,192,161,192,178,192,198,192
190 data 220,192,238,192, 0,193, 19,193, 43,193, 60,193
200 data 73,193, 94,193,110,193,128,193,137,193,159,193
210 data 173,193,190,193,216,193,235,193, 0,194, 21,194
220 data 41,194, 68,194, 97,194,123,194,148,194,170,194
230 data 180,194, 90, 85, 32, 86, 73, 69, 76, 69, 32, 68
240 data 65, 84, 69, 73, 69,206, 68, 65, 84, 69, 73, 32
250 data 79, 70, 70, 69,206, 68, 65, 84, 69, 73, 32, 78
260 data 73, 67, 72, 84, 32, 79, 70, 70, 69,206, 68, 65
270 data 84, 69, 73, 32, 78, 73, 67, 72, 84, 32, 71, 69
280 data 70, 85, 78, 68, 69,206, 71, 69, 82, 65, 69, 84
290 data 32, 78, 73, 67, 72, 84, 32, 86, 79, 82, 72, 65
300 data 78, 68, 69,206, 75, 69, 73, 78, 69, 32, 69, 73
310 data 78, 71, 65, 66, 69, 68, 65, 84, 69,201, 75, 69
320 data 73, 78, 69, 32, 65, 85, 83, 71, 65, 66, 69, 68
330 data 65, 84, 69,201, 70, 69, 72, 76, 69, 78, 68, 69
340 data 82, 32, 68, 65, 84, 69, 73, 78, 65, 77,197, 85

```

350 data 78, 71, 85, 69, 76, 84, 73, 71, 69, 32, 71, 69
 360 data 82, 65, 69, 84, 69, 78, 85, 77, 77, 69, 210, 39
 370 data 78, 69, 88, 84, 39, 32, 79, 72, 78, 69, 32, 39
 380 data 70, 79, 82, 167, 83, 89, 78, 84, 65, 88, 32, 70
 390 data 69, 72, 76, 69, 210, 39, 82, 69, 84, 85, 82, 78
 400 data 39, 32, 79, 72, 78, 69, 32, 39, 71, 79, 83, 85
 410 data 66, 167, 90, 85, 32, 87, 69, 78, 73, 71, 32, 39
 420 data 68, 65, 84, 65, 83, 167, 85, 78, 90, 85, 76, 65
 430 data 69, 83, 83, 73, 71, 69, 82, 32, 87, 69, 82, 212
 440 data 85, 69, 66, 69, 82, 76, 65, 85, 198, 90, 85, 32
 450 data 87, 69, 78, 73, 71, 32, 83, 80, 69, 73, 67, 72
 460 data 69, 82, 80, 76, 65, 84, 218, 70, 69, 72, 76, 69
 470 data 78, 68, 69, 32, 90, 69, 73, 76, 197, 85, 78, 71
 480 data 85, 69, 76, 84, 73, 71, 69, 82, 32, 73, 78, 68
 490 data 69, 216, 70, 69, 76, 68, 32, 66, 69, 82, 69, 73
 500 data 84, 83, 32, 68, 73, 77, 69, 78, 83, 73, 79, 78
 510 data 73, 69, 82, 212, 68, 73, 86, 73, 83, 73, 79, 78
 520 data 32, 68, 85, 82, 67, 72, 32, 78, 85, 76, 204, 68
 530 data 73, 82, 69, 75, 84, 45, 77, 79, 68, 85, 83, 32
 540 data 86, 69, 82, 66, 79, 84, 69, 206, 70, 65, 76, 83
 550 data 67, 72, 69, 82, 32, 86, 65, 82, 73, 65, 66, 76
 560 data 69, 78, 84, 89, 208, 90, 69, 73, 67, 72, 69, 78
 570 data 75, 69, 84, 84, 69, 32, 90, 85, 32, 76, 65, 78
 580 data 199, 70, 69, 72, 76, 69, 82, 72, 65, 70, 84, 69
 590 data 32, 68, 65, 84, 69, 78, 32, 65, 85, 83, 32, 68
 600 data 65, 84, 69, 201, 83, 84, 82, 73, 78, 71, 65, 85
 610 data 83, 68, 82, 85, 67, 75, 32, 90, 85, 32, 75, 79
 620 data 77, 80, 76, 73, 90, 73, 69, 82, 212, 70, 79, 82
 630 data 84, 70, 85, 69, 72, 82, 69, 78, 32, 78, 73, 67
 640 data 72, 84, 32, 77, 79, 69, 71, 76, 73, 67, 200, 78
 650 data 73, 67, 72, 84, 32, 68, 69, 70, 73, 78, 73, 69
 660 data 82, 84, 69, 32, 70, 85, 78, 75, 84, 73, 79, 206
 670 data 75, 69, 73, 78, 69, 32, 85, 69, 66, 69, 82, 69
 680 data 73, 78, 83, 84, 73, 77, 77, 85, 78, 199, 76, 65

```
690 data 68, 69, 70, 69, 72, 76, 69,210, 65, 66, 66, 82
700 data 85, 67,200, 13, 70, 69, 82, 84, 73, 71, 46, 13
710 data 0
720 if s <> 62278 then print "fehler in datas !!" : end
730 sys 49152 : print "ok !"
```

4.11 BASIC HARDCOPY ROUTINE UND DEREN ANWENDUNG

Haben auch Sie schon oft versucht den Inhalt des Bildschirms auf Ihrem Drucker auszugeben ? Es gibt eine Reihe von verschiedenen Maschinenprogrammen die diese Aufgabe erledigen. Aber auch von BASIC aus ist dieses kleine Problem leicht zu lösen.

Damit Sie den Aufbau des Programmes aber auch verstehen müssen wir uns zunächst noch etwas mit dem Aufbau des Bildschirmspeichers beschäftigen. Denn diesen gilt es ja auf den Drucker zu übertragen.

Wie Sie ja sicherlich schon wissen, stehen Ihnen auf dem Bildschirm 1000 Zeichen zur Verfügung. Die 1000 Zeichen setzen sich zusammen aus 40 Zeichen in 25 Zeilen. Diese 1000 Zeichen befinden sich natürlich nicht nur auf dem Bildschirm sondern auch im Arbeitsspeicher des Commodore 64. Der Bereich in dem sich die einzelnen Zeichen befinden ist im RAM des Commodore 64 von 1024 bis 2023. Um nun alle Informationen, die auf dem Bildschirm erscheinen auch auf Papier zu bringen, müssen Sie aus diesen Speicherbereich Zeichen für Zeichen mit der PEEK-Funktion auslesen und dann den erhaltenen Wert mit der CHR\$(-Funktion ausdrucken. Dabei müssen Sie dann aber noch beachten, daß Sie Werte im Bereich vom 0 bis einschließlich 31 nicht direkt ausdrucken können, da sich in diesem ASCII-Bereich sogenannte Steuerzeichen befinden, die eine merkwürdige Ausgabe bewirken.

Aber noch etwas anderes gibt es zu beachten. Wie wir eben gesehen haben, besteht eine Zeile auf dem Bildschirm aus 40 Zeichen. Eine Druckzeile besteht aber normalerweise aus 80 Zeichen. Wenn Sie sich mit der Programmierung in BASIC etwas auskennen, so wissen Sie, daß Sie, wenn Sie mehrere Zeichen hintereinander in einer Zeile (Bildschirm oder Drucker) ausgeben wollen, jeden PRINT-Befehl mit einem ";" abschließen müssen. Bei unserer HARDCOPY-Routine müssen Sie also darauf achten, erst einmal 40 Zeichen hintereinander zu drucken (also eine ganze Zeile) um danach ein Carriage Return zu senden mit dem Sie in die nächste Zeile

kommen. Um diesen Vorgang zu programmieren müssen Sie zwei ineinander verschachtelte Schleifen benutzen. Das fertige Programm sieht dann so aus:

```
50000 OPEN 4,4: REM DRUCKERKANAL OEFFNEN FUER GROSS/GRAFIK
50010 FOR I=1024 TO 1984 STEP 40: REM 25 ZEILEN
50015 BZ$="": REM GESAMTZEILE LOESCHEN
50020 FOR J=0 TO 39: REM 40 ZEICHEN
50030 Z=PEEK(I+J): REM ZEICHEN LESEN
50040 IF Z<32 THEN Z=Z+64: REM UMWANDELN VON GROSSBUCHSTABEN
50050 BZ$=BZ$+CHR$(Z): REM GESAMTZEILE BILDEN
50060 NEXT J: REM NAECHSTES ZEICHEN
50070 PRINT BZ$: REM GESAMTZEILE DRUCKEN
50080 NEXT I: REM NAECHSTE ZEILE
50090 RETURN: REM ZURUECK INS HAUPTPROGRAMM
```

Wie Sie sicherlich bemerken, haben wir in diesem Programm einen etwas anderen Weg eingeschlagen als oben angesprochen. Wenn Sie schon einmal eine HARDCOPY-Routine in Aktion gesehen haben, so werden Sie vielleicht festgestellt haben, daß vor dem Druck einer Zeile oftmals eine gewisse Zeit vergeht bevor diese auf dem Drucker ausgegeben wird. Warum dieses ??? Fast jeder Drucker verfügt über einen mehr oder weniger großen sogenannten Puffer. In diesen Puffer werden solange Zeichen aus dem Rechner übertragen, bis der Puffer gefüllt ist. Erst dann wird der Puffer ausgedruckt. Der Vorteil dieser Methode ist, daß auf diese Art und Weise der Druck schneller beendet ist, als wenn Zeichen für Zeichen direkt gedruckt würden. Aus einem ähnlichen Grund füllen wir erst einen TEXT-STRING (BZ\$) bevor wir diesen ausdrucken. Das geht schneller als wenn wir jedes Zeichen einzeln drucken.

Eine Besonderheit dieses Programms ist die Umwandlung von Grossbuchstaben (Zeile 50040). Wie Sie vielleicht wissen, wird bei den Grossbuchstaben im Bildschirmspeicher wieder mit dem Code "1" begonnen. Da aber z.B. "A" nicht den Code 1 sondern 65 hat, so wird zu allen Zahlen < 32 die Zahl 64 addiert.

4.12 DIE BERLINER KURFÜRSTENDAMM-UHR AUF DEM CBM64

Mancher unvorbereitete Bummel auf dem Kurfürstendamm in Berlin mag schon kopfschüttelnd stehengeblieben sein angesichts eines mitten auf dem Grünstreifen stehenden Gestells, an dem rote und gelbe Lampen blinken.

Was sich dort tut, ist auf den ersten Blick einer Lichtorgel nicht unähnlich, und selbst auf den zweiten Blick wird so leicht niemand darauf kommen, daß das ganze eine Normaluhr darstellt, welche die Uhrzeit auf eine etwas ungewöhnliche Weise anzeigt.

Auch der Eingeweihte tut sich schwer, auf Anhieb die richtige Zeit zu bestimmen.

Auf der Seite hinter dem Programmlisting finden Sie eine Zeichnung der Uhr mit der Erklärung der einzelnen Felder.

Wir wollten uns die reizvolle Gelegenheit nicht entgehen lassen, Sie auf diese vergnügliche Art und Weise ein wenig mit den Fähigkeiten des CBM64 vertraut zu machen und haben ein Programm entwickelt, welches eben diese Uhr auf dem Bildschirm Ihres Rechners nachvollzieht.

Natürlich kommt die Anzeige erst auf einem Farbfernseher richtig zur Geltung.

In diesem Programm arbeiten wir, wie könnte es anders sein, mit der Echtzeituhr und mit Sprites, natürlich auch mit der normalen Symbolgrafik.

Gerade die Echtzeituhr hat, was die programmtechnische Handhabung betrifft, so ihre Tücken, was nicht zuletzt auch auf die 12-Stunden-Zählweise der Uhr zurückzuführen ist.

Wir werden daher das Programm Schritt für Schritt zerpfücken, um Sie in die Lage zu versetzen, einige Teile auch für Ihre eigenen Programme anzuwenden.

Zweckmäßig beginnen wir mit der Beschreibung der verwendeten Variablen:

A\$ Uhrzeit bei der Eingabe
AP Pointer auf die Arrays für Symbolgrafik
C Basisadresse der Echtzeituhr
V Basisadresse des Videocontrollers
S1 Adresse des Punktemusters für Sprite 0
S3 Adresse des Punktemusters für Sprite 1
F1 =0 erste Hälfte einer Sekunde
=1 zweite Hälfte einer Sekunde
F2 =1 Uhr neu gestellt
FL =0 vormittags
=1 nachmittags
H Stundenzahl
M Minutenzahl
S Sekundenzahl
T Zehntelsekunden
H\$ Stringformat von H
M\$ Stringformat von M
S\$ Stringformat von S
ME\$(x) Symbolgrafik für Minuteneiner
MZ\$(x) Symbolgrafik für Minutenfünfer
SE\$(x) Symbolgrafik für Stundeneiner
SZ\$(x) Symbolgrafik für Stundenzehner

Im Folgenden finden Sie die Erläuterungen zu den einzelnen Programmschritten:

- 5-10 Die Variablen werden initialisiert.
- 15 Die Rahmenfarbe wird auf schwarz gesetzt und die Sprites ausgeschaltet, was erforderlich ist, wenn Sie das Programm anhalten und neu starten. Die Sprites wären sonst bei der Eingabe der Uhrzeit im Wege.

- 30 Das höchstwertige Bit dieses Registers muß zum Setzen der Uhrzeit =0 sein.
- 35 Zum Zeichen dafür, daß die Echtzeituhr mit einem Takt von 50Hz versorgt wird, muß das höchstwertige Bit dieses Registers =1 sein.
- Der Takt für die Uhr wird nämlich der Netzspannung entnommen, die in den meisten europäischen Ländern eben 50Hz beträgt.
- In den USA hingegen müßte dieses Bit =0 sein, da dort die Netzfrequenz 60Hz beträgt.
- 40-50 Die Uhrzeit wird über Tastatur eingegeben.
- 60-80 Der Eingabestring wird in seine numerischen Bestandteile zerlegt.
- 90-100 Da die Uhr nur einen 12-Stunden-Zyklus hat, muß der AM/PM-Merker (Bit 7 im Stundenregister) entsprechend der Eingabe gesetzt werden.
- AM/PM ist die Abkürzung der lateinischen Ausdrücke 'ante /post meridiem', was zu deutsch schlicht vor-/nachmittags bedeutet. In vielen Ländern wird die Zeit vorzugsweise so dargestellt.
- 110-160 Die einzelnen Werte werden in Einer- und Zehnerteil zerlegt und den korrespondierenden Registern zugeführt. Beachten Sie bitte, daß die Uhrenregister im BCD(Binary Coded Decimal)-Format arbeiten, d.h. jedes Halbbyte darf als Maximalwert nur 9 haben.
- 161-164 Spritedaten für den ausgefüllten Kreis.
- 165-167 Spritedaten für die Kreisumrandung.
- 168-169 Die Spritedaten werden in den Speicher gebracht.
- 170-171 Die Spritepointer werden gesetzt, die Sprites eingeschaltet und auf doppelte Größe gesetzt, die Sprites positioniert und deren Farbe bestimmt.
- 179-196 Die Rahmen für die rechteckigen Felder werden erzeugt.

- 198-224 Hier werden die Arrays mit für jede Minute und Stunde einmaligen Werten und Farben gefüllt. Dieses Verfahren ist zwar speicheraufwendiger als ein berechnetes Füllen der Felder, dafür aber wesentlich schneller.
- 225-229 In einem Intervall von einer halben Sekunde wird hier das Sprite für den ausgefüllten Kreis abwechselnd ein- und ausgeschaltet.
- 230-290 Die Echtzeituhr wird ausgelesen und die Werte in eine Dezimalzahl überführt. Zu der gelesenen Stundenzahl wird abhängig vom AM/PM-Flag (die Variable FL wird daraus in Zeile 250 abgeleitet) 12 addiert, um zu einer 24-Stunden-Anzeige zu kommen.
- 310-350 Die aus der Uhr gelesenen Werte werden in ein druckbares Format umgewandelt und in der unteren rechten Ecke des Bildschirms angezeigt.
- 380-410 Hier werden nun die in den Arrays hinterlegten Minuten- und Stundenmuster erzeugt. Da ja für jede Kombination ein Muster existiert, brauchen die Arrays praktisch nur noch mit der Uhrzeit indiziert werden.
- Dieser Vorgang ist natürlich nur jede volle Minute erforderlich (Zeile 370), auf jeden Fall jedoch nach dem Stellen der Uhr (Zeile 360).

Nachfolgend nun das Programmlisting.

Bitte, haben Sie Verständnis für die etwas geringere Schriftqualität, da wir die Liste wegen der vielen Sonderzeichen natürlich auf einem Drucker herausbringen mußten, der diese reproduzieren kann.

4.13 EIN KLEINER KOPIERSCHUTZ

Jeder, der einmal ein Programm geschrieben hat weiß, was er da geleistet hat. Stunden, ja Tage, Wochen oder Monate voller Mühe stecken in einem guten Programm. Wer will sich da nicht einigermaßen gegen eine unerlaubte Vervielfältigung seiner Arbeit und seiner Idee absichern ? Das ist allerdings leichter gesagt als getan, denn auf der ganzen Welt gibt es eine Gemeinde von Computerfreaks, die es sich zur Aufgabe gemacht hat, jeden neuen Kopierschutz und sei er noch so perfekt, zu knacken. Das ist mancherorts zu einem richtigen Volkssport geworden.

Ferner gibt es heute immer ausgefeiltere Kopierprogramme, mit denen es auch dem blutigen Laien möglich ist, geschützte Programme zu kopieren, da die vorhandene Sicherung des Programms einfach von dem Kopierprogramm mit übertragen wird. Grundsätzlich muß man zwischen zwei unterschiedlichen Schutzarten unterscheiden. Da gibt es einerseits den sogenannten Hardwareschutz. Das ist beispielsweise ein elektronisches Bauteil, welches in den Userport oder auch den Joystickport gesteckt wird. In diesem "elektronischen Schlüssel" befindet sich dann ein integrierter Schaltkreis, der die in dem Programm zu verarbeitenden Daten in einer ganz bestimmten Form umsetzt oder logisch verschiebt. Da das Innenleben eines solchen "Dongls" nicht erkennbar ist, er ist mit einer Kunststoffmasse vergossen, bedarf es schon eines fundierten Wissens auf dem Gebiet der Elektronik und manchen guten Meßgeräten, um der Art und Weise wie die Daten verschlüsselt werden, auf die Schliche zu kommen. Für einen Privatmann ist diese Methode jedoch gänzlich ungeeignet, da er nicht über die nötigen Geräte zur Herstellung solcher Hardwaresicherungen verfügt. Aber trösten Sie sich, denn abgesehen von den hohen Herstellungskosten hat auch ein Hardwareschutz den Nachteil, daß er früher oder später überlistet wird.

Wir wollen uns auf den folgenden Seiten mit der Softwarelösung dieses Problems beschäftigen. Es bleibt hier sicherlich noch anzumerken, daß die folgenden Zeilen bestenfalls eine Einführung in diese Thematik darstellen. Wenn Sie sich für dieses Gebiet interessieren, so sei an dieser auf das Buch "64 TIPS & TRICKS, Band 2" verwiesen.

Unser Kopierschutz besteht aus zwei Teilen. Der erste Teil ist die Vorbereitung, der zweite stellt dann, in modular aufgebauter Form, den eigentlichen Schutz dar. Dieses Modul läßt sich problemlos in neue, aber auch in schon geschriebene Programme implementieren. Ein sehr großer Vorteil unserer Lösung ist dadurch gegeben, daß es derzeit nur zwei Kopierprogramme auf dem einschlägigen Markt gibt, die in der Lage sind, unsere Hürde zu überwinden. Damit sind Sie zumindest gegen manchen "Gelegenheitstäter" einigermaßen abgesichert.

Bei allen Versuchen mit der nachfolgenden Lösung sollten Sie sich aber unbedingt an zwei Richtlinien halten :
Bevor Sie die Funktion der beiden Programme austesten, sollten Sie diese unbedingt vorher abspeichern, da die Gefahr relativ groß ist, daß etwas schief läuft, was zur Folge hat, daß alle Erfassungsarbeit umsonst war. Verwenden Sie bei allen Tests und eventuellem späteren Praxiseinsatz grundsätzlich nur leere Disketten, da das Vorbereitungsprogramm einen Diskettenblock verändert, ohne darauf Rücksicht zu nehmen, ob dieser vielleicht schon von normalen Daten belegt ist.

Hier nun die Vorbereitungsroutine :

```

10 REM *****
20 REM SCHEINBLOCK VORBEREITEN
30 REM *****
40 PRINT CHR$(147)
50 PRINT"    WOLLEN SIE EINEN BESTIMMTEN
BLOCK"
60 PRINT"    VORBEREITEN
(J/N)"
70 GET A$ :IF A$ <> "J" AND A$ <> "N" TH
EN GOTO 70
80 IF A$ = "N" THEN TT = 18 : SS = 2 : G
OTO 200
90 PRINT
100 INPUT"    TRACK    ";TT
110 IF TT > 35 THEN PRINT"EXISTIERT NICH
T"; CHR$(145); CHR$(145) : GOTO 100
120 PRINT"    "
130 SX = 16
140 IF TT < 18 THEN SX = 20
150 IF TT < 25 THEN SX = 18
160 IF TT < 31 THEN SX = 17
170 INPUT"    SEKTOR    ";SS
180 IF SS > SX THEN PRINT"EXISTIERT NICH
T"; CHR$(145); CHR$(145) : GOTO 170
190 PRINT"    "
200 OPEN 1,8,15 : OPEN 2,8,2,"#"
210 PRINT#1,"B-P:"2;0
220 INPUT"    CODEWORT ";C$
230 IF LEN(C$) > 15 THEN PRINT"ZU LANG";
CHR$(145); CHR$(145) : GOTO 220
240 PRINT#2,C$
250 PRINT#1,"U2:"2;0;TT;SS
260 CLOSE2
270 PRINT#1,"B-F:"0;TT;SS
280 CLOSE1
290 PRINT"    FERTIG !" : END

```

Was wird durch diese kleine Programm erreicht?

Sie können sich entscheiden, ob Sie einen von Ihnen bestimmten Block auf der Diskette vorbereiten wollen oder nicht. Wenn Sie sich für "N" entscheiden, so notieren Sie sich bitte die Werte : Spur 18, Sektor 2. Wenn Sie einen anderen Block vorbereiten, so müssen Sie sich die von Ihnen vergebenen Werte merken. Das Programm prüft automatisch ab, ob Ihre Eingaben überhaupt möglich ist und weist Sie dann auf eventuelle Fehler hin. Das geschieht alles in den Zeilen 10 bis 190. In Zeile 200 wird dann der Kommandokanal und ein Datenkanal zur Floppy eröffnet. Anschließend werden Sie aufgefordert, ein "Codewort" einzugeben. Auch dieses müssen Sie sich unbedingt merken. In der hier vorgestellten Programmversion darf das Codewort nicht länger als 15 Zeichen sein.

Das Codewort wird dann in den Datenbuffer 2 geschrieben, welcher seinerseits danach als kompletter Block an die vorgegebene Stelle auf der Diskette geschrieben wird. Abschließend wird der eben aufgebrauchte Block in der BAM der Diskette als "frei" gekennzeichnet. Das ist einer der Gründe, warum diese Schutzfunktion für viele Kopierprogramme unüberwindbar ist.

Einen Haken hat die Sache allerdings. Wenn Sie die vom Programm vorgegebene Adresse verwenden, so laufen Sie Gefahr diese wieder zu löschen, wenn Sie mehr als 24 Programme auf Ihrer vorbereiteten Diskette abspeichern. Haben Sie selbst die Adresse bestimmt, dann läßt es sich nicht vorhersagen, ob und wann Ihr Block wieder zu einem normalen Datenblock wird.

Kommen wir jetzt zum zweiten Teil unseres Sicherungspakets, der sogenannten Blockabfrage.

```

10 REM *****
20 REM SCHEINBLOCK ABFRAGEN
30 REM *****
40 PRINT"DAS IST EINE DEMONSTRATION"
50 PRINT"FUER DIESEN KOPIERSCHUTZ"
60 INPUT"CODEWORT ";C$
70 TT = 18 : SS = 2
80 GOSUB 60000
90 END

60000 OPEN1,8,15 : OPEN2,8,2,"#"
60010 PRINT#1,"U1:"2;0;TT;SS
60020 INPUT#2,A$
60030 CLOSE1 : CLOSE2
60040 IF A$ = C$ THEN RETURN
60050 SYS 64738

```

Dieses kleine Demonstrationsprogramm muß auf der vorher vorbereiteten Diskette abgespeichert werden, da es sonst nicht funktionstüchtig ist. Die eigentliche Sicherungsfunktion finden Sie in den Unterprogrammzeilen 60000 bis 60050. Somit stehen Ihnen alle Basic-Zeilen unter 60000 für Ihr eigenes Programm zur Verfügung. Zu beachten ist dabei nur, daß Sie Ihr Codewort in der Variablen C\$ ablegen, bevor Sie das Unterprogramm anspringen. Ebenso sollten Sie bei der Kontrolle, ob das richtige Codewort auf der Diskette ist (in Zeile 60040), dort den direkten Code eintragen.

```

60040 IF A$ = "Ein Codewort" THEN RETURN

```

Ferner gilt es zu beachten, daß die Adresse auf der Diskette entweder in den Variablen TT (für Spur) und SS (für Sektor) abgelegt ist oder daß auch sie direkt in dem Block-Lese Befehl der Zeile 60010 eingetragen wird.

```

60010 PRINT#1,"U1:"2;0;18;2

```

In der Zeile 60050 des Unterprogramms können Sie dann wahlfrei entscheiden, was geschehen soll, wenn bei der Kontrolle keine Übereinstimmung erzielt wurde. In unserem Fall wird ein Reset des Rechners ausgelöst. Das heißt, der Rechner wird wieder in seinen Einschaltzustand versetzt. Das Programm im Rechner ist dann selbstverständlich verloren. Sie sehen, daß sich Ihnen mit dieser Methode ungeahnte Möglichkeiten zur Sicherung Ihres geistigen Eigentums bieten.

Selbstverständlich ist es auch denkbar, daß Sie mehrere Blöcke auf Ihrer Diskette in der beschriebenen Art und Weise mit einem Schlüssel vorbereiten.

Um zu sehen, daß unser Schutz auch wirklich funktioniert, sollten Sie einmal hergehen und das von Ihnen vergebene Codewort falsch eintragen.

An dieser Stelle muß aber noch darauf hingewiesen werden, daß es relativ einfach ist, die ganze Schlüsselabfrage aus dem Programm zu entfernen. Das läßt sich vermeiden, indem Sie Ihr Programm compilieren oder einen guten Listschutz für ein Basicprogramm verwenden.

KAPITEL 5: DER 64 KANN NICHT NUR BASIC

5.1 DIE PROGRAMMIERUNG VON FORTH

WAS IST FORTH ? Diese Frage wird wohl jeder stellen, der sich bisher nur mit der Programmierung in BASIC oder Assembler beschäftigt hat. Sicherlich wird so mancher sagen "Warum soll ich noch eine andere Programmiersprache anwenden, wo mein Commodore 64 doch hervorragend BASIC spricht ?". Dieser Einwand mag auf den ersten Blick gerechtfertigt sein, auf den zweiten Blick aber sollte man aber mal überlegen, ob es nicht vielleicht doch ganz gut ist, wenn ein Computer eben nicht nur eine Sprache sprechen kann, sondern ein wahres Sprachengenie ist.

Wenn Sie schon eine Zeit in BASIC programmiert haben, dann werden Sie wissen, daß es hin und wieder Sachen gibt, die nicht oder nur recht schwierig zu programmieren sind. Für solche Fälle hat man bei großen Rechnern die Möglichkeit auf andere Sprachen zurückgreifen zu können. Es gibt FORTRAN für mathematische Problemlösungen, COBOL für kommerzielle Zwecke, ASSEMBLER für zeitkritische Aufgaben (also solche Probleme die besonderes schnell bearbeitet werden müssen, z.B. Maschinensteuerungen), BASIC für die allgemeine Problemlösung auf Microcomputern und, und, und. Dann gibt es noch Sprachen, die dem Neuling das strukturierte Programmieren beibringen sollen, wie PASCAL und ELAN. Alle Sprachen haben also ihre Stärken und ihre Schwächen.

FORTH, das heißt soviel wie 'Die Vierte', gehört zu der jüngsten Generation von Programmiersprachen, wie ihr Name schon sagt, zur vierten Generation. Man hat also versucht alle Vorteile von den bekannten, renomierten Programmiersprachen in FORTH zu implementieren, ohne aber deren Fehler mit zu übernehmen.

FORTH hat, schon in seiner Struktur, einige sehr markante Vorteile, die besonders bei den Microcomputern von Vorteil sind:

1.) Der Computer, auf dem FORTH läuft, benötigt in der Regel nur einen sehr kleinen Arbeitsspeicher. Da FORTH Programme nicht viel Platz benötigen, kann man auch auf einem sehr kleinen Rechner, große, leistungsfähige Programme erstellen.

2.) FORTH benötigt keine, oder nur sehr wenige, E/A-Leitungen. Man muß also nicht noch die Hardware seines Gerätes bis ins Innere kennen, um in FORTH zu programmieren. Außerdem stehen bei den Microcomputern sowieso oft nur sehr wenige E/A-Leitungen zur Verfügung.

3.) Aus dem unter 2 schon genannten Vorteil, geht auch hervor, daß FORTH selbst ohne eine Floppy-Station auskommen würde. Vorteilhaft ist eine Floppy natürlich in jedem Fall.

FORTH besteht aus fünf Teilen:

1.) INHALTSVERZEICHNIS: Die Philosophie von FORTH ist, daß der eigentliche Wortschatz, also die Befehle, die FORTH versteht, nur sehr klein ist. Es besteht aber die Möglichkeit, eigene Befehle zu definieren und die in den Programmen selbst anzuwenden. Sie können sich so Ihre eigene, sehr leistungsfähige Sprache schreiben - gerade so, wie Sie sie benötigen. Als Beispiel:

Sie wollen ein Programm schreiben, das mathematische Probleme lösen soll - zum Beispiel eine Kurvendiskussion. In diesem Programm werden Sie also kaum Teile zur Textbearbeitung benötigen, und wenn Sie auch auf Grafik verzichten wollen, so lassen Sie auch diese Befehle einfach 'draußen'. Aus diesem Grund benötigen Sie also auch keinen großen Arbeitsspeicher.

2.) STACK: Eigentlich das wichtigste Element bei FORTH. Der Begriff STACK wird schon allen denen bekannt sein, die sich ein wenig mit der Programmierung in Maschinensprache auskennen, oder die einen HP-Taschenrechner besitzen. Darauf kommen wir später noch zurück.

Dieser STACK arbeitet nach dem LIFO-System (Last in - First out, oder übersetzt: Als letzter rein, als erster raus). Alle Operationen, rechnerischer-, logischer-, vergleichender Art, werden auf diesem STACK bearbeitet.

3.) INTERPRETER: Wie schon vom Commodore 64 bekannt, so arbeitet auch FORTH mit einem Interpreter. Dieses bedeutet, daß Sie zuerst das FORTH Programm mit dem EDITOR erstellen, und dann mit dem entsprechenden Befehl das Programm starten. Während des Programmlaufs wird dann die Fehlerprüfung betrieben. Man muß also nicht erst das Programm auf langwierige Art und Weise compilieren, sondern kann es direkt starten. Das hat allerdings meistens zur Folge, daß interpretierte Programme langsamer sind. Bei FORTH ist dieser Zeitfaktor nicht so groß - der Interpreter unterteilt sich in den TEXT-INTERPRETER und den ADRESS-INTERPRETER. Der TEXT-INTERPRETER prüft die WÖRTER in dem Inhaltsverzeichnis, und wenn das WORT gefunden wurde, wird der ADRESS-INTERPRETER aktiviert. Dieser Interpreter arbeitet dann die absoluten Adressen ab. Die Folge ist, daß auf diese Art einen sehr schnellen Ablauf gewährleistet ist.

4.) ASSEMBLER: Jeder FORTH-Interpreter beinhaltet einen Assembler, Mit diesem Assembler können Sie WÖRTER definieren, die dann bei ihrem Aufruf ganze Maschinenroutinen ausführen können. Diese Art der Programmierung wird benötigt, um eine Verbindung zur Aussenwelt herzustellen. Wie Sie sicherlich bemerken, hat FORTH hier also auch eine große Ähnlichkeit mit der üblichen ASSEMBLER-Programmierung - sie ist genau so schnell, aber viel einfacher zu erlernen.

5.) SPEICHER: Sehr wichtig für FORTH ist natürlich der Arbeitsspeicher. Dieser Speicher hat die Besonderheit, daß bei Veränderung der einzelnen Blöcke im Speicher, die immer gleich lang sind, gleichzeitig auch der entsprechende Block auf der Diskette verändert wird, natürlich nur dann, wenn auch eine Floppy angeschaltet ist.

5.2 VERGLEICHSPROGRAMM FORTH - BASIC

Natürlich lernen Sie FORTH am leichtesten kennen, wenn Sie ein FORTH Programm im Vergleich zu einem BASIC Programm sehen, das die selbe Aufgabe erfüllt.

Bevor wir aber zum Programm kommen, müssen wir erst ein paar Begriffe und Techniken erklären.

Wir haben in Kapitel 5.1 schon darauf hingewiesen, daß der STACK eine wichtige Rolle spielt, und man hier Vergleiche mit der Arbeitsweise der HP-Taschenrechner ziehen kann:

Wir rechnen nun $(2 + 3) * (4 + 5)$ auf einem HP-Rechner. Die Tastenfolge lauten so:

2 <ENTER> 3 + 4 <ENTER> 5 + *

Das sieht auf den ersten Blick vielleicht verwirrend aus, aber wenn man mal versucht ohne die üblichen '(' ')' zu arbeiten, wird man sehen, daß dieses die einzige Möglichkeit ist, die Aufgabe zu lösen. Ich nehme also die Zahl '2', durch die Taste <ENTER>, wird sie auf den STACK gebracht. Dann drücke ich die Zahl '3', auch diese Zahl muß auf den STACK gebracht werden, also wir die '2' um einen Platz weiter nach hinter geschoben. Auf einem Bild könnte man sich den Vorgang so vorstellen:

STACK:	OBEN	-	↑
		-	↑
		-	↑
		-	↑
		-	↑
		2	↑
	UNTEN	3	↑

(Die Pfeile auf der rechten Seite sollen die Richtung anzeigen, in die der Stack jeweils erweitert wird.)

Danach wird die Taste '+' gedrückt, und die Addition durchgeführt.
Nach der Addition sieht der STACK so aus:

OBEN	-
	-
	-
	-
	-
	-
UNTEN	5

Beim Drücken der Taste '4' verschiebt sich nun die '5' einen Platz nach oben, und die '4' steht darunter:

OBEN	-
	-
	-
	-
	-
	5
UNTEN	4

Nun soll die Zahl 5 eingegeben werden. Nach dem Drücken dieser Taste, verschieben sich sowohl die alte '5' als auch die '4' weiter eine Stelle nach oben:

OBEN	-
	-
	-
	-
	5
	4
UNTEN	5

Jetzt sollen die Zahl 4 und die Zahl 5 addiert werden. Also sieht der STACK danach so aus:

OBEN	-
	-
	-
	-
	-
	5
UNTEN	9

Und als letzte Operation nun noch die Multiplikation:

OBEN	-
	-
	-
	-
	-
	-
UNTEN	45

Nun steht uns also das Ergebnis zur Verfügung. Das alles sieht zwar unheimlich kompliziert und zeitaufwendig aus, aber jeder Taschenrechner und sogar Computer arbeitet nach diesem Prinzip. Beim HP-Taschenrechner ist dieses Prinzip allerdings am deutlichsten ausgebildet. Jeder der schon mal mit solch einem Rechner gearbeitet hat, bzw. in Assembler programmiert hat, wird sich schnell in FORTH hineindenken können - aber auch der Neuling in der Programmierung wird weniger Schwierigkeiten haben, als er vielleicht denkt, denn FORTH ist tatsächlich eine sehr leicht zu erlernende Sprache. Die einzige Schwierigkeit von FORTH ist nur, daß es so ganz anders als die anderen Sprachen ist.

Hier nun ein erstes kleines FORTH-Programm, daß die Wirkungsweise des Stacks erläutert und gleichzeitig die Definition eines WORTES zeigt.

In diesem Programm geht es darum, den Kubikwert einer Zahl zu ermitteln. Da für diese Rechnung kein Befehl existiert, müssen wir ihn erst einmal definieren:

```

: KUBIK      ( DIESES WORT WIRD NUN DEFINIERT )
  DUP DUP    ( KOPIERT DIE ZAHL 2* IN DEN STACK )
  * *        ( MULIPLIZIERT DEN STACK 2* MIT SICH )
;

```

Was bedeutet nun dieses Programm ? Zunächst wird durch den ':' FORTH mitgeteilt, daß nun ein Wort definiert werden soll. Würden Sie den Doppelpunkt nicht mit angeben, so würde FORTH mit der Meldung

KUBIK ?

zurückkommen, und damit anzeigen, daß dieses Wort noch nicht im Inhaltsverzeichnis vermerkt ist. Wir wollen aber gerade ein Wort definieren, also ins Inhaltsverzeichnis übernehmen. Dazu benötigen wir also den ':'. FORTH weiß nun, daß solange die Definition eines Wortes folgen soll, bis es auf ein Semikolon ';' trifft. Nach diesem Zeichen meldet sich FORTH wieder mit seinem sympatischen

OK

Wir wissen nun, daß FORTH das Wort in sein Inhaltsverzeichnis aufgenommen hat, und können es von nun an direkt, oder in einem Programm anwenden. Übrigens wird die Definition des Wortes nur solange im Computer behalten, wie dieser angeschaltet bleibt. Um diese neuen Worte auch später wieder einsetzen zu können, müssen sie natürlich auf Floppy abgespeichert werden. So kann man später immer wieder das erweiterte FORTH von Diskette nachladen.

Der Befehl DUP kopiert den untersten STACK-Wert. Da wir aber nicht nur das Quadrat, sondern das Kubik einer Zahl berechnen wollen, so müssen wir den Stack zweimal kopieren, also ein zweites DUP. Bei der Zahl 5 sähe der STACK also nun so aus:

```

OBEN  -
      -

```

UNTEN 5

Nun müssen wir die Zahlen noch miteinander multiplizieren. Insgesamt sind also 2 Multiplikationen notwendig. FORTH verwendet hierfür das normale Multiplikationszeichen '*'. Nachdem wir so den STACK zweimal multipliziert haben, können wir die Definition beenden. Dazu geben wir, wie oben schon erwähnt, ein Semikolon ein. FORTH meldet sich wieder mit seinem OK (wenn wir alles richtig gemacht haben) und wir können nun mit den Berechnungen beginnen. Also zum Beispiel mit der 5:

SIE GEBEN EIN:

FORTH MELDET:

5 KUBIK .

125 OK

1 KUBIK .

1 OK

-15 KUBIK .

-3375 OK

Wie Sie sehen, ist es sehr leicht neue Befehle selbst zu definieren. Gerade das ist ja der enorme Vorteil von FORTH. Sie brauchen nie einen unnützen Ballast an Befehlen, die Sie überhaupt nicht benötigen mit sich herum schleppen und sparen damit eine ganze Menge an Speicherplatz.

Hier nun ein Vergleich zwischen einem FORTH und einem BASIC Programm:

Dieses Programm berechnet die Kubikzahlen zwischen 0 und 10. Wenn Sie dieses Programm in FORTH schreiben, müssen Sie daran denken, daß in diesem Programm das Wort KUBIK verwendet wird. Es muß also vorher mit dem oben angegebenen Programm erst definiert werden.

KUBIKZAHLEN:

1.) FORTH

: KUBIKZAHLEN

```
10 0      ( VON 0 BIS 10 ! LIFO !! )
DO        ( SCHLEIFENBEGINN )
    CR I . I KUBIK . ( DRUCKE ZAHL (I) UND KUBIK )
LOOP      ( SCHLEIFENENDE )
;
```

KUBIKZAHLEN

```
0 0
1 1
2 8
3 27
4 64
5 125
6 216
7 343
8 512
9 729 OK
```

2.) BASIC

```
10 REM BERECHNEN VON KUBIKZAHLEN
20 MIN=0: MAX=9
30 FOR I=MIN TO MAX: PRINT I,I*I*I
40 NEXT I
50 END
```

RUN

```
0 0
1 1
2 8
3 27
4 64
5 125
6 216
7 343
8 512
```

READY.

Wie Sie sehen, benötigen beide Programme ungefähr die gleiche Anzahl an Programmzeilen. Aber im Speicher sieht die ganze Sache schon anders aus - das FORTH Programm ist um einiges kleiner. Aber diese Platzersparnis ist bei weitem noch nicht alles. Machen wir einmal einen Zeitvergleich (Benchmarktest) in FORTH und BASIC. Was bietet sich dabei anderes an als eine einfache Schleife:

1.) FORTH

```
: ZEITTEST
    30000 0      ( VON 0 BIS 30000 )
    DO          ( SCHLEIFENBEGINN )
                ( LEERE SCHLEIFE )
    LOOP        ( SCHLEIFENENDE )
    ;           ( ENDE )
```

ZEITTEST OK

2.) BASIC

```
10 REM ZEITTEST
20 MIN=0: MAX=30000
30 FOR I=MIN TO MAX
40 NEXT I
50 END
```

RUN

READY.

Und nun kann unser kleines Wettrennen beginnen. Das Ergebnis ist vielleicht recht überraschend: Es war ein Rennen zwischen Hase und Igel, nur daß der Igel diesmal nicht ganz so intelligent war. Hier die Zeiten:

SPRACHE:	ZEIT:
-----	-----
BASIC	ca. 40 sek.
FORTH	ca. 4 sek.

Und das sollte hierbei nicht vergessen werden: Beide Zeittests wurden mit einem und demselben Rechner gemacht:

IHREM COMMODORE 64 !!!

Alleine dieses Ergebnis dürfte einige Leute dazu bringen, vielleicht doch einmal zu überlegen, FORTH kennenzulernen. Die Programmierung ist denkbar einfach, der Zeitgewinn gravierend und etwas neues zu lernen macht immer Spaß. Oder ?

Wenn Sie sich nun für das Commodore FORTH interessieren, an dieser Stelle noch einige spezielle Informationen:

FORTH 64 kann durch einige kleine Wortänderungen, zu einem FORTH-79 Standard Paket gemacht werden - jenem Standard auf dem die meisten FORTH-Programme geschrieben wurden.

FORTH 64 benutzt 3 Disk-Puffer, die jeweils 1024 Bytes lang sind, und bei der Adresse \$C000 liegen. Außerdem wird von FORTH 64 noch ein Bildschirmspeicher generiert, der bei \$0400 liegt.

FORTH 64 ist auf MODUL gespeichert und so immer griff- und startbereit.

FORTH - Die Sprache für den professionellen Software-Entwickler

Interessant dürfte auch sein, daß immer mehr professionelle Software-Entwickler auf FORTH umschwenken. Der Grund liegt in den oben skizzierten Vorteilen von FORTH. Es ist bereits nach kurzer Einarbeitungszeit ähnlich leicht und komfortabel zu benutzen und

bietet trotzdem die Flexibilität und Rechengeschwindigkeit die an Maschinenprogrammierung so geschätzt wird. Auch auf dem Commodore 64 sind bereits anspruchsvolle Programm-Pakete in FORTH entwickelt worden. Dazu gehört zum Beispiel das Programm CALC RESULT, ein sehr leistungsfähiges, universelles Kalkulations- und Planungsprogramm.

Noch einen weiteren Vorteil, den besonders Softwarehäuser schätzen, hat FORTH. Es gehört zu den wenigen portablen Sprachen, d.h. in FORTH auf einem Rechner geschriebene Programme, lassen sich sehr leicht auch auf einem anderen Rechner implementieren. Dadurch werden Softwareinvestitionen nicht so schnell wertlos, und es dauert nicht mehr so lange, bis zu neuen Computern entsprechende Programme angeboten werden können.

Zur Arbeit mit der Sprache FORTH kann man jedem Commodore 64 Programmierer uneingeschränkt raten.

5.3 WEITERE SPRACHEN: PASCAL, LOGO, TURTLE GRAPHICS

Durch verschiedene Sprachen wird ein Computer erst zu einem perfekten Computer. Auch für den Commodore 64 wird es sehr bald die verschiedensten Sprachen geben. Diese Möglichkeit wird dadurch noch erweitert, daß der Commodore 64 ja nun zu den Computern gehört, die das Betriebssystem CP/M verarbeiten können (s. Kapitel 6). Und unter CP/M sind fast alle Sprachen lauffähig, die es überhaupt für Microcomputer gibt.

Wozu dienen die einzelnen Sprachen ?

PASCAL

Diese Sprache ist eigentlich als Lehrsprache gedacht. Sie wurde von einem schweizer Professor (N. WIRTH) entwickelt, und besticht durch die Möglichkeit richtig strukturiert zu programmieren. PASCAL wird heute bei sehr vielen Instituten und Unternehmen als reguläre Computersprache eingesetzt und an den verschiedenen Schulen und Hochschulen gelehrt.

LOGO

Diese Sprache ist noch recht unbekannt. Auch Sie wurde für die Ausbildung am Computer entwickelt. SIE IST SO LEICHT ZU ERLERNEN, DASS SOGAR KINDER SCHON NACH EIN PAAR STUNDEN PROGRAMMIEREN KÖNNEN. Dies ist der Wahlspruch dieser Sprache, und er stimmt auch. Die einzelnen Elemente dieser Sprache sind leicht zu erlernen und anzuwenden.

TURTLE GRAPHICS

Dies ist eigentlich keine der bekannten Sprachen. Sie steht zwischen PILOT und LOGO. Wie ihr Name schon sagt, unterstützt sie die Graphikprogrammierung. Da der Commodore 64 ja bekanntlich über hervorragende Graphikeigenschaften verfügt, ist diese Sprache wie geschaffen für diesen Rechner. Hier ein kleines

Beispiel, welches die Programmierung einer Graphik unter TURTLE GRAPHICS verdeutlichen soll:

```
1 LABEL KASTEN
2 REMARK SETZEN DER BILDSCHIRM FARBEN
3 SCREEN COLOR CYAN
4 BORDER COLOR CYAN
5 TURTLE COLOR BLACK
6 REMARK SETZEN DES HI-RES MODUS
7 HIRES
8 REMARK STUMMSCHALTUNG DES SCHREIBERS
9 PEN UP
10 REMARK BILDSCHIRMPPOSITION OBEN-LINKS
11 MOVE TO 0-0
12 REMARK EINSCHALTEN DES SCHREIBERS
13 PEN DOWN
14 REMARK BILDSCHIRMPPOSITION UNTEN-RECHTS
15 MOVE TO 199-319
16 REMARKS WARTESCHLEIFE
17 WAIT 300
```

Mit diesem Programm zeichnen Sie eine Gerade von der Position 0,0 zur Position 199,319. Dabei verhält sich der Schreiber so, wie es ein Plotter machen würde. Sobald PEN DOWN gesetzt ist, zeichnet er jede Bewegung auf - z.B. die Bewegung durch den MOVE TO Befehl.

Wie Sie sehen, gibt es sehr viele interessante Sprachen, die aus dem Commodore 64, in jeder Hinsicht, einen Computer machen, der es auch mit größeren Systemen durchaus aufnehmen kann.

5.4 ADA FÜR DEN COMMODORE 64

Vor einiger Zeit ist zu den Sprachen, die Ihr Commodore versteht, eine neue hinzugekommen. Es ist ADA. Der Name dieser Sprache ist im Gegensatz zu vielen anderen keine Abkürzung der Sprachbedeutung (z.B. COBOL = Common Business Oriented Language). ADA ist der Vorname einer Frau, die sich schon im 18. Jahrhundert mit der Entwicklung von Computersprachen beschäftigte.

ADA, wie viele andere Sprachen vom amerikanischen Verteidigungsministerium gefördert, ist heute so aktuell wie nie zuvor. Sie zeichnet sich besonders durch eine gute Strukturierung und leichte Erlernbarkeit aus. ADA-Befehle sind, wie allgemein üblich, der englischen Sprache entlehnt. Ein ADA Programm ist immer modular aufgebaut. Jedes Programm besteht aus einer Reihe von Problemen. Bei ADA wird jedes Problem wiederum in einzelne Fragmente zerlegt, die in ihrer Gesamtheit dann die Problemlösung ergeben. Variablennamen können entsprechend ihrer Bedeutung frei gewählt werden. Die ganze Programmstruktur ist sehr übersichtlich und deshalb auch bei einer Fehlersuche leicht zu erfassen. Es gibt keine überflüssigen Formalanweisungen, die speziell einem Einsteiger leicht den Spaß an der Sache verübeln können.

Betrachten wir einmal ein kleines ADA Programm und Sie werden feststellen, daß es fast selbstdokumentierend ist.

```
010 with TEXT IO; use TEXT IO;
020 with C 64; use C 64;
030 --
040 -- Dieses Programm berechnet das
050 -- Produkt aus zwei einzugebenden
060 -- Werten und druckt es aus.
070 --
080 procedure BERECHNEN is
090 --
```

```

100 -- Variablen festlegen
110 --
120 MAND : float :=0
130 MOR  : float :=0
140 --
150 RESULT : float :=0
160 --
170 begin
180 --
190 --
200 put ("Geben Sie den Multiplikanden ein");
210 get ( MAND )
220 --
230 put ("Geben Sie den Multiplikator ein");
240 get ( MOR )
250 --
260 -- Ergebnis bilden
270 --
280 RESULT := MAND * MOR ;
290 --
300 --
310 set row(10); set col(10);
320 --
330 -- Ergebnis anzeigen
340 --
350 put ( MAND ); put ("*"); put ( MOR );
360 put ("="); put ( RESULT );
370 --
380 --
390 end BERECHNEN

```

Zu diesem Programm bedarf es eigentlich keinerlei Erklärung mehr.

Selbstverständlich ist auch die Programmverschachtelung durch mehrere komfortable Befehle möglich. Da Sie unterschiedliche Verschachtelungsbefehle nutzen können, wird ein ADA-Programm damit richtig elegant.

KAPITEL 6: CP/M AUF DEM COMMODORE 64

6.1 DAS IST CP/M

CP/M ist das am meisten verbreitete Betriebssystem, das auf Microcomputern eingesetzt wird. Es hat sich bis heute als das Standard-Betriebssystem bewährt, das am ausgereiftesten seiner Art überhaupt ist. Bei vielen neuen Betriebssystemen hat man oft das Problem, daß lange Jahre nach der Einführung des Betriebssystems noch immer nicht alle Kinderkrankheiten behoben sind. Bei CP/M gibt es dieses Problem nicht. Sie können sich also voll und ganz auf dieses Betriebssystem verlassen.

Nur, was hat nun der Commodore 64 Anwender von diesem Betriebssystem ? Er ist an sein DOS und an sein BASIC 2 gewöhnt - warum dann ein anderes Betriebssystem ?

Diese Frage stellt sich eigentlich nur dem Commodore 64 Benutzer, der noch nichts von der großen Computerwelt gesehen hat. Dort findet er nämlich einen Berg an Software, von dem er bisher nur geträumt hat. Nicht, daß der Commodore 64 über wenig Software verfügt, im Gegenteil, aber gegenüber dem ganzen Berg von CP/M Software, macht sich die Commodore 64 Software wie eine kleine Bodenerhebung aus. Es wird zwar in der nächsten Zeit eine ganze Menge an Commodore 64 auf den Markt kommen - von Textverarbeitung, über Dateiverwaltung, kommerzielle Software, bis hin zu Spielen. Aber die CP/M Software hat schon einen Vorsprung der nicht mehr einzuholen ist.

Das breite Angebot von CP/M Software umfasst nicht nur diverse Programme für Textverarbeitung und Dateiverwaltung, sondern auch individuelle Software läßt sich auf dem CP/M Markt in Hülle und Fülle finden.

Aber nicht nur der Anwender solcher Software profitiert von diesem Angebot, auch der Programmierer hat viele Vorteile von

diesem CP/M System. So kann er seine Software nun für einen wesentlich größeren Anwenderkreis schreiben, als er es bisher machen konnte. CP/M Software zu erstellen, ist ein Begriff der 100.000 Computerbenutzern etwas sagt; der Programmierer von Commodore 64 Software findet seinen Absatzmarkt nur bei dem Commodore 64 Anwenderkreis. Wir wollen damit nicht sagen, daß dieser Kreis nicht sehr groß wäre - aber CP/M ist halt zu vergleichen mit einem Gütesiegel. So mancher Programmierer auf einem 'exotischen' Betriebssystem, hat sicherlich schon gehört:

UND WANN GIBT ES DAS PROGRAMM AUF CP/M BETRIEBSSYSTEM ?

An dieser Stelle sollten wir aber auch ein Problem bei jedem CP/M Betriebssystem hinweisen:

CP/M IST NICHT GLEICH CP/M !

Leider verwenden die meisten Computerhersteller ein eigenes, modifiziertes CP/M. Dadurch ist es trotz scheinbarer Kompatibilität zu anderen Computern nicht möglich, Programme auszutauschen oder Daten zu übertragen. Auch das CP/M des Commodore 64 hat seine Eigenheiten. So ist zum Beispiel das IOBYTE des CP/M Betriebssystems überhaupt nicht implementiert, zum anderen kann der Commodore 64 nur 40 Zeichen pro Bildschirm darstellen und auch das Format der Disketten ist nicht zu anderen Rechnern kompatibel. Aber zu diesen Problemen, UND WIE MAN SIE BEHEBEN KANN, kommen wir später noch.

Sie haben gerade schon gesehen, das CP/M einige Standards verlangt:

- 1.) Es muß mindestens ein 48K RAM Rechner vorhanden sein.
- 2.) CP/M hat den freien Programmierspeicher ab Adresse \$0100
- 3.) In der Regel verlangen die meisten Programme eine Bildschirmdarstellung von 80 Zeichen in 24 (oder 25) Zeilen.

4.) Die CP/M Software wird fast ausschließlich auf großen, 8'' Floppys ausgeliefert.

Auf diese Standards wollen wir nun kurz eingehen. Als erstes wäre da der Arbeitsspeicher von 48K. Sehr viele Microcomputer können heute auf diese Speicherkapazität ausgebaut werden. Der Commodore 64 besitzt Sie schon von vorne herein. Und noch etwas ist bei diesem Rechner interessant: Man kann ja bekanntlich den ganzen ROM ausblenden (s. 64 intern). Dadurch lassen sich bei diesem Rechner Sachen verwirklichen, auch unter CP/M, die bei anderen Rechnern, die ebenfalls einen bestimmten ROM-Bereich besitzen, unmöglich oder zumindest nur unter großen Schwierigkeiten zu erreichen sind.

So gibt es zum Beispiel Computer, deren ROM in einem, für CP/M, so unglücklichen Bereich liegt (z.B. ab 0000), daß auf diesem Computer das ganze CP/M Betriebssystem anders aufgebaut sein muß. Jeder Programmierer, der auf ein solches Betriebssystem trifft, und der auf CP/M-Routinen zurückgreifen will, bekommt beim Programmieren graue Haare. Nichts von Kompatibilität zu sehen. Der Commodore 64 ist da glücklicherweise anders strukturiert. Hier liegt das CP/M da wo es hingehört, nämlich ab Adresse \$0100.

Das erste Problem stellt sich eigentlich erst bei der Zeichendarstellung auf dem Bildschirm dar. CP/M Programme, so wie WordStar, dem Star unter den Textverarbeitungsprogrammen, DataStar, dem Dateiverwaltungsprogramm, und anderen Programmen, benötigen zur richtigen Funktionsweise einen 80 Zeichen Bildschirm. Der Commodore 64 hat nur leider nur 40 Zeichen. Aber auch hier gibt es für den Anwender das richtige Mittel. Mehr darüber im Kapitel 6.3.

Das letzte Problem bereitet noch die Floppy. Wie schon gesagt, wird die meiste Software auf 8'' Floppys angeboten. Die 1541 Floppys sind da also nicht so glücklich - das denkt man

zumindest. Es ist aber vielmehr so, daß fast alle Microcomputer-Hersteller immer mehr dazu übergehen, 5 1/4'' Floppys zu verwenden. Also fällt eigentlich bei jedem Microcomputer die Datenübertragung von 8'' auf 5 1/4'' an. Wie dies geschieht, erfahren Sie im Kapitel 6.3

Woraus besteht CP/M ?

CP/M ist ein Betriebssystem, das aus mehreren Teilen zusammengesetzt ist. Genauer gesagt handelt es sich um 4 Teile, auf die wir nun etwas näher eingehen wollen.

1.) BIOS (Basic Input/Output System): Wie sein Name schon sagt, ist das BIOS für die Kommunikation mit der Außenwelt zuständig. Wenn also zum Beispiel eine Information an den Drucker, an das Terminal (Bildschirm) oder an die Floppy gegeben werden soll. Um dem Betriebssystem mitzuteilen, was zu tun ist, besitzt das BIOS einige Funktionsaufrufe. Die ganze Tabelle zeigen wir Ihnen am Schluß dieser 4 Teile.

2.) BDOS (Basic Disk Operating System): Dieser Teil steuert die Floppys - also ihren Inhalt, die Verwaltung des Inhaltsverzeichnis und die eigentlichen Lese- und Schreibkommandos. Auch diese Vorgänge werden alle über einzelne Funktionsaufrufe gesteuert. Dazu kommen wir gleich.

3.) CCP (Console Command Processor): Irgendwie muß dem Betriebssystem ja mitgeteilt werden, was Sie eigentlich wollen. Diese Mitteilung erfolgt in der Regel über die Tastatur des Commodore 64. CCP leitet Ihre Befehle also an das CP/M System weiter - man kann es sich auch als Telefonzentrale vorstellen, wo die ganzen Verbindungen zwischen den einzelnen Teilnehmern, in unserem Fall den CP/M-Teilen, getätigt werden.

4.) TPA (Transient Program Area): Dieses ist nun der eigentliche, freie, Programmspeicher, den der Anwender

belegen kann. Wenn Sie also ein Programm schreiben, oder eines anwenden, so ist dies der Arbeitsspeicher hierfür.

Und so liegen BIOS, BDOS, CCP und TPA im Hauptspeicher:

Bezeichnung -----	ab Adresse -----
FDOS (BIOS + BDOS)	\$9C00
CCP	\$9400
TPA	\$0100
System Parameter	\$0000

Hier nun im Detail die FDOS Funktionen:

Funktion: -----	Bezeichnung: -----
BIOS	
00	System Reset
01	Lese ASCII Zeichen von Terminal
02	Sende ASCII Zeichen zum Terminal
03	Lese ASCII Zeichen von Leser
04	Sende ASCII Zeichen zum Stanzer
05	Sende ASCII Zeichen zum Drucker
06	Empfange/Sende Zeichen zur Console
07	Lese Status einer Geräteeinheit
08	Sende Status zu einer Geräteeinheit
09	Sende Zeichenkette aus Buffer
10	Lese Zeichenkette in Buffer
11	Lese Status der Consoleeinheit

BDOS

12	Lese Versionsnummer von CP/M
13	Disk Reset
14	Wähle Laufwerksnummer
15	Öffne Datei (OPEN)
16	Schließe Datei (CLOSE)
17	Suche nach erstem Programm im FCB
18	Suche nach nächstem Programm im FCB
19	Lösche Programm (DELETE)
20	Lese aus sequentieller Datei
21	Schreibe in sequentielle Datei
22	Erstelle Datei
23	Ändere den Programmnamen (RENAME)
24	Gebe mögliche Laufwerke an
25	Lese momentane Laufwerksnummer
26	Setze DMA Adresse
27	Lese Adresse
28	Setze Schreibschutz
29	Lese Schreib/Lese Zeiger
30	Setze Schreib/Lese Zeiger
31	Lese Adresse der Diskparameter
32	Lese/Setze Benutzer-Kennung
33	Lese aus Random-Datei
34	Schreibe in Random-Datei
35	Berechne Programmlänge
36	Lese Adresse des Records

Alle diese Funktionen werden nach einem ganz bestimmten Muster aufgerufen. Um dieses zu verdeutlichen, müssen wir uns nun etwas mit der 8080 Maschinensprache befassen. Da CP/M auf diesem Prozessor entwickelt wurde, und der Z-80 Prozessor, der sich in dem Commodore 64 CP/M Modul befindet, auch diese Sprache versteht, werden die meisten CP/M-Anwendungen in dieser Sprache vorgestellt. Falls Sie sich noch nicht mit diesem Prozessor und seiner Sprache auskennen, sich aber intensiver mit CP/M befassen wollen, so können wir Ihnen nur empfehlen, sich nicht nur ein ausführliches CP/M-Handbuch durch zu lesen (z.B. VOM UMGANG MIT

CP/M -- Bernd Pol -- IWT Verlag, oder CP/M Handbuch -- RODNAY ZAKS), sondern auch Z-80 oder 8080 Handbücher zu Ihrer Standard Literatur zu machen.

Wie sehen die Funktionsaufrufe aus ?

Um zum Beispiel die Versionsnummer des Commodore CP/M festzustellen (es handelt sich um 2.2) läßt sich folgende Routine anwende:

```
MVI      C,12      ;FUNKTION 12
CALL     0005      ;BDOS EINSPRUNG
CPI      20H       ;$20 BEDEUTET CP/M 2.0
.         .
.         .
```

Zunächst wird das C-Register mit dem Wert 12 geladen, also die Funktionsnummer zum Lesen der CP/M Version. Dieses C-Register wird immer mit der Funktionsnummer geladen, bevor zur Adresse 0005 verzweigt wird. CP/M weiß nun, daß wir die CP/M Version erfahren wollen, und verzweigt automatisch an die Stelle des CP/M Systems, wo die Versionsnummer gelesen wird. BDOS springt dann wieder in unsere Maschinenroutinen zurück, und hat die Versionsnummer im Akku bereitgestellt. Ist der Wert des Akkus nun \$20, so wissen wir, daß es sich bei dieser Version des CP/M mindestens um eine 2.0 Version handelt. Diese Information kann sehr interessant sein, wenn wir ein Programm schreiben wollen, das Random-Dateien verwaltet. Alle Versionen vor 2.0 können nämlich nur sequentielle Dateien (s. Kapitel 8). Wenn wir also unser Programm auch noch auf einem anderen CP/M Rechner, als unseren Commodore 64, anwenden wollen, so können wir leicht feststellen, ob dieses Programm überhaupt auf dem Rechner läuft.

Eine sehr wichtige Rolle spielen bei den CP/M Funktionsaufrufen jeweils die verschiedenen Register. An erster Stelle ist hier das C Register zu nennen, in dem jeweils die Funktionsnummer, vor dem BDOS/BIOS Aufruf, abgelegt wird. Nach der Durchführung der

angesprochenen Routine stehen dann in den verschiedenen anderen Registern die gewünschten Informationen. Natürlich kann es auch vorkommen, daß keine Eingabe, sondern die Ausgabe eines Zeichens erfolgen soll. Dann ist natürlich der Aufruf etwas anders. Das Register C enthält weiterhin den Funktinscode. Die anderen Register bzw. das andere Register werden aber schon vor dem Aufruf geladen.

Warum Funktionscodes ?

Wie Sie ja nun schon wissen, ist einer der Vorteile des CP/M Betriebssystems, daß ein CP/M Programm generell auf jedem CP/M Rechner laufen soll. Dazu können aber von Rechner zu Rechner kleine Änderungen im BIOS/BDOS oder CCP notwendig sein. Um zu gewährleisten, daß die Programme laufen, ohne daß sich der Programmierer sorgen um den Aufbau des CP/M machen muß, gibt es diesen BIOS/BDOS Aufruf bei der Adresse 0005. So kann also ein bestimmter Teil des CP/M Betriebssystems verändert werden (und wenn es nur ein Byte ist), ohne daß die schon vorhandenen Programme neu angepaßt werden müßten. Den selben Vorteil haben Sie auch beim KERNAL des Standard-Commodore 64, also ohne CP/M Betriebssystem. Dort stehen die verschiedenen Einsprungpunkte zu den unterschiedlichsten E/A (Ein- Ausgabe) Routinen. Würde nun eine dieser Routinen verändert werden, so ist es trotzdem möglich die alten Programme weiter zu verwenden - die merken überhaupt nichts von der Änderung des Betriebssystems.

Die CCP Befehle

Wie schon oben erwähnt, stellt der CCP eine Schnittstelle zu den anderen CP/M Programmen dar. Er verfügt über einen eigenen kleinen Befehlsvorrat. Außerdem lassen sich von hier aus auch ablauffähige Programme ausführen. Alles in allem also die Verbindung zwischen dem CP/M Betriebssystem und dem Anwender.

Hier die Befehle im einzelnen:

1.) DIR (Directory): Mit diesem Befehl können Sie sich das Inhaltsverzeichnis der Diskette ansehen. Es gibt hierzu verschiedene Möglichkeiten:

- DIR zeigt den gesamten Inhalt der Diskette an, die sich in dem momentan angesprochenen Laufwerk befindet.

- DIR B: zeigt den gesamten Inhalt der Diskette an, die sich im Laufwerk B (1) befindet. An Stelle von B kann auch A stehen. Dann wird der Inhalt von Laufwerk A (0) angezeigt.

- DIR <NAME.EXT> zeigt nur an, ob das gesuchte Programm auch auf der Diskette ist. Der NAME kann beliebig gewählt werden, darf aber 8 Stellen nicht überschreiten und keine Sonderzeichen enthalten. EXT ist ein 3-stelliges Anhängsel des Programmnamens. Es gibt an um welche Art von Programm es sich handelt. So können zum Beispiel nur solche Programme gestartet werden, die das Kürzel COM haben. TXT würde bedeuten, daß es sich hier um eine TEXT-Datei handelt, BAS ist ein BASIC-Programm usw.

- DIR <*.EXT> zeigt alle Programme an, die das Kürzel EXT haben. So würde *.COM alle Programme anzeigen, die direkt zu starten sind.

Es gibt noch eine Reihe anderer Formen für den DIR Befehl. Sehen Sie dafür im CP/M Handbuch nach.

Hier nun das Format des Inhaltsverzeichnis bei der Anwendung des DIR-Befehls auf dem Commodore 64:

A>DIR

A: MOVCPM	COM : PIP	COM
A: SUBMIT	COM : XSUB	COM
A: ED	COM : ASM	COM
A: DDT	COM : LOAD	COM
A: STAT	COM : SYSGEN	COM

A: DUMP COM : DUMP ASM
A: COPY COM : CONFIG COM
A>

2.) ERA (Erase): Dieser Befehl löscht ein Programm oder auch mehrere Programme aus dem Inhaltsverzeichnis. Auch hier gibt es wieder verschiedene Möglichkeiten:

- ERA <NAME.EXT> s.o.

- ERA <*.EXT> s.o

3.) REN (Rename): Mit diesem Befehl können Sie einem bestehenden Programm oder einer bestehenden Datei einen neuen Namen geben. Hier gibt es nur eine Form

- REN <NEUER NAME> = <ALTER NAME>

4.) TYPE: Dieser Befehl hat seine Wirkung nur bei Textdateien. Wollen Sie also zum Beispiel eine Datei, die Sie mit einem Dateiverwaltungsprogramm erstellt haben, auf dem Bildschirm (oder den Drucker) auflisten, so können Sie diesen Befehl benutzen. Er hat die Form:

- TYPE <DATEINAME.EXT>

In den meisten Fällen wird das Kürzel vom Type TXT, PRN oder dergleichen sein.

5.) SAVE: Dieser Befehl ist auf den ersten Blick recht kompliziert. Normalerweise wenden Sie diesen Befehl dann an, wenn Sie ein Programm mit DDT (s.u.) geändert haben und es wieder abspeichern möchten. Aber auch beim Abspeichern von einer modifizierten CP/M Version (s.u.). Die Form lautet:

- SAVE <SEITENZAHL> <NAME.EXT>

Die Seitenzahl gibt an, wieviele 'Seiten' je 256 Byte abgespeichert werden sollen. So speichert der Befehl:

SAVE 50 TEST.COM

das Programm TEST.COM von der Adresse \$0100 (Anfang TPA) bis zur Adresse \$32FF ab. Die Länge beträgt also 50*256 Byte.

6.) USER: Mit Hilfe dieses Befehls können Sie ein Inhaltsverzeichnis für unterschiedliche Benutzer aufteilen. So ist es möglich, bestimmte Bereiche des Inhaltsverzeichnis gegen Benutzung von anderen Anwendern (an anderen Rechnern) abzusichern. Beim Commodore 64 hat dieser Befehl allerdings keine größere Bedeutung. Er ist hier nur der Form halber erwähnt. Die Form lautet:

- USER BENUTZERNUMMER

Die Benutzernummer kann einen Wert zwischen 0 und 15 haben. Durch Angeben einer Zahl gelangt man also ins Inhaltsverzeichnis des entsprechenden Benutzers. Gesetzt ist die Nummer 0.

Dieses sind die Befehle, die jeder Computer mit einem CP/M Betriebssystem versteht. Nur den USER-Befehl gibt es erst seit der CP/M-Version 2.0. Alle anderen aber gehören zum Standard Befehlsvorrat vom CP/M. Auf den nächsten Seiten geben wir Ihnen noch einen kurzen Einblick in die Standardprogramme von CP/M: PIP, ED, DDT, STAT. Sie gehören zwar zum Lieferumfang jeder CP/M Systemdiskette, sind aber nicht als Befehle, sondern als Programme zu verstehen. Sie bilden zwar in ihrer Gesamtheit eine Befehlserweiterung, müssen aber erst von Diskette nachgeladen werden.

6.2 DER UMGANG MIT DEN EINZELNEN CP/M PROGRAMMEN

Was wäre CP/M ohne sein Grundgerüst an Hilfs-Programmen ? DIGITAL RESEARCH, der Hersteller von CP/M, hat dafür gesorgt, daß der Anwender schon vom ersten Augenblick an mit der Programmierung beginnen kann - ein Wermutstropfen den Commodore 64 Anwender: Eine spezielle BASIC-Version gibt es nicht standardmäßig. Aber nichts ist für CP/M unmöglich - auch ein BASIC-Interpreter kann später eingesetzt werden.

Nun aber zu den einzelnen Programmen. Folgende Programme gehören zum CP/M-Standard:

- STAT.COM Ein Programm mit denen die verschiedensten Systeminformationen abgefragt werden können (z.B. Speicherplatz auf Disk, angeschlossene Schnittstellen, Gerätezuordnungen usw.)
- ASM.COM Ein Assembler. Wie schon oben erwähnt, wird häufig noch in 8080 Assembler-Sprache programmiert. So ist also auch dieser Assembler auf die Programmierung im 8080-Code ausgelegt.
- LOAD.COM Dieses Programm macht aus den fertig assemblierten Programmen (Programme mit dem Kürzel .HEX) ablauffähige Programme.
- DUMP.COM Mit diesem Programm kann ein Programm (.COM) auf den Bildschirm in lesbare HEX-Form dargestellt werden.
- PIP.COM Auf gut Deutsch könnte man zu diesem Programm sagen: Ein Programm zum Austausch von Daten zwischen verschiedenen Peripherieeinheiten.
- ED.COM Der CP/M Texteditor. Dieses Programm ist ein wichtiges Hilfsmittel zum Erstellen von Texten,

Maschinenprogrammen und dergleichen.

- SYSGEN.COM Wo man mit PIP nur Dateien kopieren konnte, da kann man mit SYSGEN die einzelnen BIOS-Spuren auf Disk schreiben und damit sogar ein neues BIOS generieren (s. Kapitel 6.3).
- MOVCPM.COM Dieses Programm paßt das Standard CP/M an Ihren speziellen Computertyp an (s. Kapitel 6.3)
- SUBMIT.COM Es kommt immer wieder vor, daß bestimmte Eingaben immer wieder, zu einem gleichen Zeitpunkt, gemacht werden müssen. Um Ihnen die Arbeit zu erleichtern, steht Ihnen das SUBMIT Programm zur Verfügung.
- XSUB.COM Auch dieses Programm erleichtert Ihnen die Arbeit mit immer wieder vorkommenden Befehlsfolgen. Allerdings wird dieses Programm nur im Zusammenhang mit SUBMIT angewendet. Es erlaubt die manuelle Eingabe, also über Tastatur, während des automatischen Ablaufs durch SUBMIT.

Wir können uns an dieser Stelle nicht ausführlich mit allen Programmen befassen. Wir begnügen uns an dieser Stelle nur mit einer kurzen Einführung in die verschiedenen Programmen - CP/M Handbücher geben näher Auskunft darüber.

STAT

Eines der wichtigsten Programme ist sicherlich das STAT-Programm. Man könnte es auch STATUS-Programm nennen. Denn mit Hilfe dieses Programmes kann man eine ganze Menge Informationen über den Zustand des gesamten Systems erlangen. Es kann nicht nur der verbliebene Speicherplatz auf Diskette und die Länge der Programme bestimmt werden, es können auch die Schreib/Lese Zeiger geändert werden, die anzeigen, ob auf die Diskette geschrieben und von ihr gelesen werden darf, oder vielleicht nur lesender

Zugriff besteht. Und auch Informationen über die verschiedenen Schnittstellen können abgefragt und geändert werden. Hier gibt es allerdings bei der Commodore 64 CP/M Version eine Einschränkung. Das IOBYTE ist nicht implementiert, das heißt, daß die einzelnen Zuordnungen nicht geändert werden können. Das ist aber für die normale Anwendung nicht besonders störend.

Hier nun ein Beispiel zum STAT Befehl:

```
A>STAT VAL:
```

```
TEMP R/O DISK: D:=R/O
SET INDICATOR: D:FILENAME.TYP $R/O $R/W $SYS $DIR
DISK STATUS   : DSK: D:DSK:
USER STATUS   : USR:
IOBYTE ASSIGN:
```

```
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
```

Hier erfahren wir nun, welche Werte, und in welcher Form wir (theoretisch) ändern können - denn wie schon gesagt, die Werte unter IOBYTE ASSIGN sind beim Commodore 64 nicht zu ändern.

Wenn wir nun zum Beispiel den Schreibschutz auf der Diskette ändern wollen, so genügt ein: STAT A:=R/O. Dieser Schreibschutz bleibt solange erhalten, wie das Gerät angeschaltet bleibt.

Um nun noch die angeschlossenen CP/M Schnittstellen zu erfahren tippen Sie einfach: STAT A:DEV:

Der Ausdruck beim Commodore 64 sieht dann folgendermaßen aus:

```
A>STAT DEV:
CON: IS TTY:
```

RDR: IS TTY:
PUN: IS TTY:
LST: IS TTY:

Eine etwaige Änderung eine dieser Zuweisungen würde ohne Änderung auf das System bleiben. So z.B.: STAT LST:=LPT:.

Sehr aufschlußreich ist auch die Information über die Diskettencharacteristik. Hier erfahren Sie wie groß die Kapazität der Diskette ist, wie ihr Aufbau aussieht und manches mehr. Die entsprechende Abfrage lautet:

A>STAT DSK:

A: DRIVE CHARACTERISTICS
1088: 128 BYTE RECORD CAPACITY
136: KILOBYTE DRIVE CAPACITY
64: 32 BYTE DIRECTORY ENTRIES
64: CHECKED DIRECTORY ENTRIES
128: RECORDS/ EXTENT
8: RECORDS/ BLOCK
34: SECTORS/ BLOCK
2: RESERVED TRACKS

Dies soweit die Erklärung von STAT. Nun zu einem weiteren Programm,

PIP

Dieses Programm ist ein universelles Programm zum Kopieren von Dateien. Es besteht nicht nur die Möglichkeit zwischen verschiedenen Floppys zu kopieren, sondern auch eine Textdatei auf den Drucker auszugeben. Dies hat den Vorteil, daß das ganze Listing formatiert, das heißt mit Zeilennummern und Tabulatoren, ausgedruckt werden kann. Um eine ganze Diskette zu kopieren, wird folgende Form angewendet: PIP B:=A:*. *. Dieser Befehl kopiert alle Dateien und Programme von Laufwerk A nach Laufwerk B.

Um eine Textdatei auf dem Drucker auszugeben, müssen Sie folgenden Befehl anwenden: PIP LST:=DUMP.ASM. Nun wird, sofern der Drucker angeschlossen ist, das ganze Programm DUMP.ASM auf diesem ausgegeben.

ED

Dieser Texteditor ermöglicht sowohl die Eingabe von Texten, als auch die Eingabe von Maschinenprogrammen und allen Programmen, die später kompiliert werden sollen, wie zum Beispiel FORTRAN oder COBOL Programme. Die Arbeit mit ED erfordert etwas Übung - Sie ist für den Commodore Anwender sicherlich etwas umständlich und gewöhnungsbedürftig, aber man ist recht schnell mit der Arbeitsweise dieses einfachen Editors vertraut. Hier ein Beispiel:

```
A>ED TEST.TXT
NEW FILE
*I
DIES IST DIE ERSTE TEXTZEILE
UND DIES DIE ZWEITE
<CTRL>-Z           Drücken Sie die CTRL und die Z
                    Taste gleichzeitig
*E
```

Mit diesem kleinen Beispiel schreiben Sie Ihre erste eigene Textdatei auf Diskette. Wenn der Schreibvorgang beendet ist, meldet sich das Betriebssystem wieder mit A>. Versuchen Sie doch nun einmal den TYPE-Befehl (s.o.).

Mit ED können Sie aber nicht nur Texte (Programme) erstellen und wegschreiben, sondern Sie können Sie auch wieder einlesen, verändern und weitere Manipulationen an ihnen vornehmen.

6.3 DIE ANPASSUNG VON STANDARD CP/M SOFTWARE AN DEN C-64

Auf was ist bei der Anpassung von CP/M Software an die Hardware des Commodore 64 zu achten ? Zunächst müssen Sie bedenken, daß Sie nur einen 40 Zeichen Bildschirm zur Verfügung haben. Da aber die meiste Anwendersoftware auf die 80 Zeichen eines Standardterminals zugeschnitten ist, benötigt man hierfür eine 80 Zeichen-Karte.

Außerdem ist bei der Benutzung von CP/M eine große und schnelle Floppy von Vorteil.

Durch dieses System haben Sie dann Zugriff auf das größte Softwareangebot der Welt.

Zur Anpassung von CP/M an einen Computer verfügt das Betriebssystem über zwei Programme: MOVCPM und SYSGEN.

Mit MOVCPM können Sie das Betriebssystem auf eine bestimmte Speicherkonfiguration einstellen. So ist es möglich, den maximalen Speicherplatz voll auszunutzen. Beim Start des CP/M Betriebssystems meldet sich der Rechner mit der Meldung 44K CP/M. Es ist aber sogar möglich, den vollen Speicher für CP/M zu nutzen. Wie schon gesagt, ist es ja möglich, den Commodore 64 ganz auf RAM umzuschalten.

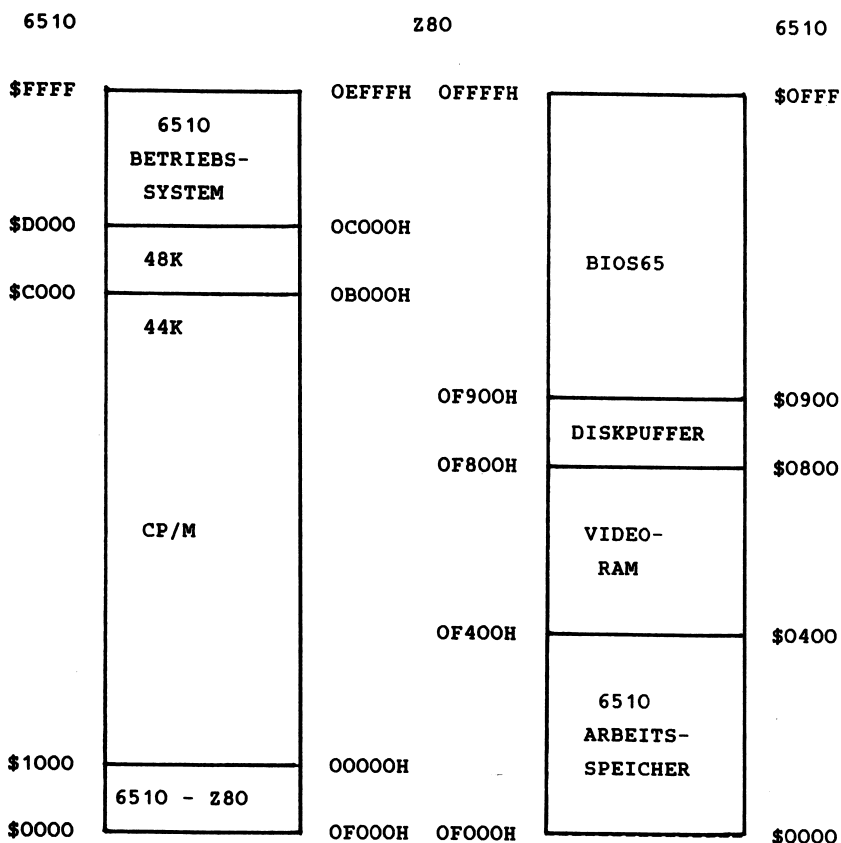
Mit SYSGEN kann dann das CP/M System auf Ihre Disketten kopiert werden. Schon haben Sie einen CP/M Rechner, der noch mehr kann.

6.4 DIE SPEICHERVERWALTUNG DES Z80-PROZESSORS

Der Z80-Prozessor auf der CP/M-Karte kann die vollen 64 KByte des Commodore 64 adressieren. Da der Z80 die Adresse Null als Resetadresse benötigt, diese Adresse jedoch im Commodore 64 mit dem Prozessorport des 6510 belegt ist, ist man bei der Adressierung des Speichers durch den Z80-Prozessor einen anderen Weg gegangen. Die CP/M-Karte enthält die Hardware zur Erzeugung eines Offsets bei der Adressierung des Speichers durch den Z80. Der Offset beträgt \$1000 gleich 4096. Dies ergibt folgende Situation: Spricht der Z80-Prozessor die Speicherzelle 0 an, so wird durch hardwaremäßige Manipulation der Adressleitungen dabei die Adresse \$1000 des Commodore 64 unter dem 6510 angesprochen. Da der Z80 zusammen mit dem 6510 betrieben werden soll, steht somit dem Z80-Prozessor ein durchgehender Speicherbereich ab Adresse 0 zur Verfügung. Will man aus einer 6510-Adresse die entsprechende Z80-Adresse berechnen, so zieht man einfach \$1000 ab und erhält dann die Z80-Adresse. Alternativ kann man auch \$F000 dazu addieren und einen eventuellen Übertrag unberücksichtigt lassen. Durch diesen Trick wird ein Bereich von 4 KBytes von Adresse 0 bis \$1000 des 6510 an des Ende des Adressbereichs des Z80 gelegt. Dieser Speicherbereich wird vom 6510 als Zero-Page, Stack und Arbeitsspeicher belegt. Außerdem liegt dort auch das Video-RAM. Die restlichen 2 Kbyte von \$800 bis \$FFF werden bei der CP/M-Karte zur Übergabe von Daten vom 6510 zum Z80 und umgekehrt sowie als Programmspeicher für 6510-Ein-Ausgabe Routinen benutzt. Der Z80 delegiert die Ein-Ausgabe an den 6510. Auf der nächsten Seite wird diese Situation noch einmal grafisch dargestellt.

Im folgenden sind die wichtigsten gemeinsamen Adressen des CP/M zwischen Z80 und 6510 zusammengefasst.

<u>Label</u>	<u>6510</u>	<u>Z80</u>	<u>Bedeutung</u>
HSTBUF	\$0800	0F800H	256 Byte Diskettenpuffer
CMD	\$0900	0F900H	Befehlsregister für 6510
DATA	\$0901	0F901H	Datenregister
SECTOR	\$0902	0F902H	Sektorregister
TRACK	\$0903	0F903H	Trackregister
DISKNO	\$0904	0F904H	Register für Laufwerknummer
KEYCHAR	\$0905	0F905H	Nummer der gedrückten Taste
MODESW	\$DE00	0CE00H	Schalter für 6510 / Z80
IOTYPE	\$0CFF	0FCFFH	I/O Konfiguration



6.5 DIE DISKETTENVERWALTUNG UNTER CP/M

Eine Diskette wird in mehrere konzentrisch angelegte Spuren (Tracks) unterteilt, und diese wiederum in einzelne Sektoren (Segmente). Bei der VC 1541 sieht das folgendermaßen aus:

Track	Sektoren
1-17	0-20
18-24	0-18
25-30	0-17
31-35	0-16

Dies sind insgesamt 683 Sektoren (Blocks), von denen intern die Spur 18 für das Disketteninhaltsverzeichnis benutzt wird, so daß Ihnen noch 664 Blocks zur Verfügung stehen.

Unter CP/M sind die ersten beiden Tracks für das CP/M System selbst reserviert; die anderen Programme sind für Daten- und Programmspeicherung frei. Track 18 wird ausgespart. Da die Diskettenverwaltung unter CP/M keine variablen Sektorenzahlen pro Track verarbeiten kann, werden nur die Sektoren 0 bis 16 belegt. Im Endeffekt haben Sie also noch 32 Tracks zu je 17 Sektoren, das sind 544 Blocks zu 256 Byte gleich 143 KByte. Davon geht noch etwas für das CP/M-Directory ab (max. 64 Einträge zu je 32 Byte gleich 2 KByte). Diese Daten sind im BIOS (Basic Input/- Output System) des CP/M im sogenannten Disk Parameter Block gespeichert und können vom Anwender an seine Diskettenkapazität angepasst werden. Im BIOS wird zum Beispiel auch die Spur 18 (die das Commodore Directory enthält) ausgespart. Schauen Sie dazu bitte im BIOS-Listing des CP/M 64 nach.

Die Belegung der ersten beiden Tracks durch das Betriebssystem sieht folgendermaßen aus:

Track 1 Sektor 0 enthält das Ladeprogramm 'CPM'. In Track 1 liegen von Sektor 1 bis 5 das 'BIOS 65', das sind die I/O-Routinen für den 6510 sowie der Kaltstartlader für das CP/M. Das Programm 'CPM' lädt diese 5 Sektoren in den Speicherbereich \$0A00 bis \$0EFF. Danach wird der Bereich von \$0E00 bis \$0EFF an die Adresse \$1000 bis \$10FF übertragen. Dies ist die Z80 Adresse 0, an die der Kaltstartlader übertragen wird. Anschließend schaltet der 6510 sich selbst aus und gleichzeitig den Z80 ein. Der Z80 führt jetzt das Programm ab Adresse 0 aus, das das CP/M Betriebssystem von Diskette lädt. CCP und BDOS (Command Console Prozessor und Basic Disk Operation System) belegen 22 Sektoren auf Spur eins und zwei; und zwar von Track 1 Sektor 6 bis Track 2 Sektor 10. Diese Sektoren werden auch bei jedem Warmstart mit Control C wieder neu von Diskette geladen; für jeden geladenen Sektor erscheint ein Sternchen auf dem Bildschirm. Das BIOS, das nur beim Kaltstart mit geladen wird, belegt 5 Sektoren von Track 2 Sektor 11 bis Sektor 16. Das Directory belegt auf Track 3 die Sektoren 0 bis 7.

Beim CP/M-Hilfsprogramm 'COPY' werden entsprechend der gewählten Option dann jeweils nur die erforderlichen Sektoren gelesen und auf die neue Diskette geschrieben, z.B. bei 'system tracks only' nur die Sektoren 1 und 2 sowie 18 und Track 3 für das CP/M-Directory.

Wollen Sie eine andere Floppy anschließen, z.B. über den IEC-Bus, so müssen Sie Track- und Sektorenaufteilung der Diskettenstation kennen. Bei der Floppy 4040 ist keine Anpassung erforderlich, da 4040-Disketten völlig kompatibel zur 1541 sind. Um bei einer 8050 oder 8250-Floppy die erhöhte Speicherkapazität auszunutzen, ist eine Anpassung im BIOS erforderlich. Dazu dient der 'Disk Parameter Block'. Dort stehen die Werte für Sektoren pro Track (23 für 8050) sowie

die Diskettenkapazität. Zusätzlich müssen dann noch anstelle der Spur 18 die Spuren 38 und 39 ausgespart werden, da dort das Commodore-Directory steht. Diese Änderungen sind auch im COPY-Programm erforderlich.

6.6 DIE ZUSAMMENARBEIT ZWISCHEN 6510 UND Z80

Wenn Sie auf dem Commodore 64 mit CP/M arbeiten, teilen sich zwei Mikroprozessoren die Arbeit. Während der Z80 das eigentliche CP/M bedient, wird der 6510 für Ein-/Ausgabe Operationen herangezogen, da das Commodore 64 Betriebssystem diese Routinen bereits zur Verfügung stellt. Folgende Aufgaben werden vom Z80 an den 6510 delegiert:

Befehlsnummer	Operation
0	einen Sektor von Diskette lesen
1	einen Sektor auf Diskette schreiben
2	Abfrage der Tastatur
3	Zeichen auf Bildschirm ausgeben
4	Status des Druckers holen
5	Zeichen auf Drucker ausgeben
6	Diskette formatieren
7 - 9	reserviert für zukünftige Erweiterungen, z.B. serielle Ein-/Ausgabe

Die beiden Prozessoren können nicht gleichzeitig, sondern nur abwechselnd arbeiten. Zum Umschalten zwischen den beiden Prozessoren dient die Adresse \$DE00 des 6510 bzw. 0CE00H des Z80. Will der Z80 eine Ein-/Ausgabe Funktion vom 6510 ausführen lassen, so übergibt er an den 6510 die Nummer des gewünschten Befehls aus der obigen Tabelle und soweit erforderlich weitere Parameter in einem gemeinsamen Speicherbereich und schaltet sich dann durch Einschreiben einer '1' an die Adresse 0CE00H selber aus und den 6510 damit ein. Dieser weiß nun anhand der Befehlsnummer welche Funktion er für den Z80 ausführen soll. Ist der Befehl ausgeführt, so schaltet sich nun der 6510 durch Einschreiben einer '0' an Adresse \$DE00 selbst aus und den Z80 damit wieder ein. Der Z80 kann nun in seinem Programm wieder an der Stelle

fortfahren, an der er die Kontrolle an den 6510 abgegeben hatte. Aus Hardwareerfordernissen muß der erste Befehl nach dem Wiedereinschalten von 6510 und Z80 jeweils ein NOP-Befehl (No Operation) sein.

Zur Parameterübergabe dient der Speicherbereich ab Adresse \$900 oder 0F900H.

6510	Z80	Bedeutung
\$900	0F900H	Befehlsnummer
\$901	0F901H	Daten zur Ein- oder Ausgabe
\$902	0F902H	Sektornummer
\$903	0F903H	Tracknummer
\$904	0F904H	Diskettennummer
\$905	0F905H	Tastennummer bei Tastaturabfrage

Der Bereich von \$800 bis \$8FF bzw. 0F800H bis 0F8FFH dient als Puffer für einen zu schreibenden bzw. einen gelesenen Sektor von Diskette.

Das Lesen bzw. Schreiben von Diskettensektoren geschieht im Direktzugriff über Block-Read und Block-Write Befehle.

Die Tastaturabfrage liefert lediglich die Nummer der gedrückten Taste. Die Zuordnung einer Taste zu einem ASCII-Wert geschieht vom BIOS aus über eine Tabelle von Adresse \$D00 bis \$DFF bzw. 0FD00H bis 0FDFFH. Ab Adresse \$C00 bzw. 0FC00H stehen die Adressen der Texte, mit denen die Funktionstasten belegt sind. Die Texte selbst, die bis zu 16 Zeichen umfassen können, stehen dann ab Adresse \$C10 bzw. 0FC10H. Diese Zuordnungen können mit dem Programm CONFIG beliebig geändert werden.

6.7 KOMMENTIERTES CP/M-BIOS-LISTING

Auf den folgenden Seiten finden Sie nun ein ausführlich dokumentiertes Listing des Basic I/O Systems (BIOS) des Commodore 64 CP/M. Das BIOS ist in 4 Teile gegliedert. Der erste Teil umfaßt das Programm "CPM" auf Ihrer Systemdiskette und wird mit LOAD "CPM",8 in den Adressbereich von \$800 bis \$8FF geladen. Dieses Programm lädt das BIOS65 in den Speicher von Adresse \$A00 bis \$DFF. Dies ist der Teil des BIOS, der vom 6510-Prozessor ausgeführt wird. Der dritte Teil ist das sogenannte 'BOOT'-Programm, das beim Kaltstart des CP/M den vierten Teil, das eigentliche Z80-BIOS, lädt.

6.7.1 LADEPROGRAMM FÜR DAS BIOS 65

```
***** Commodore 64 Ladeprogramm für CP/M
080F 78      SEI
0810 20 E7 FF JSR $FFE7      CLALL, alle I/O Kanäle schließen
0813 A9 0F    LDA #$0F        logische Filenummer 15
0815 A2 08    LDX #$08        Gerätenummer 8
0817 A0 0F    LDY #$0F        Sekundäradresse 15
0819 20 BA FF JSR $FFBA      Fileparameter setzen
081C A9 00    LDA #$00        kein Filenamen
081E 20 BD FF JSR $FFBD      Parameter für Filenamen setzen
0821 20 C0 FF JSR $FFC0      OPEN Fehlerkanal
0824 A9 02    LDA #$02        logische Filenummer 2
0826 A2 08    LDX #$08        Gerätenummer 8
0828 A0 02    LDY #$02        Sekundäradresse 2
082A 20 BA FF JSR $FFBA      Fileparameter setzen
082D A9 01    LDA #$01        Länge des Filenamens gleich eins
082F A2 B5    LDX #$B5
0831 A0 08    LDY #$08        Zeiger auf "#"
```

0833 20 BD FF JSR \$FFBD	Parameter für Filenamen setzen
0836 20 C0 FF JSR \$FFC0	OPEN Direktzugriffskanal
0839 A9 05 LDA #\$05	
083B 8D B6 08 STA \$08B6	Zähler für 5 Sektoren laden
083E A2 0F LDX #\$0F	15
0840 20 C9 FF JSR \$FFC9	CKOUT, Ausgabe auf Kanal 15
0843 A2 00 LDX #\$00	
0845 A0 0B LDY #\$0B	12 Zeichen
0847 BD AA 08 LDA \$08AA,X	"U1:2 0 1 s" Track 1, Sektor 1 bis 5
084A 20 D2 FF JSR \$FFD2	ausgeben
084D E8 INX	
084E 88 DEY	
084F D0 F6 BNE \$0847	
0851 20 CC FF JSR \$FFCC	CLRCH, Ausgabe wieder auf default
0854 A2 02 LDX #\$02	2
0856 20 C6 FF JSR \$FFC6	CHKIN, Eingabe von Kanal 2
0859 A2 00 LDX #\$00	
085B 20 CF FF JSR \$FFCF	Zeichen holen
085E 9D 00 0A STA \$0A00,X	und speichern
0861 E8 INX	
0862 D0 F7 BNE \$085B	schon 256 Zeichen ?
0864 20 CC FF JSR \$FFCC	CLRCH, Eingabe wieder auf default
0867 EE B3 08 INC \$08B3	Sektornummer erhöhen
086A EE 60 08 INC \$0860	Speicheradresse um eine Page erhöhen
086D CE B6 08 DEC \$08B6	schon 5 Sektoren gelesen ?
0870 D0 CC BNE \$083E	nein, weiter lesen
0872 A2 00 LDX #\$00	
0874 BD 00 0E LDA \$0E00,X	BOOT-Programm für Z80
0877 9D 00 10 STA \$1000,X	nach Z80-Adresse 0000H kopieren
087A E8 INX	
087B D0 F7 BNE \$0874	
087D 20 E7 FF JSR \$FFE7	CLALL, I/O-Kanäle schließen
0880 A9 36 LDA #\$36	
0882 85 01 STA \$01	BASIC-ROM abschalten

0884 A9 09	LDA #\$09	CHR\$(9)
0886 20 D2 FF JSR \$FFD2		Umschaltung erlauben
0889 A9 0E	LDA #\$0E	CHR\$(14)
088B 20 D2 FF JSR \$FFD2		Text-Modus setzen
088E A9 08	LDA #\$08	CHR\$(8)
0890 20 D2 FF JSR \$FFD2		Umschaltung sperren
0893 A9 93	LDA #\$93	CHR\$(147)
0895 20 D2 FF JSR \$FFD2		Bildschirm löschen
0898 A9 0D	LDA #\$0D	CHR\$(13)
089A 20 D2 FF JSR \$FFD2		neue Zeile
089D A9 FF	LDA #\$FF	
089F 8D 00 09 STA \$0900		Befehlsregister
08A2 A9 28	LDA #\$28	Tastenummer
08A4 8D 05 09 STA \$0905		merken
08A7 4C 00 0A JMP \$0A00		zur Hauptschleife 280 - 6510
08AA 55 31 3A 32 20 30 20 31 20 31 0D	"U1:2 0 1 1" Block-Read Befehl	
08B5 23	"#" Filename für Direktzugriff	

6.7.2 BIOS 65

```

***** Hauptschleife Z80 - 6510
OA00 A9 00 LDA #$00
OA02 8D 00 DE STA $DE00      6510 ausschalten
OA05 EA NOP
OA06 20 0C 0A JSR $0A0C      Routine für Z80 ausführen
OA09 4C 00 0A JMP $0A00      zurück zur Schleife

***** Routinen für Z80 selektieren
OA0C AD 00 09 LDA $0900      Befehlskode
OA0F C9 FF CMP #$FF          Z80 nicht aktiv ?
OA11 D0 03 BNE $0A16
OA13 6C FC FF JMP ($FFFC)    RESET des Commodore 64
OA16 C9 0A CMP #$0A          Befehlsnummer größer gleich 10 ?
OA18 90 01 BCC $0A1B          nein
OA1A 60 RTS                  fertig

OA1B D8 CLD
OA1C 18 CLC
OA1D 6D 00 09 ADC $0900      Befehlsnummer * 2
OA20 69 28 ADC #$28          plus $28
OA22 8D 26 0A STA $0A26      indirekte Adresse setzen
OA25 6C 28 0A JMP ($0A28)    Befehl ausführen

***** Sprungtabelle der 10 Funktionen
OA28 3F 0A Funktion 0, READ SECTOR
OA2A 57 0A Funktion 1, WRITE SECTOR
OA2C 86 0A Funktion 2, TASTATUR-ABFRAGE
OA2E 8F 0A Funktion 3, BILDSCHIRM-AUSGABE
OA30 99 0A Funktion 4, DRUCKER-STATUS HOLEN
OA32 9F 0A Funktion 5, DRUCKER-AUSGABE
OA34 49 0B Funktion 6, DISKETTE FORMATIEREN

```

OA36 00 0E	Funktion 7, AUX1, Sprung nach \$E00
OA38 00 0F	Funktion 8, AUX2, Sprung nach \$F00
OA3A 3C 0A	Funktion 9, INDIRECT, Sprung nach (\$906)

***** Funktion 9, INDIRECT

OA3C 6C 06 09 JMP (\$0906)	indirekter Sprung nach (\$906)
----------------------------	--------------------------------

***** Funktion 0, READ SECTOR

OA3F A9 31 LDA #\$31	"1" für "U1" gleich Block Read
OA41 20 F2 0A JSR \$0AF2	sendet Block-Read Befehl an Diskette
OA44 20 DE 0B JSR \$0BDE	Eingabe von Direktzugriffs kanal
OA47 A2 00 LDX #\$00	
OA49 20 CF FF JSR \$FFCF	Zeichen holen
OA4C 9D 00 08 STA \$0800,X	in Puffer speichern
OA4F E8 INX	
OA50 D0 F7 BNE \$0A49	schon 256 Zeichen ?
OA52 F0 60 BEQ \$0AB4	CLRCH

OA54 20 97 0B JSR \$0B97	Diskette initialisieren
--------------------------	-------------------------

***** Funktion 1, WRITE SECTOR

OA57 20 F4 0B JSR \$0BF4	Ausgabe auf Kommandokanal der Diskette
OA5A A0 08 LDY #\$08	8 Zeichen
OA5C BD 80 0B LDA \$0B80,X	"B-P 2 0" , Zeiger auf Sektorbeginn setzen
OA5F 20 D2 FF JSR \$FFD2	ausgeben
OA62 E8 INX	
OA63 88 DEY	
OA64 D0 F6 BNE \$0A5C	
OA66 20 CC FF JSR \$FFCC	CLRCH
OA69 20 CE 0B JSR \$0BCE	Fehlerkanal lesen
OA6C D0 E6 BNE \$0A54	Fehler, dann initialisieren und nochmal
OA6E 20 CC FF JSR \$FFCC	CLRCH
OA71 20 E9 0B JSR \$0BE9	Ausgabe auf Kanal 2
OA74 A2 00 LDX #\$00	

0A76 BD 00 08 LDA \$0800,X	Zeichen aus Puffer
0A79 20 D2 FF JSR \$FFD2	zur Diskette schicken
0A7C E8 INX	
0A7D D0 F7 BNE \$0A76	schon 256 Zeichen
0A7F 20 CC FF JSR \$FFCC	CLRCH
0A82 A9 32 LDA #\$32	"2" für "U2" gleich Block Write
0A84 D0 6C BNE \$0AF2	Block Write Befehl an Diskette
***** Funktion 2, TASTATUR-ABFRAGE	
0A86 20 9F FF JSR \$FF9F	Tastatur abfragen
0A89 A5 C5 LDA \$C5	Matrix-Nummer des gelesenen Zeichens
0A8B 8D 05 09 STA \$0905	an Z80 übergeben
0A8E 60 RTS	
***** Funktion 3, BILDSCHIRM-AUSGABE	
0A8F A9 00 LDA #\$00	
0A91 85 D4 STA \$D4	Hochkommaflag löschen
0A93 AD 01 09 LDA \$0901	Zeichen laden
0A96 4C D2 FF JMP \$FFD2	und auf Bildschirm ausgeben
***** Funktion 4, DRUCKER-STATUS HOLEN	
0A99 A9 00 LDA #\$00	
0A9B 8D 01 09 STA \$0901	Status gleich null
0A9E 60 RTS	
***** Funktion 5, DRUCKER-AUSGABE	
0A9F AD 01 09 LDA \$0901	auszugebendes Zeichen
0AA2 C9 0A CMP #\$0A	Line Feed ?
0AA4 D0 01 BNE \$0AA7	nein ?
0AA6 60 RTS	ja, dann unterdrücken
0AA7 A2 04 LDX #\$04	logische Filenummer 4
0AA9 20 C9 FF JSR \$FFC9	CKOUT, Ausgabe auf Drucker legen
0AAC B0 09 BCS \$0AB7	Fehler ?
0AAE AD 01 09 LDA \$0901	auszugebendes Zeichen

OAB1 20 D2 FF JSR \$FFD2	zum Drucker schicken
OAB4 4C CC FF JMP \$FFCC	CLRCH, Ausgabe wieder auf default
OAB7 C9 03 CMP #\$03	File not open ?
OAB9 D0 05 BNE \$OACO	nein, anderer Fehler
OABB 20 C6 0A JSR \$OAC6	Druckdatei Öffnen
OABE 90 DF BCC \$OA9F	kein Fehler, dann Zeichen ausgeben
OACO A9 FF LDA #\$FF	Kode für Fehler
OAC2 8D 01 09 STA \$0901	an Z80 übergeben
OAC5 60 RTS	
OAC6 A0 07 LDY #\$07	Sekundäradresse 7 für VC Drucker
OAC8 20 DE 0A JSR \$OADE	Druckdatei schließen und neu öffnen
OACB AD FF 0C LDA \$OCFF	I/O-Typ (Druckertyp)
OACE 29 02 AND #\$02	
OADO F0 F3 BEQ \$OAC5	1515 Drucker ?
OAD2 A2 04 LDX #\$04	logische Filenummer
OAD4 20 C9 FF JSR \$FFC9	CKOUT, Ausgabe auf Drucker
OAD7 A9 0D LDA #\$0D	carriage return
OAD9 20 D2 FF JSR \$FFD2	zum Drucker
OADC A0 00 LDY #\$00	Sekundäradresse 0
OADE A9 04 LDA #\$04	logische Filenummer 4
OAEO 20 C3 FF JSR \$FFC3	CLOSE
OAE3 A9 04 LDA #\$04	logische Filenummer 4
OAE5 A2 04 LDX #\$04	Gerätenummer 4
OAE7 20 BA FF JSR \$FFBA	Fileparameter setzen
OAEA A9 00 LDA #\$00	kein Filename
OAEC 20 BD FF JSR \$FFBD	Parameter für Filenamen setzen
OAEF 4C C0 FF JMP \$FFC0	OPEN
*****	Block-Read/Write Befehl senden
OAF2 8D 63 0B STA \$0B63	Platz für "1" oder "2"
OAF5 AD 04 09 LDA \$0904	Diskettennummer
OAF8 20 89 0B JSR \$0B89	Zahl nach ASCII wandeln

OAFB 8D 67 OB STA \$0B67	und merken
OAFE AD 03 09 LDA \$0903	Tracknummer
OB01 20 89 OB JSR \$0B89	nach ASCII wandeln
OB04 8E 69 OB STX \$0B69	und merken
OB07 8D 6A OB STA \$0B6A	
OBOA AD 02 09 LDA \$0902	Sektornummer
OBOD 20 89 OB JSR \$0B89	nach ASCII wandeln
OB10 8E 6C OB STX \$0B6C	und merken
OB13 8D 6D OB STA \$0B6D	
OB16 A9 02 LDA #\$02	2 Versuche
OB18 8D 01 09 STA \$0901	
OB1B 20 F4 OB JSR \$0BF4	CKOUT, Ausgabe auf Kanal 15
OB1E AO OD LDY #\$0D	13 Zeichen
OB20 BD 62 OB LDA \$0B62,X	"U1: 2 0 tt ss" bzw "U2: 2 0 tt ss"
OB23 20 D2 FF JSR \$FFD2	an Diskette senden
OB26 E8 INX	
OB27 88 DEY	
OB28 DO F6 BNE \$0B20	
OB2A 20 CC FF JSR \$FFCC	CLRCH
OB2D 20 CE OB JSR \$0BCE	Fehlerkanal lesen
OB30 FO OB BEQ \$0B3D	ok
OB32 CE 01 09 DEC \$0901	bereits zweiter Versuch ?
OB35 FO OE BEQ \$0B45	ja
OB37 20 97 OB JSR \$0B97	Diskette initialisieren
OB3A 4C 1B OB JMP \$0B1B	und nochmal versuchen
OB3D A9 00 LDA #\$00	Flag für 'ok'
OB3F 8D 01 09 STA \$0901	merken
OB42 4C CC FF JMP \$FFCC	CLRCH, Ausgabe wieder auf default
OB45 A9 FF LDA \$FF	Flag für 'Fehler'
OB47 DO F6 BNE \$0B3F	weiter wie oben

***** Funktion 6, DISKETTE FORMATIEREN

*****		Initialisieren und Direktzugriffdatei öffnen
OB97 A9 OF	LDA #\$0F	logische Filenummer
OB99 20 C3 FF JSR \$FFC3		CLOSE 15
OB9C A9 OF	LDA #\$0F	logische Filenummer 15
OB9E A2 08	LDX #\$08	Gerätenummer 8
OBA0 A0 OF	LDY #\$0F	Sekundäradresse 15
OBA2 20 BA FF JSR \$FFBA		Fileparameter setzen
OBA5 A9 01	LDA #\$01	Länge des Dateinamens gleich eins
OBA7 A2 88	LDX #\$88	
OBA9 A0 0B	LDY #\$0B	Zeiger auf "I"
OBAB 20 BD FF JSR \$FFBD		Parameter für Filenamen setzen
OBAE 20 C0 FF JSR \$FFC0		OPEN 15,8,15,"I"
OBB1 A9 02	LDA #\$02	logische Filenummer 2
OBB3 20 C3 FF JSR \$FFC3		CLOSE 2
OBB6 A9 02	LDA #\$02	logische Filenummer 2
OBB8 A2 08	LDX #\$08	Gerätenummer 8
OBBA A0 02	LDY #\$02	Sekundäradresse 8
OBBC 20 BA FF JSR \$FFBA		Fileparameter setzen
OBBF A9 01	LDA #\$01	Länge des Dateinamens gleich eins
OBC1 A2 6F	LDX #\$6F	
OBC3 A0 0B	LDY #\$0B	Zeiger auf "#"
OBC5 20 BD FF JSR \$FFBD		Parameter für Filenamen setzen
OBC8 4C C0 FF JMP \$FFC0		OPEN 2,8,2,"#"
OBCB 20 97 0B JSR \$0B97		Diskette initialisieren
*****		Fehlerkanal lesen
OBCE A2 OF	LDX #\$0F	logische Filenummer 15
OBDO 20 C6 FF JSR \$FFC6		CHKIN
OBDB B0 F6	BCS \$0BCB	Fehler, dann Diskette erst initialisieren
OBDB 20 CF FF JSR \$FFCF		Zeichen vom Fehlerkanal holen
OBDB C9 30	CMP #\$30	mit "0" vergleichen
OBDA 60	RTS	

OBDB 20 B1 OB JSR \$OBB1	Direktzugriffskanal neu öffnen
***** Eingabe auf Direktzugriffskanal setzen	
OBDE A2 02 LDX #\$02	logische Filenummer 2
OBEO 20 C6 FF JSR \$FFC6	CHKIN
OBE3 B0 F6 BCS \$OBDB	Fehler, dann neu öffnen
OBE5 60 RTS	
OBEB 20 B1 OB JSR \$OBB1	Direktzugriffskanal neu öffnen
***** Ausgabe auf Direktzugriffskanal setzen	
OBEB 20 C9 FF JSR \$FFC9	CKOUT
OBEE B0 F6 BCS \$OBE6	Fehler, dann Kanal neu öffnen
OBFO 60 RTS	
OBF1 20 97 OB JSR \$OB97	Diskette initialisieren
***** Ausgabe auf Kommandokanal setzen	
OBF4 A2 0F LDX #\$0F	logische Filenummer 15
OBF6 20 C9 FF JSR \$FFC9	CKOUT
OBF9 B0 F6 BCS \$OBF1	Fehler, dann Diskette erst initialisieren
OBFB A2 00 LDX #\$00	Zeiger auf Null
OBFD 60 RTS	

***** Z80-Adressen der Texte der Funktionstasten

OC00 10 FC 20 FC 30 FC 40 FC

OC00 50 FC 60 FC 70 FC 80 FC

***** Belegung der Funktionstasten

OC10 44 49 52 0D 00 00 00 00 00 00 00 00 00 00 DIR

OC20 44 49 52 20 42 3A 0D 00 00 00 00 00 00 00 DIR B:

OC30 53 54 41 54 20 2A 2E 2A 0D 00 00 00 00 00 00 STAT *.*

OC40 53 54 41 54 20 42 3A 2A 2E 2A 0D 00 00 00 00 00 STAT B:*,*

OC50 43 4F 50 59 0D 00 00 00 00 00 00 00 00 00 00 COPY

OC60 43 4F 4E 46 49 47 0D 00 00 00 00 00 00 00 00 CONFIG

OC70 44 44 54 0D 00 00 00 00 00 00 00 00 00 00 00 DDT

OC80 44 44 54 20 00 00 00 00 00 00 00 00 00 00 00 DDT

***** Tastaturkode Tabelle

	Commodore-	normal	Shift
	Taste	C=	CTRL
OD00 08 08 08 18	DEL	BS	CAN
OD04 0D 0D 0D 0D	CR	CR	
OD08 1C 1C 1D 1D	Cursor right	FS	GS
ODOC 86 86 87 87	F7, F8	F7	F8
OD10 80 80 81 81	F1, F2	F1	F2
OD14 82 82 83 83	F3, F4	F3	F4
OD18 84 84 85 85	F5, F6	F5	F6
OD1C 1E 1E 1F 1F	Cursor down	RS	US
OD20 33 33 23 33	3 #	3 3 #	3
OD24 77 57 57 17	W	w W W	^W
OD28 61 41 41 01	A	a A A	^A
OD2C 34 34 24 34	4 \$	4 4 \$	4
OD30 7A 5A 5A 1A	Z	z Z Z	^Z
OD34 73 53 53 13	S	s S S	^S
OD38 65 45 45 05	E	e E E	^E
OD3C 00 00 00 00	Shift	NUL	
OD40 35 35 25 35	5 %	5 5 %	5

OD44 72 52 52 12	R	r R R ^R
OD48 64 44 44 04	D	d D D ^D
OD4C 36 36 26 36	6 &	6 6 & 6
OD50 63 43 43 03	C	c C C ^C
OD54 66 46 46 06	F	f F F ^F
OD58 74 54 54 14	T	t T T ^T
OD5C 78 58 58 18	X	x X X ^X
OD60 37 37 27 37	7 ^	7 7 ^ 7
OD64 79 59 59 19	Y	y Y Y ^Y
OD68 67 47 47 07	G	g G G ^G
OD6C 38 38 28 7B	8 (	8 8 ((
OD70 62 42 42 02	B	b B B ^B
OD74 68 48 48 08	H	h H H ^H
OD78 75 55 55 15	U	u U U ^U
OD7C 76 56 56 16	V	v V V ^V
OD80 39 39 29 7D	9)	9 9) }
OD84 69 49 49 09	I	i I I ^I
OD88 6A 4A 4A 0A	J	j J J ^J
OD8C 30 30 30 00	O	O O O NUL
OD90 6D 4D 4D 0D	M	m M M ^M
OD94 6B 4B 4B 0B	K	k K K ^K
OD98 6F 4F 4F 0F	O	o O O ^O
OD9C 6E 4E 4E 0E	N	n N N ^N
ODAO 2B 2B 2B 2B	+	+ + + +
ODAA 70 50 50 10	P	p P P ^P
ODAA 6C 4C 4C 0C	L	l L L ^L
ODAC 2D 2D 2D 2D	-	- - - -
ODBO 2E 2E 3E 7D	. >	. . > }
ODB4 3A 3A 5B 7B	: [	: : [(
ODB8 40 40 40 60	@	@ @ @ ^
ODBC 2C 2C 3C 7B	, <	, , < }
ODCO 5C 5C 5C 7C	\	\ \ \
ODCA 2A 2A 2A 2A	*	* * * *
ODC8 3B 3B 5D 7D	;]	; ;] }

ODCC 1B 1B 1B 7F	Home	ESC	DEL
ODD0 00 00 00 00	Shift	NUL	
ODD4 3D 3D 3D 3D	=	= = = =	
ODD8 5E 5E 5E 7E	^	^ ^ ^ ^	
ODDC 2F 2F 3F 5C	/ ?	/ / ? \	
ODE0 31 31 21 31	1 !	1 1 ! 1	
ODE4 5F 5F 5F 5F	Pfeil links	- - - -	
ODE8 00 00 00 00	CTRL	NUL	
ODEC 32 32 22 32	2 "	2 2 " 2	
ODF0 20 20 20 20	Leertaste	Leertaste	
ODF4 00 00 00 00	Commodore	NUL	
ODF8 71 51 51 11	Q	q Q Q ^Q	
ODFC 03 03 03 03	STOP	^C ^C ^C ^C	

6.7.3 Z80-BOOT-ROUTINE

```
;      Z80 BOOT-Routine für den Commodore 64
;
;      COPYRIGHT (C) 1982
;      COMMODORE INTERNATIONAL
;
;      Diese Routine wird von Track 1, Sektor 5 der
;      Commodore 64 CP/M Diskette von einer Routine
;      im BIOS65 geladen.
;
;      Die Ladeadresse ist 0000H (aus der Sicht des Z80
;      Prozessors). Wenn der Z80 eingeschaltet wird, lädt
;      diese Routine das Z80 BIOS sowie den CCP und das
;      BIOS ins RAM und springt dort hin.
;
3400 =    CCP      EQU      3400H      ;Offset für 20K CP/M
; CCP      EQU      0000H      ;für BOOT0.HEX
; CCP      EQU      0100H      ;für BOOT1.HEX
001C =    NSECTS   EQU      1CH       ;Anzahl der zu ladenden Sektoren
F903 =    TRACK    EQU      0F903H    ;Register für Tracknummer
F902 =    SECTOR   EQU      0F902H    ;Register für Sektornummer
F904 =    DISKNO   EQU      0F904H    ;Register für Diskettennummer
FCFF =    IOTYPE   EQU      0FCFFH    ;IO Konfigurationsbyte im BIOS65
4A33 =    KYBDMD   EQU      CCP+1633H ;Flag für Großbuchstabenmodus
0000 =    VICRD    EQU      0         ;Befehlsnummer für Sektor lesen
F900 =    CMD      EQU      0F900H    ;Befehlsregister
0001 =    OFF      EQU      01H       ;Wert zum Ausschalten des Z80
CE00 =    MODESW   EQU      0CE00H    ;Adresse zum Ausschalten des Z80
F901 =    DATA    EQU      0F901H    ;Übergaberegister für Daten
F800 =    BUFFER   EQU      0F800H    ;Diskettenpuffer
4A00 =    BOOT     EQU      CCP+1600H ;BOOT-Adresse des BIOS
;
```

```

0000                                ORG      0000H    ;Z80 Reset Adresse

;

0000 00                            NOP                ;wird von der Hardware benötigt
0001 110034                        LXI      D,CCP      ;erste Ladeadresse
0004 3E00                          MVI      A,0
0006 3204F9                        STA      DISKNO     ;Laden von Laufwerk A
0009 2601                          MVI      H,1       ;Start bei Track 1, Sektor 6
000B 2E06                          MVI      L,6
000D 7C          LOAD1             MOV      A,H
000E 3203F9                        STA      TRACK
0011 7D                            MOV      A,L
0012 3202F9                        STA      SECTOR
0015 3E00                          MVI      A,VICRD   ;Sektor-Lese-Befehl
0017 3200F9                        STA      CMD
001A 3E01                          MVI      A,OFF
001C 3200CE                        STA      MODESW    ;Z80 abschalten
001F 00                            NOP
0020 3A01F9                        LDA      DATA     ;Übertragung in Ordnung ?
0023 B7                            ORA      A
0024 C20D00                        JNZ      LOAD1     ;nochmal falls nicht

;

;      Ausgabe von '*' für jeden geladenen Sektor

;

0027 3E2A                          MVI      A,'*'
0029 3201F9                        STA      DATA
002C 3E03                          MVI      A,3       ;Ausgabe auf Bildschirm
002E 3200F9                        STA      CMD
0031 3E01                          MVI      A,OFF
0033 3200CE                        STA      MODESW    ;an 6510 übergeben
0036 00                            NOP

;

;      gelesenen Sektor in Speicher übertragen

;

0037 0100F8                        LXI      B,BUFFER ;BC zeigt auf Diskettenpuffer

```

```

003A 0A      LOAD2: LDAX   B      ;Byte aus Puffer
003B 12              STAX   D      ;in Speicher übertragen
003C 0C              INR     C
003D 1C              INR     E      ;Zeiger erhöhen
003E C23A00      JNZ     LOAD2

;
;      Track/Sektornummer erhöhen
;

0041 14              INR     D
0042 2C              INR     L      ;Sektornummer erhöhen
0043 7D              MOV     A,L    ;und in Akku

;
;      auf Ende des Tracks prüfen
;

0044 FE11              CPI     17    ;Sektor 17 schon erreicht ?
0046 DA4C00      JC      LOAD3    ;nein
0049 24              INR     H      ;Tracknummer erhöhen
004A 2E00              MVI     L,0    ;Sektornummer auf Null
004C 7C      LOAD3: MOV     A,H      ;Tracknummer in Akku

;
;      Fertig falls H auf Track 3 zeigt
;

004D FE03              CPI     3
004F C20D00      JNZ     LOAD1    ;weiteren Sektor lesen

;

0052 3AFFFC      LDA     IOTYPE ;Groß/Kleinbuchstaben-Modus
0055 E620              ANI     20H
0057 C25F00      JNZ     LOAD4
005A 3E01              MVI     A,1
005C 32334A      STA     KYBDMD ;Flag im BIOS setzen

;
LOAD4: ;Sprung zum BOOT-Teil des BIOS
005F C3004A      JMP     BOOT

```

6.7.4 BIOS FÜR CP/M 2.2 AUF COMMODORE 64

```
;
;   COPYRIGHT (C) 1982
;   COMMODORE INTERNATIONAL
;
;   Diese Version hat die folgenden Eigenschaften:
;
;   1. Das BIOS ist für ein 52K RAM System
;       mit I/O Treibern über BIOS65 aufgesetzt
;
;   2. Disketten Tabellen und Vektoren für
;       2 Laufwerke sind enthalten
;
;   3. Das Intel I/O Byte ist nicht implementiert
;
;   4. Punch und Reader Routinen sind noch
;       nicht implementiert
;
;   5. Tastatur und Funktionstastenbelegung
;       sind im BIOS65 enthalten
;
;   6. Der Commodore 64 kann ein 20K bis 48K CP/M
;       unterstützen (44K mit IEEE-Karte)
;
;   7. Mit der VC-1541 wird ein virtuelles
;       Laufwerk B unterstützt
;
;   8. Mit IEC-Bus wird ein reeles Laufwerk B
;       unterstützt
;
;
0000 =   BASE   EQU   0000H   ;Beginn des adressierbaren RAMs
```

```

;
002C =      MSIZE  EQU    44      ;Speichergröße des CP/M in Kilobyte
;
;      "BIAS" ist der Adressoffset von 3400H für Speicher
;      mit mehr als 20 K (im Text als "B" bezeichnet)
;
6000 =      BIAS   EQU    (MSIZE-20)*1024
;
;      Hinweis : Zur Erzeugung von MOVCPM müssen
;      folgende Zuweisungen benutzt werden:
;
;CCP  EQU    0000H      ;für BIOS0.HEX
;CCP  EQU    0100H      ;für BIOS1.HEX
;
9400 =      CCP    EQU    3400H+BIAS      ;Basis des CCP
9C06 =      BDOS   EQU    CCP+806H        ;Basis des BDOS
AA00 =      BIOS   EQU    CCP+1600H       ;Basis des BIOS
0004 =      CDISK  EQU    BASE+0004H      ;momentane Diskettennummer
;
;      0..15 gleich A..P
0003 =      IOBYTE EQU    BASE+0003H      ;INTEL I/O Byte
0000 =      TRANS  EQU    0000H          ;0 bedeutet keine Übersetzung
0005 =      ENTRY  EQU    0005H          ;BDOS Einsprungvektor
;
;      Die folgenden Zuweisungen definieren den gemeinsamen
;      Speicherbereich zur Übergabe von Daten zu und von den
;      6510 I/O-Routinen
;
F800 =      HSTBUF  EQU    0F800H         ;256 Byte Diskettenpuffer
F900 =      CMD     EQU    0F900H         ;Befehlsregister
F901 =      DATA   EQU    0F901H         ;Datenregister
F902 =      SECTOR  EQU    0F902H         ;Sektorregister
F903 =      TRACK   EQU    0F903H         ;Trackregister
F904 =      DISKNO  EQU    0F904H         ;Register für Laufwerksnummer
F905 =      KYCHAR  EQU    0F905H         ;Register für Tastaturcode

```



```

FCFF =      IOTYPE EQU      OFCFFH      ;IO Konfigurationsbyte
;
;      Der Z80 schaltet sich durch Einschreiben von
;      'OFF' in die Speicherzelle 'MODESW' selbst aus
;
0001 =      OFF     EQU      1
CE00 =      MODESW  EQU      0CE00H
;
;      Es folgen die Befehle für die 6510 I/O-Routinen
;
0000 =      VICRD   EQU      0      ;Lesen des spezifizierten Sektors
0001 =      VICWR   EQU      1      ;Schreiben des spezifizierten Sektors
0002 =      VICIN   EQU      2      ;Tastaturabfrage
0003 =      VICOUT  EQU      3      ;Ausgabe auf den Bildschirm
0004 =      VICPST  EQU      4      ;Druckerstatus holen
0005 =      VICPRT  EQU      5      ;Ausgabe zum Drucker
0006 =      VICFMT  EQU      6      ;Diskette formatieren
0007 =      AUX1    EQU      7      ;Sprung nach 6510-Adresse $E00
0008 =      AUX2    EQU      8      ;Sprung nach 6510-Adresse $F00
0009 =      INDIR   EQU      9      ;Indirekter Sprung über ($906)
;
;
AA00          ORG      BIOS      ;Start dieses Programms
0016 =      NSECTS  EQU      ($-CCP)/256      ;Sektorzähler für Warmstart
;
;      Sprungvektoren auf die einzelnen Routinen
AA00 C36CAA      JMP      BOOT      ;Kaltstart
AA03 C31DAB      WBOOTE: JMP      WBOOT      ;Warmstart
AA06 C39EAB      JMP      CONST      ;Konsolstatus
AA09 C30AAC      JMP      CONIN      ;Zeichen von Konsole holen
AA0C C383AC      JMP      CONOUT     ;Zeichen auf Konsole ausgeben
AA0F C3C8AC      JMP      LIST      ;Zeichen auf LIST-Einheit ausgeben
AA12 C316AD      JMP      PUNCH      ;Zeichen auf PUNCH-Einheit ausgeben
AA15 C319AD      JMP      READER      ;Zeichen von READER holen

```

AA18 C31EAD	JMP	HOME	;Lesekopf in HOME-Position
AA1B C329AD	JMP	SELDISK	;Diskette selektieren
AA1E C33DAD	JMP	SETTRK	;Tracknummer setzen
AA21 C343AD	JMP	SETSEC	;Sektornummer setzen
AA24 C348AD	JMP	SETDMA	;DMA-Adresse setzen
AA27 C351AD	JMP	READ	;Diskette lesen
AA2A C365AD	JMP	WRITE	;Diskette schreiben
AA2D C3EBAC	JMP	LISTST	;LIST-Status holen
AA30 C34EAD	JMP	SECTRAN	;Sektornummer Translate
;			
AA33 00	KYBDMD: DB	OOH	;Flag für Großbuchstabenmodus
;			
; Feste Tabelle für zwei Laufwerke			
; Disketten Parameter Header für Diskette 0			
AA34 00000000	DPBASE: DW	TRANS,0000H	
AA38 00000000	DW	0000H,0000H	
AA3C 1FAF54AA	DW	DIRBF,DPBLK	
AA40 DDAF9FAF	DW	CHK00,ALLO0	
; Disketten Parameter Header für Diskette 1			
AA44 00000000	DW	TRANS,0000H	
AA48 00000000	DW	0000H,0000H	
AA4C 1FAF54AA	DW	DIRBF,DPBLK	
AA50 EDAFBEAF	DW	CHK01,ALLO1	
;			
;			
DPBLK: ;Disketten Parameter Block, gemeinsam für alle Disketten			
AA54 2200	DW	34	;Sektoren pro Track
AA56 03	DB	3	;Blockverschiebe-Faktor
AA57 07	DB	7	;Block Maske
AA58 00	DB	0	;Null Maske
AA59 8700	DW	135	;Diskettenkapazität - 1
AA5B 3F00	DW	63	;max. Directoryeinträge
AA5D C0	DB	192	;Allocation 0
AA5E 00	DB	0	;Allocation 1

```

AA5F 1000      DW      16      ;überprüfte Einträge
AA61 0200      DW      2      ;Track Offset

;
;      Ende der festen Tabelle
;
;      Der folgende Speicherbereich wird beim Booten
;      initialisiert
;

AA63 40      LASTKY: DB      40H      ;Vektor der letzten gedrückten Taste
AA64 00      TOGGLE: DB      00H      ;Verwaltung für Großbuchstabenmodus
AA65 00      CSTAT:  DB      00H      ;Flag für Zeichen bereit
AA66 0000     MSGPTR: DW      0000H    ;Zeiger auf Texte der Funktionstasten
AA68 00FD     TBLPTR: DW      0FD00H   ;Tastaturdekodiertabelle
AA6A 00FC     MSGTBL: DW      0FC00H   ;Vektor für Funktionstasten

;
;      Sonstige Tastaturdefinitionen
;

F28D =      SHFTST EQU      0F28DH   ;Control, Commodore und Shift-Taste
FOCC =      FLASH  EQU      0FOCCH   ;Cursor einschalten
FOCF =      CURSOR EQU      0FOCFH   ;Cursor Zeichen

;
;      Individuelle Routinen für die einzelnen Funktionen
BOOT:

AA6C 3E20      MVI      A,20H      ;ASCII Leerzeichen
AA6E 32CFF0     STA      CURSOR     ;als Cursorzeichen
AA71 AF        XRA      A          ;Akku löschen
AA72 320300     STA      IOBYTE     ;I/O Byte löschen
AA75 320400     STA      CDISK      ;Diskette null auswählen
AA78 321EAF     STA      CURDSK     ;Zeiger auf virtuelle Diskette
AA7B 3210AF     STA      HSTACT     ;Hostpuffer inaktiv
AA7E 3212AF     STA      UNACNT     ;Zähler für Unallocated löschen
AA81 3EC3       MVI      A,0C3H     ;C3 ist der Opcode für JUMP
AA83 320000     STA      0+BASE     ;Jump nach WBOOT
AA86 2103AA     LXI      H,WBOOTE   ;WBOOT Einsprungadresse

```

AA89 220100	SHLD	1+BASE	;setzen
;			
AA8C 320500	STA	5+BASE	;Jump zum BDOS
AA8F 21069C	LXI	H,BDOS	;BDOS Einsprungadresse
AA92 220600	SHLD	6+BASE	;setzen
;			
AA95 018000	LXI	B,80H+BASE	;Default DMA Adresse
AA98 CD48AD	CALL	SETDMA	
;			
AA9B 11A6AA	LXI	D,SIGNON	;DE zeigt auf Einschaltmeldung
AA9E OE09	MVI	C,9	;Stringausgabe Funktion
AAA0 CD0500	CALL	ENTRY	;des BDOS
AAA3 C38DAB	JMP	GOCPM1	;fertig für CCP
;			
AAA6 OCOA	SIGNON: DB	OCH, OAH	;Bildschirm löschen
AAA8 202020243	DB	' COMMODORE 64 20k CP/M vers 2.2 '	
AACC ODOAOA	DB	ODH, OAH, OAH	
AACF 2020436F70	DB	' Copyright (C) 1979, Digital Research ', ODH, OAH	
AAF7 2020202020	DB	' Copyright (C) 1982, Commodore ', ODH, OAH	
AB1B OA24	DB	OAH, '\$'	;Kennzeichen für Stringende
;			
;			
WBOOT:			
AB1D 318000	LXI	SP,80H+BASE	;Der Speicher unterhalb des Puffers
			;wird als Stack benutzt
AB20 OE00	MVI	C,0	;Diskette 0 wählen
AB22 CD29AD	CALL	SELDSK	
AB25 AF	XRA	A	;Laufwerk A
AB26 3204F9	STA	DISKNO	;auswählen
AB29 CDA5AE	CALL	CHGDSK	;falls noch nicht geschehen
AB2C CD1EAD	CALL	HOME	;Lesekopf auf Track 0
AB2F 3E0D	MVI	A,ODH	;Carriage return
AB31 CDCOAC	CALL	COUT5	;ausgeben

```

;
AB34 110094      LXI      D,CCP      ;Laden starten
AB37 0616        MVI      B,NSECTS
AB39 2601        MVI      H,1        ;Track Nummer
AB3B 2E06        MVI      L,6        ;Sektor Nummer
AB3D 7C          LOAD1: MOV      A,H
AB3E 3203F9      STA      TRACK
AB41 7D          MOV      A,L
AB42 3202F9      STA      SECTOR
AB45 3E00        MVI      A,VICRD ;Befehl zum Disketten lesen
AB47 CD94AB      CALL     IO6510
;
AB4A 3A01F9      LDA      DATA
AB4D B7          ORA      A
AB4E C23DAB      JNZ      LOAD1      ;wenn nicht null dann Fehler
AB51 E5          PUSH     H
AB52 C5          PUSH     B
AB53 010001      LXI      B,256
AB56 2100F8      LXI      H,HSTBUF ;Diskettenpuffer
AB59 ED          DB       OEDH      ;LDIR Befehl des Z80
AB5A B0          DB       OBOH
AB5B 0E2A        MVI      C,'*'      ;nach jedem Sektor ein Stern ausgeben
AB5D CD83AC      CALL     CONOUT
AB60 C1          POP      B
AB61 E1          POP      H
AB62 05          DCR      B          ;Sektorzähler erniedrigen
AB63 CA73AB      JZ       GOCPM
AB66 2C          INR      L          ;nächsten Sektor
AB67 7D          MOV      A,L
AB68 FE11        CPI      17
AB6A DA3DAB      JC       LOAD1
AB6D 24          INR      H
AB6E 2E00        MVI      L,0
AB70 C33DAB      JMP      LOAD1

```

```

;      Ende der Ladeoperation; Parameter setzen und Sprung ins CP/M
GOCPM:

AB73 3EC3      MVI    A,0C3H  ;C3 ist der JUMP-Befehl
AB75 320000     STA    0+BASE  ;Jump zum WBOOT
AB78 2103AA     LXI    H,WBOOT  ;WBOOT Einsprungadresse
AB7B 220100     SHLD   1+BASE  ;Adressfeld für Jump 0 setzen
;
AB7E 320500     STA    5+BASE  ;Jump zum BDOS
AB81 21069C     LXI    H,BDOS  ;BDOS Einsprungadresse
AB84 220600     SHLD   6+BASE  ;Adressfeld für Jump 5 zum BDOS
;
AB87 018000     LXI    B,80H+BASE ;Default DMA Adresse 80H
AB8A CD48AD     CALL   SETDMA
;
;
AB8D 3A0400     GOCPM1: LDA    CDISK  ;Diskettennummer holen
AB90 4F         MOV     C,A      ;zum CCP für weitere
AB91 C30094     JMP     CCP      ;Verarbeitung schicken
;
;
;      Hauptroutine zur Übertragung der Programmausführung
;      an den 6510
;
AB94 3200F9     IO6510: STA    CMD    ;Akkuinhalt in Befehlsregister
AB97 3E01      MVI     A,OFF
AB99 3200CE     STA    MODESW  ;Z80 ausschalten
AB9C 00        NOP             ;von der Hardware benötigt
AB9D C9        RET
;
;
;
CONST: ;Konsolstatus, FFH wenn Zeichen bereit, ansonsten OOH
AB9E 2A66AA     LHLD   MSGPTR  ;Funktionstasten-Modus ?
ABA1 7C        MOV     A,H

```

ABA2 B5	ORA	L	
ABA3 3EFF	MVI	A,OFFH	;Flag für Daten bereit
ABA5 C0	RNZ		;Return falls MSGPTR<>0
;			
ABA6 3A65AA	LDA	CSTAT	;schon ein Zeichen ?
ABA9 A7	ANA	A	
ABAA C0	RNZ		;ja falls nicht 0
;			
ABAB 3E02	MVI	A,VICIN	;Befehl für Zeichen holen
ABAD CD94AB	CALL	I06510	
;			
ABBO 3A8DF2	LDA	SHFTST	;Status der Controлтasten holen
ABB3 E602	ANI	02H	;prüfen auf Commodore-Taste
ABB5 CAC9AB	JZ	CONST0	;Sprung falls nicht gedrückt
;			
ABB8 3A64AA	LDA	TOGGLE	;War Taste bereits gedrückt ?
ABBB A7	ANA	A	
ABBC C2C9AB	JNZ	CONST0	;nein, warten bis losgelassen
;			
ABBF 3A33AA	LDA	KYBDMD	;Flag für Großbuchstabenmodus holen
ABC2 EE01	XRI	01H	;Modusbit umdrehen
ABC4 3233AA	STA	KYBDMD	
ABC7 3E01	MVI	A,1	
ABC9 3264AA	CONST0: STA	TOGGLE	
;			
ABCC 3A05F9	LDA	KYCHAR	;Zeichen holen
ABCF FE3A	CPI	3AH	;ungültiges Kontrollzeichen
ABD1 CAEOAB	JZ	CONST1	
;			
ABD4 FE3D	CPI	3DH	;ungültiges Kontrollzeichen
ABD6 CAEOAB	JZ	CONST1	
;			
ABD9 2163AA	LXI	H,LASTKY	;mit letzter Taste vergleichen
ABDC BE	CMP	M	;Daten überprüfen

```

ABDD C2E5AB          JNZ      CONST2 ;falls unterschiedlich dann neue Taste
;
ABE0 AF             CONST1: XRA      A      ;Flag für keine Daten bereit
ABE1 3265AA          STA      CSTAT   ;für später merken
ABE4 C9              RET
;
ABE5 F5             CONST2: PUSH     PSW
ABE6 01F401          LXI      B,500
ABE9 0B             CONST3: DCX      B      ;Verzögerung für Tastaturentprellung
ABEA 79             MOV      A,C
ABEB B0             ORA      B
ABEC C2E9AB          JNZ      CONST3
;
ABEF 3E02           MVI      A,VICIN ;noch ein Zeichen holen
ABF1 CD94AB          CALL     IO6510
;
ABF4 F1             POP      PSW
ABF5 2105F9          LXI      H,KYCHAR
ABF8 BE             CMP      M
ABF9 C2EOAB          JNZ      CONST1 ;falls nicht 0 hat Taste nicht geprellt
;
ABFC 3263AA          STA      LASTKY ;letzte Taste updaten
ABFF FE40           CPI      40H      ;40H bedeutet keine Taste gedrückt
AC01 CAEOAB          JZ       CONST1
;
AC04 3EFF           MVI      A,OFFH  ;Flag für Zeichen bereit
AC06 3265AA          STA      CSTAT   ;für später merken
AC09 C9             RET
;
CONIN: ;Zeichen von Konsole in Register A holen
AC0A 3E00           MVI      A,0      ;Cursor einschalten
AC0C 32CCF0          STA      FLASH
;
AC0F 2A66AA          LHLD     MSGPTR  ;Funktionstastenmodus ?

```


AC12 7C	MOV	A,H	
AC13 B5	ORA	L	
AC14 C25FAC	JNZ	CONIN5	
	;		
	;		
AC17 CD9EAB	CONIN1: CALL	CONST	;Konsolstatus prüfen
AC1A B7	ORA	A	
AC1B CA17AC	JZ	CONIN1	;auf neues Zeichen warten
	;		
AC1E AF	XRA	A	
AC1F 3265AA	STA	CSTAT	;Status löschen
AC22 3A33AA	CONIN2: LDA	KYBDMD	;ohne Shift=0, Groß=1
AC25 47	MOV	B,A	
AC26 3A8DF2	LDA	SHFTST	;Status für Shift/Ctrl holen
AC29 E601	ANI	01H	;ist die Shifttaste gedrückt ?
AC2B CA30AC	JZ	CONIN3	;Sprung falls nicht
	;		
AC2E 0602	MVI	B,2	;Shift=2
AC30 3A8DF2	CONIN3: LDA	SHFTST	;Status holen
AC33 E604	ANI	04H	;ist die Control-Taste gedrückt ?
AC35 CA3AAC	JZ	CONIN4	;Sprung falls nicht
	;		
AC38 0603	MVI	B,3	;Control=3
AC3A 3A63AA	CONIN4: LDA	LASTKY	;Tastennummer holen
AC3D 87	ADD	A	;*2
AC3E 87	ADD	A	;*4
AC3F 80	ADD	B	;Nummer addieren
AC40 2A68AA	LHLD	TBLPTR	;Adresse der Tastaturtabelle
AC43 85	ADD	L	;Zeiger in Tabelle
AC44 6F	MOV	L,A	
AC45 3E00	MVI	A,0	
AC47 8C	ADC	H	
AC48 67	MOV	H,A	
AC49 7E	MOV	A,M	;Zeichen aus Tabelle holen

AC4A FE80	CPI	80H	; Falls >7FH dann Funktionstaste
AC4C DA71AC	JC	CONIN7	; Sprung falls ASCII-Zeichen
;			
AC4F 2A6AAA	LHLD	MSGTBL	; Start der Texttabelle
AC52 E67F	ANI	7FH	; Bit 7 löschen
AC54 87	ADD	A	; *2
AC55 85	ADD	L	; Zeiger in Tabelle
AC56 6F	MOV	L,A	
AC57 3E00	MVI	A,0	
AC59 8C	ADC	H	
AC5A 67	MOV	H,A	
AC5B 7E	MOV	A,M	; Lo Byte
AC5C 23	INX	H	
AC5D 66	MOV	H,M	; Hi Byte
AC5E 6F	MOV	L,A	
AC5F 46	CONIN5: MOV	B,M	; Zeichen holen
AC60 23	INX	H	; nächstes Zeichen prüfen
AC61 7E	MOV	A,M	
AC62 A7	ANA	A	
AC63 C269AC	JNZ	CONIN6	; falls 0 ist in B das letzte Zeichen
;			
AC66 210000	LXI	H,0000H	; Ende des Text-Modus
AC69 2266AA	CONIN6: SHLD	MSGPTR	; Zeiger auf Text merken
AC6C 78	MOV	A,B	; Zeichen prüfen
AC6D A7	ANA	A	; ist das erste bereits 0 ?
AC6E CA17AC	JZ	CONIN1	; falls 0 dann kein Zeichen
;			
AC71 F5	CONIN7: PUSH	PSW	; Zeichen merken
AC72 3E01	MVI	A,1	
AC74 32CCF0	STA	FLASH	; Cursor ausschalten
AC77 0E20	MVI	C,' '	; Leerzeichen
AC79 CD83AC	CALL	CONOUT	; ausgeben
AC7C 3E9D	MVI	A,9DH	; Cursor links
AC7E CDC0AC	CALL	COUT5	; Filter überspringen

AC81 F1	POP	PSW	; Zeichen zurückholen
AC82 C9	RET		; fertig
;			
CONOUT: ; Zeichen aus Register C auf Konsole ausgeben			
AC83 3AFFFC	LDA	IOTYPE	; Konfigurationsbyte
AC86 E610	ANI	10H	; Bit 4 = 1 bedeutet Filter ignorieren
AC88 79	MOV	A,C	; Zeichen in Akku
AC89 C2COAC	JNZ	COUT5	; wie erhalten ausgeben
;			
AC8C CDF4AC	CALL	SWAP	; Groß- und Kleinbuchstaben vertauschen
AC8F FEOC	CPI	OCH	; ASCII-Zeichen für Bildschirm löschen?
AC91 C299AC	JNZ	COUT1	; Sprung falls nicht
;			
AC94 3E93	MVI	A,93H	; Commodorekode für Bildschirm löschen
AC96 C3COAC	JMP	COUT5	
;			
AC99 FEO8	COUT1: CPI	08H	; ASCII Backspace (Rückschritt)
AC9B C2A3AC	JNZ	COUT2	; Sprung falls nicht
;			
AC9E 3E14	MVI	A,14H	; Commodorekode für Zeichen löschen
ACA0 C3COAC	JMP	COUT5	
;			
ACA3 FEOA	COUT2: CPI	0AH	; Line Feed ?
ACA5 C2ADAC	JNZ	COUT3	
;			
ACA8 3E11	MVI	A,17	; Commodorekode für Line Feed
ACAA C3COAC	JMP	COUT5	
;			
ACAD FEOD	COUT3: CPI	ODH	; Carriage Return ?
ACAF C2BAAC	JNZ	COUT4	
;			
ACB2 CDCOAC	CALL	COUT5	
ACB5 3E91	MVI	A,145	; eine Zeile hoch um Auto LF zu
ACB7 C3COAC	JMP	COUT5	; ignorieren

```

;
ACBA FE20      COUT4: CPI      20H
ACBC D8        RC              ;Return falls andres Control-Zeichen
ACBD FE80      CPI      80H
ACBF D0        RNC              ;Return falls kein ASCII-Zeichen
;
ACCO 3201F9    COUT5: STA      DATA ;Zeichen in DATA übergeben
ACC3 3E03      MVI      A,VICOUT ;Befehl für Bildschirmausgabe
ACC5 C394AB    JMP      I06510
;
LIST: ;Zeichen aus Register C zur LIST-Einheit schicken
ACC8 3AFFFC    LDA      I0TYPE ;Welcher Druckertyp ?
ACCB E604      ANI      04H ;0 falls 1515, 1 falls 4022
ACCD 79        MOV      A,C ;Zeichen in Register A
ACCE C2E3AC    JNZ      LIST2 ;überspringen falls kein Vertauschen
;
ACD1 3AFFFC    LDA      I0TYPE
ACD4 E608      ANI      08H ;welcher Typ von Vertauschen ?
ACD6 79        MOV      A,C ;Zeichen holen
ACD7 C2EOAC    JNZ      LIST1
;
ACDA CDF4AC    CALL     SWAP ;Groß- und Kleinbuchstaben vertauschen
ACDD C3E3AC    JMP      LIST2
;
ACE0 CD08AD    LIST1: CALL     SWAP2 ;Tauschroutine für 4022
ACE3 3201F9    LIST2: STA      DATA ;Zeichen übergeben
ACE6 3E05      MVI      A,VICPRT ;1515 annehmen
ACE8 C394AB    LIST3: JMP      I06510
;
LISTST: ;Liststatus (0 falls nicht bereit, 1 ist bereit)
ACEB 3E04      MVI      A,VICPST ;Befehl für Druckerstatus
ACED CD94AB    CALL     I06510
ACFO 3A01F9    LDA      DATA ;DATA ist der Status
ACF3 C9        RET

```

```

;
SWAP: ;Groß- und Kleinbuchstaben für Commodore 64 vertauschen
ACF4 FE41 CPI 41H ;kleiner als großes 'A'?
ACF6 D8 RC ;Return falls ja
;
ACF7 FE5B CPI 5BH ;Großbuchstabe ?
ACF9 DA05AD JC SWAP1 ;Sprung falls ja
;
ACFC FE61 CPI 61H ;kleiner als kleines 'A'
ACFE D8 RC ;Return falls ja
;
ACFF FE7B CPI 7BH ;Kleinbuchstabe ?
ADO1 D0 RNC ;Return falls nicht
;
ADO2 E65F ANI 5FH ;Bit 5 löschen
ADO4 C9 RET
;
ADO5 F620 SWAP1: ORI 20H ;Bit 5 setzen
ADO7 C9 RET
;
ADO8 FE41 SWAP2: CPI 41H ;CY falls kleiner als großes 'A'
ADOA D8 RC
ADOB FE60 CPI 60H ;CY falls 40H < A < 60H
ADOD D213AD JNC SWAP3
;
AD10 F680 ORI 80H
AD12 C9 RET
;
AD13 E65F SWAP3: ANI 5FH
AD15 C9 RET
;
PUNCH: ;Zeichen aus Register C zur PUNCH-Einheit
AD16 79 MOV A,C ;Zeichen in Register
AD17 00 NOP

```

```

AD18 C9          RET          ;Null Routine

;
;
READER: ;Zeichen von READER-Einheit nach Register A
AD19 3E1A        MVI         A,1AH    ;End Of File Kode (CTRL Z) zurückgeben
AD1B E67F        ANI         7FH     ;Paritybit löschen
AD1D C9          RET

;
;
;*****
;*
;*          KONSTANTEN HOSTDISK ZU CP/M          *
;*
;*****

0400 =          BLKSIZ EQU      1024    ;CP/M Allocation Größe
0100 =          HSTSIZ EQU      256    ;Host Diskette Sektorgröße
0011 =          HSTSPT EQU      17     ;Host Diskette Sektoren/Track
0002 =          HSTBLK EQU      HSTSIZ/128    ;CP/M Sektoren/Hostpuffer
0022 =          CPMSPT EQU      HSTBLK * HSTSPT ;CP/M Sektoren/Track
0001 =          SECMSK EQU      HSTBLK-1    ;Sektor Maske
0001 =          SECSHF EQU      1         ;LOG2(HSTBLK)

;
;*****
;*
;*          BDOS KONSTANTEN BEI WRITE-EINSPRUNG    *
;*
;*****

0000 =          WRALL EQU      0         ;WRITE to Allocated
0001 =          WRDIR EQU      1         ;WRITE to Directory
0002 =          WRUAL EQU      2         ;WRITE to Unallocated

;
;          HOME-Funktion der selektierten Diskette
HOME:
AD1E 3A11AF      LDA         HSTWRT      ;Prüfen auf WRITE

```

AD21 B7	ORA	A	
AD22 C228AD	JNZ	HOMED	
AD25 3210AF	STA	HSTACT	;Host-Aktiv Flag löschen
HOMED:			
AD28 C9	RET		
;			
SELDISK:			
;Diskette selektieren			
AD29 210000	LXI	H,0000H	;Fehler Returnkode
AD2C 79	MOV	A,C	;Nummer der selektierten Diskette
AD2D 3207AF	STA	SEKDSK	;Diskettennummer merken
AD30 FE02	CPI	2	;muß 0 oder 1 sein
AD32 D0	RNC		;kein Carry falls größer1
AD33 6F	MOV	L,A	;Diskettennummer nach HL
AD34 29	DAD	H	;mit 16 multiplizieren
AD35 29	DAD	H	
AD36 29	DAD	H	
AD37 29	DAD	H	
AD38 1134AA	LXI	D,DPBASE	;Basis der Parameterblocks
AD3B 19	DAD	D	;HL=DPB
AD3C C9	RET		
;			
SETTRK:			
;Tracknummer aus Register BC setzen			
AD3D 60	MOV	H,B	
AD3E 69	MOV	L,C	
AD3F 2208AF	SHLD	SEKTRK	;zu suchender Track
AD42 C9	RET		
;			
SETSEC:			
;Sektornummer aus Register C setzen			
AD43 79	MOV	A,C	
AD44 320AAF	STA	SEKSEC	;zu suchender Sektor
AD47 C9	RET		

```

;
SETDMA:
;DMA Adresse aus Register BC setzen
AD48 60      MOV     H,B
AD49 69      MOV     L,C
AD4A 221BAF  SHLD    DMAADR
AD4D C9      RET

;
SECTTRAN:
;Übersetze Sektornummer aus Register BC
AD4E 60      MOV     H,B
AD4F 69      MOV     L,C
AD50 C9      RET

;
;*****
;*
;*      DER READ EINSPRUNGUNKT FOLGT DEN      *
;*      VORHERGEHENDEN BIOS DEFINITION FÜR READ *
;*
;*
;*****
READ:
;Lese den ausgewählten CP/M Sektor
AD51 AF      XRA     A
AD52 3212AF  STA     UNACNT
AD55 3E01     MVI     A,1
AD57 3219AF  STA     READOP      ;READ Operation
AD5A 3218AF  STA     RSFLAG      ;Daten lesen
AD5D 3E02     MVI     A,WRUAL
AD5F 321AAF  STA     WRTYPE      ;behandle als Unallocated
AD62 C3DOAD  JMP     RWOPER      ;READ-Befehl ausführen

;
;*****
;*
;*      DER WRITE EINSPRUNGUNKT FOLGT DEN      *

```



```

;*      VORHERGEHENDEN BIOS DEFINITIONEN FÜR WRITE      *
;*
;*****
WRITE:
;Schreibe des ausgewählten Sektor

AD65 AF      XRA      A      ;0 in den Akku
AD66 3219AF   STA      READOP ;kein READ Befehl
AD69 79       MOV      A,C    ;WRITE Type nach C
AD6A 321AAF   STA      WRTYPE
AD6D FE02     CPI      WRUAL   ;WRITE Unallocated?
AD6F C289AD   JNZ      CHKUNA  ;Prüfe auf Unallocated

;
;      WRITE Unallocted, Paramater setzen

AD72 3E08     MVI      A,BLKSI2/128 ;nächster Unallocated Record
AD74 3212AF   STA      UNACNT
AD77 3A07AF   LDA      SEKDSK    ;Diskette
AD7A 3213AF   STA      UNADSK    ;UNADSK = SEKDSK
AD7D 2A08AF   LHLD     SEKTRK
AD80 2214AF   SHLD     UNATRK    ;UNATRK = SECTRK
AD83 3A0AAF   LDA      SEKSEC
AD86 3216AF   STA      UNASEC    ;UNASEC = SEKSEC

;
CHKUNA:
;Prüfe auf WRITE zu Unallocated Sektor

AD89 3A12AF   LDA      UNACNT    ;noch UNALLOC übrig ?
AD8C B7       ORA      A
AD8D CAC8AD   JZ       ALLOC     ;Sprung falls nicht

;
;      Weitere Unallocated Records übrig

AD90 3D       DCR      A      ;UNACNT = UNACNT-1
AD91 3212AF   STA      UNACNT
AD94 3A07AF   LDA      SEKDSK    ;gleiche Diskette ?
AD97 2113AF   LXI      H,UNADSK
AD9A BE       CMP      M      ;SEKDSK = UNADSK?

```

```

AD9B C2C8AD      JNZ      ALLOC      ;Sprung falls nicht
;
;      Gleiche Disketten
AD9E 2114AF      LXI      H,UNATRK
ADA1 CD6BAE      CALL     TRKCMP      ;SEKTRK = UNATRK?
ADA4 C2C8AD      JNZ      ALLOC      ;Sprung falls nicht
;
;      Gleiche Tracks
ADA7 3A0AAF      LDA      SEKSEC      ;gleicher Sektor ?
ADAA 2116AF      LXI      H,UNASEC
ADAD BE          CMP      M          ;SEKSEC = UNASEC?
ADAE C2C8AD      JNZ      ALLOC      ;Sprung falls nicht
;
;      Passend, nächster Sektor für zukünftigen Zugriff
ADB1 34          INR      M          ;UNASEC = UNASEC+1
ADB2 7E          MOV      A,M        ;Ende des Tracks ?
ADB3 FE22        CPI      CPMSPT     ;CP/M Sektorzähler
ADB5 DAC1AD      JC       NOOVF      ;Sprung falls kein Übertrag
;
;      Übertrag zu nächstem Track
ADB8 3600        MVI      M,0        ;UNASEC = 0
ADBA 2A14AF      LHLD     UNATRK
ADBD 23          INX      H
ADBE 2214AF      SHLD     UNATRK     ;UNATRK = UNATRK+1
;
NOOVF:
;Übereinstimmung, als unnötiges Lesen markieren
ADC1 AF          XRA      A          ;0 in Akku
ADC2 3218AF      STA      RSFLAG     ;RSFLAG = 0
ADC5 C3DOAD      JMP      RWOPER     ;für WRITE
;
ALLOC:
;kein Unallocated Record, erfordert vorheriges Lesen
ADC8 AF          XRA      A          ;0 in Akku

```

```

ADC9 3212AF      STA      UNACNT      ;UNACNT = 0
ADCC 3C          INR      A            ;1 in Akku
ADCD 3218AF      STA      RSFLAG      ;RSFLAG = 1

```

```

;
;*****
;*
;*   ES FOLGT GEMEINSAMER KODE FÜR READ UND WRITE   *
;*
;*****

```

RWOPER:

```

;Einsprung für READ und WRITE
ADDO AF          XRA      A            ;Akku löschen
ADD1 3217AF      STA      ERFLAG      ;keine Fehler (bis hierhin)
ADD4 3A0AAF      LDA      SEKSEC      ;berechne Host Sektor
ADD7 B7          ORA      A            ;Carry = 0
ADD8 1F          RAR                     ;nach rechts verschieben
ADD9 320FAF      STA      SEKHST      ;Host Sektor

```

```

;
;   Aktiver Host Sektor?
ADDC 2110AF      LXI      H,HSTACT      ;Host-Aktiv Flag
ADDF 7E          MOV      A,M
ADE0 3601        MVI      M,1          ;wird immer 1
ADE2 B7          ORA      A            ;war es schon ?
ADE3 CA0AAE      JZ       FILHST      ;Host füllen falls nicht

```

```

;
;   Hostpuffer aktiv, gleicher wie Suchpuffer ?
ADE6 3A07AF      LDA      SEKDSK
ADE9 210BAF      LXI      H,HSTDSK      ;gleiche Diskette
ADEC BE          CMP      M            ;SEKDSK = HSTDSK?
ADED C203AE      JNZ      NOMTCH

```

```

;
;   Gleiche Diskette, gleicher Track ?
ADFO 210CAF      LXI      H,HSTTRK
ADF3 CD6BAE      CALL     TRKCOMP      ;SEKTRK = HSTTRK?

```

```

ADF6 C203AE      JNZ      NOMTCH
;
;      Gleiche Diskette, gleicher Track, gleicher Puffer ?
ADF9 3A0FAF      LDA      SEKHST
ADFC 210EAF      LXI      H,HSTSEC      ;SEKHST = HSTSEC?
ADFF BE          CMP      M
AE00 CA27AE      JZ       MATCH      ;Sprung falls Übereinstimmung
;
NOMTCH:
;Richtige Diskette, aber falscher Sektor
AE03 3A11AF      LDA      HSTWRT      ;Host geschrieben ?
AE06 B7          ORA      A
AE07 C477AE      CNZ      WRHST      ;Hostpuffer löschen
;
FILHST:
;Hostpuffer füllen
AEOA 3A07AF      LDA      SEKDSK
AEOD 320BAF      STA      HSTDSK
AE10 2A08AF      LHLD     SEKTRK
AE13 220CAF      SHLD     HSTTRK
AE16 3A0FAF      LDA      SEKHST
AE19 320EAF      STA      HSTSEC
AE1C 3A18AF      LDA      RSFLAG      ;READ erforderlich ?
AE1F B7          ORA      A
AE20 C4CAAE      CNZ      RDHST      ;ja, in 1
AE23 AF          XRA      A      ;Akku löschen
AE24 3211AF      STA      HSTWRT      ;kein WRITE
;
MATCH:
;Daten in oder zum Puffer kopieren
AE27 3A0AAF      LDA      SEKSEC      ;Nummer des Maskenpuffers
AE2A E601      ANI      SECMSK      ;unterste Bits isolieren
AE2C 6F          MOV      L,A      ;für zum Verschieben
AE2D 2600      MVI      H,0      ;2 Byte Zähler

```

```

AE2F 29          DAD    H          ;sieben nach links schieben
AE30 29          DAD    H
AE31 29          DAD    H
AE32 29          DAD    H
AE33 29          DAD    H
AE34 29          DAD    H
AE35 29          DAD    H
;               HL enthält relative Hostpuffer Adresse
AE36 1100F8      LXI     D,HSTBUF
AE39 19          DAD    D          ;HL = Hostadresse
AE3A EB          XCHG
;               ;jetzt in DE
AE3B 2A1BAF      LHLD    DMAADR    ;Holen/Schreiben von CP/M Daten
AE3E OE80        MVI     C,128     ;Anzahl zu übertagen
AE40 3A19AF      LDA     READOP    ;welche Richtung
AE43 B7          ORA     A
AE44 C24DAE      JNZ     RWMOVE    ;Überspringen falls Lesen
;
;               WRITE Operation markieren und Richtung vertauschen
AE47 3E01        MVI     A,1
AE49 3211AF      STA     HSTWRT    ;HSTWRT = 1
AE4C EB          XCHG
;               ;Quelle/Ziel vertauschen
;
RWMOVE:
;               ;C enthält 128, DE ist Quelle, HL ist Ziel
AE4D 1A          LDAX    D          ;Zeichen aus Quellbereich
AE4E 13          INX     D
AE4F 77          MOV     M,A        ;in Zielbereich übertragen
AE50 23          INX     H
AE51 OD          DCR     C          ;128 mal
AE52 C24DAE      JNZ     RWMOVE
;
;               Daten sind vom/zum Hostpuffer transferiert
AE55 3A1AAF      LDA     WRTYPE    ;WRITE Type ?
AE58 FE01        CPI     WRDIR     ;ins Directory ?

```

```

AE5A 3A17AF      LDA      ERFLAG      ;falls Fehler
AE5D CO          RNZ              ;keine weitere Verarbeitung
;
;      Hostpuffer für WRITE in Directory
AE5E B7          ORA      A          ;Fehler ?
AE5F CO          RNZ              ;ja, dann fertig
AE60 AF          XRA      A          ;Akku löschen
AE61 3211AF      STA      HSTWRT     ;Puffer geschrieben
AE64 CD77AE      CALL     WRHST
AE67 3A17AF      LDA      ERFLAG
AE6A C9          RET
;
;*****
;*
;*      HILFSROUTINE FÜR 16-BIT VERGLEICH
;*
;*
;*****
TRKCOMP:
;HL = .UNATRK oder .HSTTRK, vergleiche mit SEKTRK
AE6B EB          XCHG
AE6C 2108AF      LXI      H,SEKTRK
AE6F 1A          LDAX    D          ;Lo Byte vergleichen
AE70 BE          CMP     M          ;gleich
AE71 CO          RNZ              ;Return falls nicht
;      Lo Byte gleich, teste Hi Byte
AE72 13          INX     D
AE73 23          INX     H
AE74 1A          LDAX    D
AE75 BE          CMP     M          ;setze Flags
AE76 C9          RET

```

```

;
;*****
;*
;*      WRHST schreibt physikalisch zur Hostdiskette *
;*      RDHST liest die Diskette physikalisch      *
;*
;*****
WRHST:
;HSTDSK = Hostdisketten#, HSTTRK = Hosttrack #,
;HSTSEC = Hostsector#. Schreibt "HSTSIZ" Bytes
;des HSTBUF und gibt Fehlerflag in ERFLAG zurück
;ERFLAG ungleich null falls Fehler

AE77 3E01          MVI    A,VICWR          ;Disk-Write Befehl
AE79 321DAF        WRHST0: STA    RW          ;in Befehlsregister schreiben
AE7C 3A0BAF        LDA    HSTDSK          ;Hostdiskettennummer holen
AE7F 3204F9        STA    DISKNO          ;und setzen
AE82 CDA5AE        CALL   CHGDSK          ;richtige virtuelle Diskette ?
AE85 3A0CAF        WRHST2: LDA    HSTTRK          ;Hosttracknummer holen
AE88 3C            INR    A                ;eins als Offset addieren
AE89 FE12          CPI    18              ;wir wollen Track 18 überspringen
AE8B DA8FAE        JC     WRHST3          ;Carry falls Track<18
AE8E 3C            INR    A
AE8F 3203F9        WRHST3: STA    TRACK          ;in gemeinsamen Bereich schreiben
AE92 3A0EAF        LDA    HSTSEC          ;Hostsektornummer holen
AE95 3202F9        STA    SECTOR          ;in gemeinsamen Bereich schreiben
AE98 3A1DAF        LDA    RW              ;Diskettenbefehl
AE9B CD94AB        CALL   IO6510
AE9E 3A01F9        LDA    DATA          ;Diskstatus holen
AEA1 3217AF        STA    ERFLAG          ;und als ERFLAG speichern
AEA4 C9            RET

;
AEA5 67            CHGDSK: MOV    H,A          ;Diskettennummer merken
AEA6 3AFFFC        LDA    IOTYPE          ;Bit 0=0 für virtuell
AEA9 E601          ANI    01

```

AEAB C0	RNZ		;nicht 0 falls 2 Laufwerke
AEAC 3204F9	STA	DISKNO	;Laufwerk A setzen
AEAF 7C	MOV	A,H	;Diskettennummer zurückholen
;			
AEB0 211EAF	LXI	H,CURDSK	;ist dies unsere Diskette?
AEB3 BE	CMP	M	
AEB4 C8	RZ		;Return falls ja
;			
AEB5 77	MOV	M,A	;neue Diskettennummer setzen
AEB6 C641	ADI	'A'	;ASCII-Zeichen herstellen
AEB8 32DDAE	STA	DSKMNT	;in Mitteilung einsetzen
;			
AEBB 21CFAE	LXI	H,MNTMSG	; 'Insert Disk' Mitteilung
AEBE CDFAAE	CALL	PMSG	;ausgeben
AEC1 CDOAAC	CHGD1: CALL	CONIN	;auf Return warten
AEC4 FE0D	CPI	ODH	
AEC6 C2C1AE	JNZ	CHGD1	
AEC9 C9	RET		
;			
RDHST:			
;HSTDSK = Hostdisk #, HSTTRK = Hosttrack #,			
;HSTSEC = Hostsektor #. Lese "HSHSIZ" Bytes			
;in HSTBUF und setze Fehlerflag nach ERFLAG			
AECA 3E00	MVI	A,VICRD	;Disketten READ-Befehl
AECC C379AE	JMP	WRHSTO	;Rest wie bei DISK WRITE
;			
AECF ODOA496E73MNTMSG:	DB	ODH,OAH,'Insert disk '	
AEDD 41	DSKMNT: DB	'A'	
AEDE 20696E746F	DB	' into drive 0, press return'	
AEF9 00	DB	OOH	
;			
AEFA 7E	PMSG: MOV	A,M	
AEFB A7	ANA	A	
AEFC C8	RZ		

AEFD E5	PUSH	H
AEFE 4F	MOV	C,A
AEFF CD83AC	CALL	CONOUT
AFO2 E1	POP	H
AFO3 23	INX	H
AFO4 C3FAAE	JMP	PMSG

```

;
;*****
;*
;*          NICHTINITIALISIERTE RAM DATEN-BEREICHE
;*
;*
;*****
;

```

AFO7	SEKDSK: DS	1	;SEEK Diskettennummer
AFO8	SEKTRK: DS	2	;SEEK Tracknummer
AFOA	SEKSEC: DS	1	;SEEK Sektornummer
;			
AFOB	HSTDSK: DS	1	;HOST Diskettennummer
AFOC	HSTTRK: DS	2	;HOST Tracknummer
AFOE	HSTSEC: DS	1	;HOST Sektornummer
;			
AFOF	SEKHST: DS	1	;SEEK Shift right SECSHF
AF10	HSTACT: DS	1	;HOST Aktive-Flag
AF11	HSTWRT: DS	1	;HOST Written-Flag
;			
AF12	UNACNT: DS	1	;UNALLOC Recordzähler
AF13	UNADSK: DS	1	;Letzte UNALLOC Diskette
AF14	UNATRK: DS	2	;Letzter UNALLOC Track
AF16	UNASEC: DS	1	;Letzter UNALLOC Sektor
;			
AF17	ERFLAG: DS	1	;Fehler-Flag
AF18	RSFLAG: DS	1	;READ Sektor Flag
AF19	READOP: DS	1	;1 bei READ Operation
AF1A	WRTYPE: DS	1	;WRITE Operation

AF1B	DMAADR:	DS	2	;Letzte DMA Adresse
AF1D	RW:	DS	1	;Temporäres Befehlsregister
AF1E	CURDSK:	DS	1	;Virtueller Diskettenzeiger
				;
				; RAM für BDOS
AF1F =	BEGDAT	EQU	\$	;Beginn des Datenbereichs
AF1F	DIRBF:	DS	128	;Bereich für Directory
AF9F	ALLOO:	DS	31	;Allocation Vektor 0
AFBE	ALLO1:	DS	31	;Allocation Vektor 1
AFDD	CHKOO:	DS	16	;Check Vektor 0
AFED	CHKO1:	DS	16	;Check Vektor 1
				;
AFFD =	ENDDAT	EQU	\$	;Ende des Datenbereichs
OODE =	DATSIZ	EQU	\$-BEGDAT	;Größe des Datenbereichs
AFFD			END	

6.8 IMPLEMENTIERUNG EIGENER EIN-/AUSGABEFUNKTIONEN INS BIOS

Sehen wir uns das BIOS-Listing im vorhergehenden Kapitel an, so können wir feststellen, daß für die Routinen PUNCH und READER lediglich die Sprungvektoren existieren, die eigentlichen Routinen jedoch nur aus einem Return-Befehl bestehen. Die READER-Routine gibt einfach Control Z zurück, das Kennzeichen für Ende der Textdatei. Hier können wir nun eigene Routinen einbinden. Die PUNCH-Routine könnte z.B. einen Drucker mit Centronicsschnittstelle unterstützen, wie es ähnlich auch in Kapitel 7.1 beschrieben ist. Die Treiberoutine selbst kann in 6510-Kode formuliert werden. Zur Ansprache der Routinen können wir die bisher unbenutzten Befehlskodes 7 und 8 benutzen, die zur 6510 Adresse \$E00 bzw. \$F00 verzweigen. Der Aufruf könnte z.B. so aussehen:

```
PUNCH: ;Zeichen auf PUN: Centronics-Drucker ausgeben
        MOV     A,C      ;Zeichen in Akku
        STA     DATA    ;in Übergaberegister
        MVI     A,7      ;Kode für eigene Routine
        STA     CMD
        CALL    IO6510   ;6510-Routine aufrufen
        RET
```

Unsere 6510-Ausgaberoutine müßte dann ab Adresse \$E00 stehen; es stehen 256 Byte dafür zur Verfügung. Da lediglich die Routine zur Ausgabe der Daten auf den Port und für den Handshake benötigt wird, steht Ihnen sicher genug Speicherplatz zur Verfügung.

Ähnlich läßt sich auch die READER-Routine implementieren.

READER: ;Zeichen von RDR: holen

```
MVI    A,8      ;Kode für eigene Routine
STA     CMD
CALL    IO6510   ;6510-Routine aufrufen
LDA     DATA    ;Zeichen holen
ANI     7FH      ;Paritätsbit löschen
RET
```

Der Befehlskode 8 erwartet die Eingaberoutine ab der Adresse \$F00; auch hier stehen Ihnen wieder 256 Bytes zur Verfügung. Als RDR: können Sie somit ein beliebiges Eingabegerät definieren, z.B. den Kassettenrekorder oder ein Diskettenlaufwerk. Anbieten würde sich hier auch eine V24 Schnittstelle zur Übertragung von Daten von anderen Rechnern. Die Routine READER erwartet Textdateien im Standard ASCII-Format. Das Dateiende wird im allgemeinen mit Control Z gekennzeichnet. Das PIP-Programm erwartet z.B. diesen Kode am Dateiende.

Sie können also nach Belieben eigene Ein- oder Ausgabegeräte vom CP/M aus bedienen, falls Sie die Routinen wie oben beschrieben ins BIOS implementieren.

6.9 ÜBERTRAGUNG VON CP/M PROGRAMMEN UND DATEN INS COMMODORE-BASIC UND UMGEKEHRT

Wenn man auf dem Commodore 64 unter CP/M arbeitet, hat man normalerweise keine Möglichkeit, Programme oder Daten, z.B. Textdateien, später im normalen BASIC-Modus des Commodore 64 zu nutzen. An diese Dateien kommt man nur im CP/M-Modus dran. Mit einer kleinen Änderung im BIOS65 ist dies jedoch möglich.

Sie können im CP/M ja Daten auf den Drucker ausgeben. Diese Ausgabe geschieht im BIOS65 über entsprechende OPEN und PRINT# Befehle. Hier können wir nun eingreifen und im OPEN-Befehl einfach die Gerätenummer ändern. Setzt man hier die Nummer eins ein, so geht sämtliche Ausgabe, die sonst zum Drucker ginge, zum Kassettenrekorder. Als Sekundäradresse müssen wir ebenfalls eins angeben, damit die Banddatei zum Schreiben geöffnet wird. Diese Änderungen können wir vom CP/M aus einfach mit dem DDT-Programm durchführen. Geben Sie dazu folgende Befehle ein (Ihre Eingaben sind unterstrichen dargestellt):

DDT

-SFAC7

FAC7 07 01

FAC8 20 1

-SFADD

FADD 00 01

FADE A9 1

-SFAE6

FAE6 04 01

FAE7 20 1

-^C

Machen Sie jetzt mit ^P oder PIP LST:= eine Ausgabe, so wird nicht mehr der Drucker, sondern das Kassettenlaufwerk angesprochen. Beim ersten Mal erscheint dann auf dem Bildschirm die Aufforderung 'press record & play on tape' und der Bildschirm wird dunkel. Ist sämtlicher Text ausgegeben, so drücken Sie bitte Control C und dann während des Warmstarts STOP und RESTORE gleichzeitig. Der Rechner meldet sich jetzt mit 'ready' wieder im Commodore-Modus. Jetzt müssen Sie noch die Banddatei mit CLOSE 4 schließen. Sie können jetzt nachdem Sie den Rechner aus- und wieder eingeschaltet haben, die Banddatei lesen:

```
100 OPEN 1
110 GET#1, A$
...
200 IF ST<>64 THEN 110
210 CLOSE 1
```

Das Programm holt die Datei zeichenweise in A\$ und Sie können die Daten im Commodoremodus bearbeiten.

Wollen Sie öfter Daten vom CP/M übertragen, so können Sie sich mit dem Editor (ED) und dem Assembler (ASM) ein kleines Programm schreiben, das die Änderung vornimmt, so daß Sie nicht immer mit dem DDT arbeiten müssen. Erstellen Sie mit dem Editor das folgende kleine Assemblerprogramm,

ED BAND.ASM

```
ORG 100H
MVI A,1
STA OFAC7H
STA OFADFH
STA OFAE6H
JMP 0
```

und assemblieren Sie es

ASM BAND.AAX

Machen Sie dann ein .COM-File daraus

LOAD BAND

Jetzt können Sie die Umschaltung einfach durchführen, indem Sie das Programm BAND durch Eingabe des Namens aufrufen.

Auch der umgekehrte Weg, die Übernahme von Daten von Commodore-Modus zum CP/M ist mit einem Trick möglich. Wollen Sie z.B. einen Text oder ein Programm übertragen, so müssen Sie daraus zuerst eine Programmdatei mit einer Ladeadresse von \$1100 machen. Dies ist die Z80-Adresse 100H, der Beginn der Transient Program Area (TPA). Wollen Sie einen Text aus einer sequentiellen Datei übertragen, können Sie folgendermaßen vorgehen:

```
100 INPUT "NAME DER DATEI ";N$
110 OPEN 2,8,2, N$
120 OPEN 1,8,1, N$+".CPM": REM PROGRAMDATEI ÖFFNEN
130 PRINT#1, CHR$(0)CHR$(17);:REM STARTADRESSE $1100
140 GET#2, A$:IF ST=64 THEN CLOSE 1:CLOSE 2: END
150 PRINT#1, A$;: GOTO 140
```

Bevor wir das Programm laden können, müssen wir noch die Größe kennen. Laden Sie dazu das Inhaltsverzeichnis der Diskette. Sie erhalten z.B

```
...
25  NAME.CPM      PRG
...
```

Die Zahl 25 müssen wir uns merken. Jetzt laden wir die Programmdatei.

```
LOAD "NAME.CPM",8,1
```

Legen Sie nun die CP/M-Diskette ein und starten Sie wie gewöhnlich. Wenn das CP/M geladen ist, können wir die Datei unter CP/M abspeichern.

```
SAVE 25 NAME.TXT
```

Die Zahl 25 gibt die Anzahl der zu speichernden 256-Byte-Blöcke an und ist mit der Blockzahl im Commodore-Directory identisch. Jetzt haben wir die Datei unter CP/M zur Verfügung und können sie benutzen. Bei Textdateien kann es eventuell zu Problemen bei der Klein/Großschreibung kommen. Ist dies der Fall, müssen Sie eine eventuelle Konvertierung von A\$ in Zeile 140 des obigen Programms durchführen, um den Standard-ASCII-Kode zu erhalten.

KAPITEL 7 ANSCHLUSS- UND ERWEITERUNGSMÖGLICHKEITEN DES COMMODORE 64

7.1 EINE SINNVOLLE ANWENDUNG DES USER-PORTS: SCHNITTSTELLE FÜR EINEN CENTRONICS-DRUCKER

Der Commodore 64 hat eine Schnittstelle, die normalerweise vom Betriebssystem nicht benutzt wird, und die Ihnen für eigene Anwendungen zur Verfügung steht. Diese Schnittstelle besteht aus einem 8-Bit Port und zwei Handshake-Leitungen. Der 8-Bit Port kann sowohl auf Ausgabe als auch auf Eingabe geschaltet sein und zwar unabhängig für jedes Bit.

Diese Schnittstelle eignet sich hervorragend zur Realisierung einer Druckerschnittstelle. Da die meisten Drucker serienmäßig über eine Centronics-Schnittstelle verfügen (z.B. Epson-Drucker, Centronics-Drucker), wollen wir diese Schnittstelle verwirklichen. Zudem läßt sich diese Schnittstelle besonders einfach handhaben. Hier kurz das Verfahren:

Die 8 Bit eines Byte werden gleichzeitig parallel über acht Datenleitungen übertragen. Damit bei der Übertragung keine Daten verloren gehen, werden zwei sogenannte 'Handshake-Leitungen' benutzt. Ehe der Rechner ein Datenbyte an den Drucker schickt, prüft er ob der Drucker bereit ist, Daten zu empfangen. Dies geschieht über die BUSY-Leitung. Ist die BUSY-Leitung high, ist der Drucker beschäftigt und der Rechner muß warten. Wenn der Drucker bereit ist, legt der Rechner die Daten auf den Port und signalisiert dies dem Drucker über die STROBE-Leitung. Der Drucker übernimmt die Daten und setzt die BUSY-Leitung solange high, bis die Daten verarbeitet (gespeichert oder gedruckt) sind. Jetzt kann das nächste Datenbyte übertragen werden. Durch diese Prozedur wird sichergestellt, daß der Drucker auch jedes Datenbyte

erhält, das der Rechner abschickt.

Nun zur Programmierung der Schnittstelle. Damit die Daten mit PRINT# zum Drucker geschickt werden können, muß die Software ins Betriebssystem eingebunden werden. Zusätzlich ergibt sich jedoch noch ein Problem:

Die meisten Drucker mit Centronics-Schnittstelle besitzen einen ASCII-Zeichensatz, der sich von Zeichensatz des Commodore 64 unterscheidet. Wir müssen deshalb eine Kodewandlung vornehmen, damit der Drucker das ASCII-Zeichen erhält, das dem Zeichen des Commodore 64 entspricht. Diese Wandlung ist so ausgelegt, daß die Zeichen, wie sie im Groß/Kleinschrift-Modus auf dem Bildschirm erscheinen, ebenso auf dem Drucker wiedergegeben werden. Zusätzlich ist es jedoch manchmal erforderlich, die Zeichen so zu übertragen, wie sie der Rechner sendet. Dies ist dann nötig, wenn z.B. Daten für Einzelpunktgrafik übertragen werden. In dem untenstehenden Programm wurde dies so gelöst:

Gibt man beim OPEN-Befehl keine Sekundäradresse an, so werden die Daten umgewandelt, damit die Groß/Kleinschrift richtig erscheint. Mit einer Sekundäradresse von eins werden die Daten ohne Wandlung übertragen. Als Geräteadresse wurde 2 gewählt, was normalerweise die RS 232-Schnittstelle ist. Da diese Schnittstelle jedoch nicht zusammen mit unserer betrieben werden kann, gibt es deshalb keine Probleme.

Wollen Sie z.B. ein Programmlisting über diese Schnittstelle ausgeben, sehen die Befehle dazu so aus:

```
OPEN 1,2 : CMD 1 : LIST
```

Nach dem der Cursor wieder erscheint, geben Sie

PRINT#1 : CLOSE 1

ein, um den CMD-Modus wieder aufzuheben und den Kanal zu schließen. Wollen Sie Grafikdaten übertragen, sieht die Programmierung so aus:

OPEN 1,2,1

PRINT# 1,

CLOSE 1

Zum hardwaremäßigen Anschluß brauchen wir lediglich ein Kabel mit USER-Port-Stecker auf der einen Seite und einem Centronics-Stecker auf der anderen. Die Pinbelegung des Kabels ist am Ende des Assemblerlistings angegeben. Beim Anschluß des Druckers gehen Sie so vor, daß Sie zuerst den Commodore 64 über das Kabel mit dem Drucker verbinden, dann den Rechner und danach den Drucker einschalten. Laden Sie jetzt das Maschinenprogramm und initialisieren Sie es mit SYS 12*4096.

; Centronics-Schnittstelle für CBM 64

; Anschluß an USER-Port

;

; Definition der I/O-Vektoren

;

200: 031a *= \$31a

210: 031c openv *= **2

220: 031e closev *= **2

230: 0320 chkinv *= **2

240: 0322 chkoutv *= **2

260: 0326 *= **4

270: 0328 bsoutv *= **2

280: 0328 xreg = \$97 ; Speicher für Register

;

; Definition der Ports

;

```

284: dd00          *= $dd00 ; CIA2
285: dd01      porta *= *+1 ; Port
286: dd02      portb *= *+1
287: dd03      drra  *= *+1 ; Datenrichtung
288: dd0d      drrb  *= *+10
289: dd0e      icr   *= *+1 ; Interrupt Control Register
290: dd0e      lf    = $b8
291: dd0e      sa    = $b9
292: dd0e      fa    = $ba
293: dd0e      nmbfiles = $98
294: dd0e      lftab = $259
295: dd0e      fatab = $263
296: dd0e      satab = $26d
297: dd0e      srchfil = $f30f

```

;

; Initialisierung

;

```

330: c000      init  *= $c000
340: c000 a9 58      lda #< open
350: c002 a0 c0      ldy #> open
360: c004 8d 1a 03    sta openv
370: c007 8c 1b 03    sty openv+1
380: c00a a9 8b      lda #< close
390: c00c a0 c0      ldy #> close
400: c00e 8d 1c 03    sta closev
410: c011 8c 1d 03    sty closev+1
420: c014 a9 a3      lda #< chkin
430: c016 a0 c0      ldy #> chkin
440: c018 8d 1e 03    sta chkinv
450: c01b 8c 1f 03    sty chkinv+1
460: c01e a9 ba      lda #< chkout
470: c020 a0 c0      ldy #> chkout

```

```

480:  c022 8d 20 03      sta  chkoutv
490:  c025 8c 21 03      sty  chkoutv+1
540:  c028 a9 d1         lda  #< bsout
550:  c02a a0 c0         ldy  #> bsout
560:  c02c 8d 26 03      sta  bsoutv
570:  c02f 8c 27 03      sty  bsoutv+1
580:  c032 a9 ff         lda  #$ff
590:  c034 8d 03 dd      sta  drrb      ; Port B auf Ausgabe
600:  c037 ad 00 dd      lda  porta
610:  c03a 09 04         ora  #%100      ; Strobe hi
610:  c03c 8d 00 dd      sta  porta
620:  c03f 60            rts

;
; Ausgabe mit Handshake
; Daten auf Port B
; Strobe auf PA2
; BUSY über FLAG nach ICR
;

795:  c040 8d 01 dd  ausgabe sta  portb      ; Daten ausgeben
800:  c043 a9 10         lda  #%10000      ; Maske für 'FLAG'-Bit
810:  c045 2c 0d dd  testbusy bit  icr
820:  c048 f0 fb         beq  testbusy
830:  c04a ad 00 dd      lda  porta
840:  c04d 29 fb         and  #%11111011 ; Strobe lo
850:  c04f 8d 00 dd      sta  porta
860:  c052 09 04         ora  #%100      ; Strobe hi
870:  c054 8d 00 dd      sta  porta
880:  c057 60            rts

;

900:  c058 a6 b8   open  ldx  lf      ; logische Filenummer
910:  c05a f0 05         beq  openerr
930:  c05c 20 0f f3      jsr  srchfil ; sucht Filenummer
940:  c05f d0 03         bne  *+5
950:  c061 4c fe f6  openerr jmp  $f6fe ; 'file open error'

```

```

960:  c064 a6 98          ldx  nmbfiles  ; Anzahl der offenen Files
970:  c066 e0 0a          cpx  #10
980:  c068 90 03          bcc  *+5
990:  c06a 4c fb f6       jmp  $f6fb    ; 'too many files error'
1000: c06d e6 98          inc  nmbfiles
1010: c06f a5 b8          lda  lf
1020: c071 9d 59 02       sta  lftab,x
1030: c074 a5 b9          lda  sa
1040: c076 09 60          ora  #$60
1050: c078 9d 6d 02       sta  satab,x
1060: c07b a5 ba          lda  fa
1070: c07d 9d 63 02       sta  fatab,x
1080: c080 c9 02          cmp  #2
1090: c082 d0 02          bne  *+4
1100: c084 18            clc
1100: c085 60            rts          ; fertig
1110: c086 c9 00          cmp  #0
1120: c088 4c 77 f3       jmp  $f377    ; weiter wie gehabt

;
1300: c08b 20 14 f3 close jsr  $f314    ; sucht logische Filenummer
1310: c08e f0 02          beq  *+4
1320: c090 18            clc
1320: c091 60            rts          ; fertig
1330: c092 20 1f f3       jsr  $f31f    ; setzt File Parameter
1340: c095 8a            txa
1340: c096 48            pha
1350: c097 a5 ba          lda  fa
1360: c099 c9 02          cmp  #2
1370: c09b f0 03          beq  *+5
1380: c09d 4c 9d f2       jmp  $f29d    ; normal weiter
1390: c0a0 4c f1 f2       jmp  $f2f1    ; Eintrag in Tabelle löschen

;
1400: c0a3 20 0f f3 chkin jsr  srchfil  ; sucht Filenummer
1410: c0a6 f0 03          beq  *+5

```

```

1420: c0a8 4c 01 f7      jmp $f701 ; 'file not open error'
1430: c0ab 20 1f f3      jsr $f31f ; setzt Fileparameter
1440: c0ae a5 ba      lda fa
1450: c0b0 c9 02      cmp #2
1460: c0b2 d0 03      bne *+5
1470: c0b4 4c 0a f7      jmp $f70a ; 'not input file error'
1480: c0b7 4c 19 f2      jmp $f219

;
1500: c0ba 20 0f f3 chkout jsr srchfil ; sucht Filenummer
1510: c0bd f0 03      beq *+5
1520: c0bf 4c 01 f7      jmp $f701 ; 'file not open error'
1530: c0c2 20 1f f3      jsr $f31f ; setzt Fileparameter
1540: c0c5 a5 ba      lda fa
1550: c0c7 c9 02      cmp #2
1560: c0c9 d0 03      bne *+5
1570: c0cb 4c 75 f2      jmp $f275
1580: c0ce 4c 5b f2      jmp $f25b

;
1600: c0d1 48      bsout pha
1610: c0d2 a5 9a      lda $9a ; Outputgerät
1620: c0d4 c9 02      cmp #2
1630: c0d6 f0 03      beq *+5
1640: c0d8 4c cd f1      jmp $f1cd ; normal weiter
1641: c0db a5 b9      lda sa ; Sekundäradresse
1641: c0dd 29 0f      and #$1111
1642: c0df d0 0a      bne out ; ungleich null
1642: c0e1 86 97      stx xreg
1643: c0e3 68      pla
1643: c0e4 aa      tax
1643: c0e5 bd f3 c0      lda table,x ; Kode aus Tabelle holen
1643: c0e8 a6 97      ldx xreg
1643: c0ea 24      .byt $24
1650: c0eb 68      out pla
1655: c0ec 48      pha

```

```

1660: c0ed 20 40 c0      jsr  ausgabe ; Zeichen ausgeben
1660: c0f0 68           pla
1660: c0f1 18           clc
1660: c0f2 60           rts
1665: c0f3           table  =  *
      c0f3 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
      c103 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
      c113 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
      c123 30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f
      c133 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f
      c143 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f
      c153 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f
      c163 70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f
      c173 80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f
      c183 90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f
      c193 a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 aa ab ac ad ae af
      c1a3 b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 ba bb bc bd be bf
      c1b3 c0 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f
      c1c3 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f
      c1d3 e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef
      c1e3 f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff

```

Hier wieder ein Ladeprogramm in BASIC

```

100 for i = 49152 to 49650
110 read x : poke i,x : s=s+x : next
120 data 169, 88,160,192,141, 26,  3,140, 27,  3,169,139
130 data 160,192,141, 28,  3,140, 29,  3,169,163,160,192
140 data 141, 30,  3,140, 31,  3,169,186,160,192,141, 32
150 data  3,140, 33,  3,169,209,160,192,141, 38,  3,140
160 data 39,  3,169,255,141,  3,221,173,  0,221,  9,  4
170 data 141,  0,221, 96,141,  1,221,169, 16, 44, 13,221

```


180 data 240,251,173, 0,221, 41,251,141, 0,221, 9, 4
 190 data 141, 0,221, 96,166,184,240, 5, 32, 15,243,208
 200 data 3, 76,254,246,166,152,224, 10,144, 3, 76,251
 210 data 246,230,152,165,184,157, 89, 2,165,185, 9, 96
 220 data 157,109, 2,165,186,157, 99, 2,201, 2,208, 2
 230 data 24, 96,201, 0, 76,119,243, 32, 20,243,240, 2
 240 data 24, 96, 32, 31,243,138, 72,165,186,201, 2,240
 250 data 3, 76,157,242, 76,241,242, 32, 15,243,240, 3
 260 data 76, 1,247, 32, 31,243,165,186,201, 2,208, 3
 270 data 76, 10,247, 76, 25,242, 32, 15,243,240, 3, 76
 280 data 1,247, 32, 31,243,165,186,201, 2,208, 3, 76
 290 data 117,242, 76, 91,242, 72,165,154,201, 2,240, 3
 300 data 76,205,241,165,185, 41, 15,208, 10,134,151,104
 310 data 170,189,243,192,166,151, 36,104, 72, 32, 64,192
 320 data 104, 24, 96, 0, 1, 2, 3, 4, 5, 6, 7, 8
 330 data 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
 340 data 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32
 350 data 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44
 360 data 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56
 370 data 57, 58, 59, 60, 61, 62, 63, 64, 97, 98, 99,100
 380 data 101,102,103,104,105,106,107,108,109,110,111,112
 390 data 113,114,115,116,117,118,119,120,121,122,123,124
 400 data 125, 94, 95, 96, 97, 98, 99,100,101,102,103,104
 410 data 105,106,107,108,109,110,111,112,113,114,115,116
 420 data 117,118,119,120,121,122,123,124,125,126,127,128
 430 data 129,130,131,132,133,134,135,136,137,138,139,140
 440 data 141,142,143,144,145,146,147,148,149,150,151,152
 450 data 153,154,155,156,157,158,159,160,161,162,163,164
 460 data 165,166,167,168,169,170,171,172,173,174,175,176
 470 data 177,178,179,180,181,182,183,184,185,186,187,188
 480 data 189,190,191,192, 65, 66, 67, 68, 69, 70, 71, 72
 490 data 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84
 500 data 85, 86, 87, 88, 89, 90, 91, 92, 93,222,223,224
 510 data 225,226,227,228,229,230,231,232,233,234,235,236

```

520 data 237,238,239,240,241,242,243,244,245,246,247,248
530 data 249,250,251,252,253,254,255
540 if s <> 58498 then print "fehler in datas !!" : end
550 sys 49152 : print "ok !"

```

Das Verbindungskabel zwischen Userport und Druckerschnittstelle hat folgende Belegung:

USER PORT	-	CENTRONICS
A	GND	16
B	FLAG - BUSY	11
C	D0	2
D	D1	3
E	D2	4
F	D3	5
H	D4	6
J	D5	7
K	D6	8
L	D7	9
M	PA2 - STROBE	1

7.2 DATENÜBERTRAGUNG VON UND ZU ANDEREN RECHNERN MITTELS DES USER-PORT

Nehmen wir einmal folgenden Fall an:

Sie besitzen außer Ihrem CBM64, den Sie sich wegen der Möglichkeiten in Farbe und Ton zugelegt haben, noch einen CBM8032. Wie schön wäre es doch, wenn Sie Daten, die aus dem 8032 stammen, zwecks farbiger Anzeige auf den CBM64 transferieren könnten, und zwar in dem Augenblick, wo sie auch anfallen.

Sie könnten wohl die Daten mittels Kassette austauschen, falls Sie ein solches Gerät besitzen, aber das ist ein umständliches und langwieriges Unterfangen.

Um Ihnen den Nutzen des Userport einmal aufzuzeigen (die wenigsten wissen damit etwas anzufangen), haben wir eben dieses Beispiel gewählt und dazu auch gleich ein kleines Programm geschrieben, welches es erlaubt, Daten von und zu Ihrem C64 zu übertragen. In unserem Falle ist der Partner ein CBM8032. Es ist jedoch durchaus denkbar, dieses Verfahren auch mit anderen Rechnern zu versuchen, die über eine ähnliche Einrichtung wie den Userport verfügen.

Einzige Voraussetzung ist eben, daß diese Einrichtung über acht Datenleitungen und zwei Handshakeleitungen verfügt. Geeignet wäre z.B. auch eine Centronicsschnittstelle, über die ja auch im Abschnitt 7.1 einiges gesagt ist.

Das Programm, von dem im folgenden die Rede sein wird, benötigt natürlich zum geordneten Ablauf eine Verbindung zwischen den beiden Rechnern.

Anweisungen, wie eine solche Leitung beschaffen sein muß, finden Sie hinter dem Programmlisting.

Zunächst werden wir Ihnen das Programm vorstellen. Vorab die verwendeten Variablen und Speicherplätze, die in diesem Falle keinen echten Speicher darstellen, sondern Adressen

sind, mittels derer der Userport gehandhabt wird, und anschließend eine schrittweise Erläuterung zu dem Programm.

Zunächst die Variablen:

X ASC-Wert eines zu sendenden oder empfangenen Bytes
TI Betriebssystemuhr; zählt in Schritten von 1/60 sec.
D\$ vollständiger zu sendender oder empfangener String

Verwendete Speicheradressen im CBM8032:

59457 Datenregister des Userport

59459 Datenrichtungsregister des Userport. Wie Ihnen vielleicht bekannt ist, kann der Userport Daten sowohl senden als auch empfangen. Deshalb muß hier die Datenrichtung festgelegt werden, was sogar für jedes einzelne Bit möglich ist.

59468 Bit 5 dieser Adresse steuert die Leitung CB2 des Userport.

Im Sendebetrieb wird hiermit die Gültigkeit der Daten auf der Leitung angezeigt.

Im Empfangsbetrieb dient sie als Quittungssignal dafür, daß die Daten übernommen wurden. Dies stellt sicher, daß keine Daten verlorengehen.

59469 Bit 1 gibt den Zustand der Leitung CA1 des Userport wieder.

Im Sendebetrieb wird hier auf das Quittungssignal gewartet, während im Empfangsbetrieb auf die Gültigmeldung des Senders gewartet wird.

Speicheradressen im CBM64

56576 Bit 2 steuert die Leitung PA2 des Userport.

Verwendung wie unter 59468 beschrieben.

56577 Datenregister des Userport

56579 Datenrichtungsregister des Userport

56589 Bit 4 gibt den Zustand der Leitung FLAG des Userport wieder.

Verwendung wie unter 59469 beschrieben.

Nun das Programm im Einzelnen:

1000-1080 Senderoutine

1000 Das Datenrichtungsregister wird auf Ausgabe gesetzt.

1010 Die Länge der Sendeschleife entspricht der Anzahl der zu übertragenden Bytes.

1020 Das durch I indizierte Byte in D\$ wird in das Datenregister geschrieben.

1030 Zum Zeichen, daß die Daten auf der Leitung jetzt gültig sind, wird das entsprechende Signal =0 gesetzt.

1040 In dieser Schleife wird solange gewartet, bis der Empfänger den Erhalt des Bytes quittiert.

1050-1060 Anschließend wird das Gültigkeitssignal wieder =1 gesetzt und das nächste Byte übergeben, falls noch weitere vorhanden sind.

1070-1080 Jetzt wird der Userport wieder in den Ruhezustand versetzt und die Senderoutine verlassen.

2000-2090 Empfangsroutine

2000 Hier wird auf das erste Byte gewartet.

2020-2030 Die Uhr wird =0 gesetzt und in der Folge darauf gewartet, daß entweder ein Byte empfangen wird oder zwei Sekunden herumgegangen sind. Ist das Letztere der Fall, so kann angenommen werden, daß

die Übertragung zu Ende ist. Das Zeitlimit in 2020 läßt sich nach Bedarf verändern.

In unserem Falle wurde in 2020 abgefragt, ob ti mehr als 120 Zeiteinheiten enthält, was 2sec entspricht, nämlich $120 * 1/60 \text{sec}$. Für 3sec wäre der entsprechende Wert 180.

2040-2050 Zunächst wird auf die Gültigkeitsmeldung gewartet. Ist diese eingetroffen, wird das in X gewonnene Byte an D\$ angehängt.

2060-2080 Als Quittung dafür wird die entsprechende Leitung vorübergehend =0 gesetzt und anschließend auf das Eintreffen des nächsten Zeichens gewartet.

Der Einsatz dieser Routinen ist sehr einfach.

Nachdem Sie ein entsprechendes Kabel angefertigt und die in Frage kommenden Rechner damit verbunden haben, kann es eigentlich schon losgehen.

Das beschriebene Programm besteht eigentlich nur aus zwei Unterprogrammen: Eins zum Senden und eines zum Empfang.

Diese Unterprogramme sollten Sie in Ihr eigenes Programm einbinden. Wollen Sie Zeichen senden, so schaffen Sie diese nach D\$ und sagen GOSUB1000.

Wollen Sie empfangen, so sagen sie GOSUB2000 und haben nach der Rückkehr die Zeichen in D\$ zur Verfügung, so einfach ist das.

Nachfolgend finden Sie zwei Programmlistings: eines für den C64 und eins für den CBM8032. In der Struktur sind beide gleich, jedoch sind die Userportadressen zwangsläufig andere.

Außerdem unterscheiden sie sich noch in der Zeile 2010. Hier muß mit Rücksicht auf den andersartigen Baustein, der im C64 den Userport handhabt, ein Sprung eingefügt werden.

```

995 rem subroutinensammlung zur uebertragung
    von daten ueber den userport
996 rem version für 6522 an adresse 59456 des CBM8032
998 rem
999 rem senden eines strings
1000 poke 59459,255:rem datenrichtung ausgang
1005 poke 59468,peek(59468) or 224:rem cb2 high
1010 for i=1 to len(d$):rem sendeschleife für d$
1020 x=asc(mid$(d$,i,1)):poke 59457,x:rem daten ausgeben
1030 poke 59468,peek(59468) and 223:rem cb2 low
1040 wait 59469,2:
    rem warten bis daten empfangen
1050 poke 59468,peek(59468) or 224:rem cb2 high
1060 next
1070 poke 59457,0:poke 59459,0:rem port rücksetzen
1080 return
1996 rem
1997 rem empfangen von d$
1998 rem
1999 rem
2000 wait 59469,2:
    rem anfang der übertragung abwarten
2010 d$="":rem d$ vorbereiten
2020 ti$="000000"
2030 if ti>120 then 2090
2040 if (peek(59469) and 2)=0 then 2030:
    rem warten auf datenbyte
2050 x=peek(59457):d$=d$+chr$(x)
2060 poke 59468,peek(59468) and 223:rem cb2 low
2070 poke 59468,peek(59468) or 224:
    rem cb2 high = empfangsbestätigung
2080 goto 2020
2090 return

```

```

995 rem subroutinensammlung zur übertragung
    von daten über den userport
996 rem version fuer 6526 an adresse 56576 des CBM64
998 rem
999 rem senden eines strings
1000 poke 56579,255:rem datenrichtung ausgang
1010 for i=1 to len(d$):rem sendeschleife für d$
1020 x=asc(mid$(d$,i,1)):poke 56577,x:
    rem daten ausgeben
1030 poke 56576,147:rem pa2 low
1040 wait 56589,16:
    rem warten bis daten empfangen
1050 poke 56576,151:rem pa2 high
1060 next
1070 poke 56577,0:poke 56579,0:rem port rücksetzen
1080 return
1996 rem
1997 rem empfangen von d$
1998 rem
1999 rem
2000 wait 56589,16:
    rem anfang der übertragung abwarten
2010 d$="":goto 2050:rem d$ vorbereiten
2020 ti$="000000"
2030 if ti>120 then 2090
2040 if (peek(56589) and 16)=0 then 2030:
    rem warten auf datenbyte
2050 x=peek(56577):d$=d$+chr$(x)
2060 poke 56576,147:rem pa2 low
2070 poke 56576,151:rem pa2 high:
    rem empfangsbestätigung
2080 goto 2020
2090 return

```


Ein kleines Beispiel soll die Anwendung der Routinen verdeutlichen:

Vorausgesetzt, sie haben in beiden Rechnern die entsprechende Routine geladen, fügen Sie nur noch im Sender die Zeile

```
10 d$="test":gosub1000:end
```

ein, und im Empfänger die Zeile

```
10 gosub2000:print d$:end
```

Starten Sie nun beide Programme und schauen Sie, was geschieht.

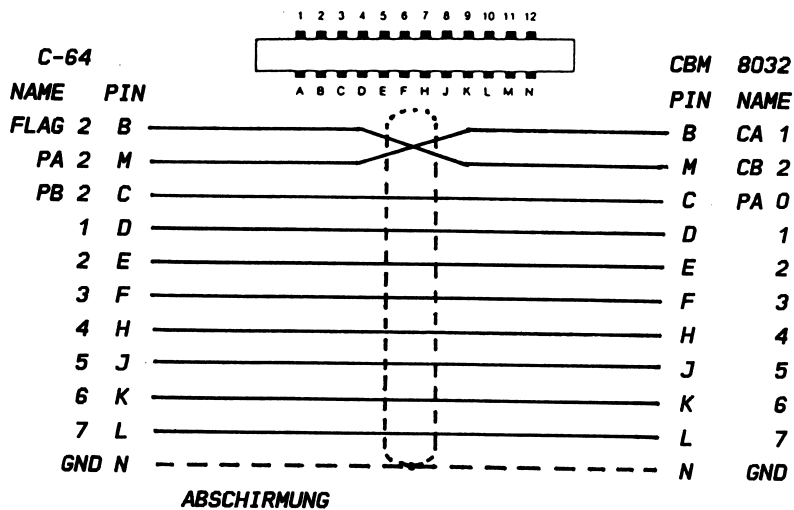
Die Verbindungsleitung ist nach untenstehendem Diagramm anzufertigen.

Wir empfehlen als Kabel eine 10-adrige, abgeschirmte Leitung mit einem Querschnitt der Einzeladern von $0,14-0,35 \text{ mm}^2$.

Die Abschirmung ist auf beiden Seiten mit dem Pin GND zu verbinden.

Bei unseren Versuchen funktionierte die Anordnung bei einer Leitungslänge von 5m fehlerfrei. Ob wesentlich größere Längen noch möglich sind, kann nicht vorhergesagt werden. Das hängt u.a. auch von der Umgebung und deren Störfeldern ab. Z.B. erzeugen Elektromotoren (Waschmaschine, Staubsauger) starke Störungen, die zu Fehlern in der Übertragung führen können.

Als völlig unbedenklich kann eine Leitungslänge von 3m angesehen werden.



7.3 DIE CP/M-CARTRIDGE AM EXPANSIONSPORT EINE FALLSTUDIE

In diesem Abschnitt wird beschrieben, wie durch eine ausgeklügelte Zusatzhardware optimaler Gebrauch vom Expansionslot des C64 gemacht wird. Zum besseren Verständnis der hier beschriebenen Vorgänge ist die Kenntnis der entsprechenden Kapitel unseres Buches '64 intern' von Vorteil.

Zunächst: Was ist die CP/M-Cartridge?

Die CP/M-Cartridge ist ein von Commodore entwickeltes Steckmodul, welches im Wesentlichen einen Prozessor vom Typ Z80 und einige zur Kommunikation mit dem C64 notwendige Logikbausteine beinhaltet.

Dieses Modul ermöglicht es, das weitverbreitete Betriebssystem CP/M und die vielfältige, hierzu erhältliche Software auf dem C64 zu betreiben.

Wie man es in diesem Falle anstellt, zwei Prozessoren an der gleichen übrigen Hardware zu betreiben, wollen wir Ihnen hier näherbringen.

Dazu finden Sie am Ende des Abschnittes ein Blockschaltbild der CP/M-Cartridge, welches sich auf das Wesentliche beschränkt, um die Darstellung nicht unnötig kompliziert zu machen.

Die nun folgende Beschreibung der einzelnen Vorgänge bezieht sich auf diese Abbildung und die in ihr enthaltenen Bezeichnungen der Funktionsgruppen.

Wir wollen versuchen, die Abläufe so darzustellen, daß Sie nicht unbedingt ein Hardwareexperte sein müssen, um diese zu verstehen.

Zunächst zur Einstimmung auf das Thema eine kurze

Beschreibung der Leitungen des Expansionsport, soweit sie in diesem Zusammenhang eine Rolle spielen:

CDO-7 Datenleitungen des Systembus.

Diese können vom Prozessor 6510 im C64 nur kontrolliert werden, solange DMA=1 und BA=1 ist.

Diesen Umstand sollten Sie sich merken, da er für den weiteren Verlauf wesentlich ist.

CA0-15 Adressleitungen des Systembus.

Für diese trifft das oben gesagte ebenfalls zu.

I/O1 Diese Leitung ist =0, wenn irgendwelche Aktivitäten im Adressbereich \$DE00-DEFF (56832-57087) stattfinden.

RES Wenn diese Leitung =0 wird (in der Regel nur beim Einschalten der Fall), wird alle angeschlossene Hardware in den Grundzustand versetzt.

DMA Diese Leitung ist ein Eingang. Setzt man sie =0, so wird der Prozessor 6510 stillgelegt, wobei dieser den Systembus zur externen Steuerung freigibt.

BA Auf dieser Leitung signalisiert der Videocontroller im C64, daß er auf den Speicher zugreift (BA=0). Während dieser Zeit darf auf den Systembus weder vom 6510, noch von außen via Expansionsport zugegriffen werden.

S02 Dies ist der Systemtakt, der sämtliche Vorgänge im C64 steuert.

Um alle Aktivitäten synchron zu denen im C64 zu auszuführen, wird in unserem Falle auch der 280 des CP/M-Moduls von diesem Takt gesteuert.

Zur Beschreibung der weiteren Vorgänge beginnen wir am besten mit dem Ruhezustand, also der Zustand, der sich nach dem Einschalten des Gerätes einstellt.

Vorab die Erklärung einer Leitung am Z80, nämlich BUSREQ. Dieses Signal hat eine ähnliche Wirkung wie AEC (hervorgerufen durch DMA=0) am 6510. Ist nämlich BUSREQ=0, so stellt der Z80 alle Aktivitäten ein und gibt seinen Systembus frei.

Wird also das Gerät eingeschaltet, so ist die Leitung RES kurzzeitig =0, wodurch auf dem Modul der Z80 in den Ausgangszustand geht, und das Flipflop FF zurückgesetzt wird (Q=0, -Q=1).

Damit wird der Ausgang des Gatters & ebenfalls =0, unabhängig von BA.

Das wiederum hat zur Folge, daß die Puffer A1, A2 und D verriegelt werden und somit nicht den Systembus steuern können.

Des weiteren wird das Signal BUSREQ =0, d.h. der Z80 verharrt im Grundzustand.

Sie können jetzt schon sehen, daß die Funktionstüchtigkeit des ganzen Moduls nur vom Zustand des Flipflops FF in Verbindung mit dem Signal BA (Verknüpfung durch das UND-Gatter &) abhängt.

Nur wenn FF gesetzt (Q=1, -Q=0) und gleichzeitig BA=1 ist, wird BUSREQ =1, sodaß der Z80 arbeiten kann.

Sie können nun mit Ihrem C64 wie gewohnt arbeiten, solange Sie einen bestimmten Befehl nicht ausführen, nämlich POKE 56832,1.

Wie Sie aus der Blockschaltung und der Beschreibung des Expansionslot entnehmen können, wird durch diesen POKE die Leitung I/O1 aktiviert. Da wir auch noch ausgerechnet den Wert 1 poken, wird durch dieses Kommando unser Flipflop FF gesetzt, und schon läuft der Z80 los, denn BA ist die meiste Zeit =1.

Gleichzeitig wird der 6510 durch DMA=0 abgeschaltet und Ihr Rechner stürzt ab, da Sie ja kein Programm im Speicher

stehen haben, welches der Z80 jetzt sinnvoll ausführen könnte, womit wir gleich beim nächsten Thema sind, nämlich: Wo muß ein Programm stehen, damit es der Z80 ausführen kann? Dazu müssen wir ein wenig weiter ausholen.

Im Gegensatz zum 6510 beginnt der Z80 nach dem Reset (RES=0) mit der Abarbeitung seines Programmes an der Speicheradresse 0.

Hier entsteht der erste Konfliktfall, denn bei 0 hat der 6510 seinen I/O-Port und auf den folgenden Adressen befindet sich die Zeropage, ein Bereich, den der Prozessor zum Betrieb unbedingt benötigt, da hier auch wichtige Betriebssystemparameter hinterlegt sind. In diesem Adressraum kann also dem Z80 unmöglich ein Programm zur Verfügung gestellt werden. Andererseits kann man ihn auch nicht veranlassen, seinen Betrieb bei einer anderen Adresse als 0 aufzunehmen.

Dieses Dilemma wird auf dem CP/M-Modul sehr elegant gelöst. Wenn Sie sich das Blockschema anschauen, finden Sie einen Funktionsblock, den wir mit ADD bezeichnet haben. Dahinter verbirgt sich ein 4-Bit-Volladdierer. Dieser nimmt an seinen Eingängen je ein 4-Bit-Wort auf, addiert die beiden und stellt die Summe an seinem Ausgang zur Verfügung.

Dieser Addierer ist in unserem Falle in die höchstwertigen vier Adressbits eingeschleift. In einen Eingang gelangen die Adressleitungen aus dem Z80, der andere Eingang ist fest auf den Wert 1 verdrahtet. Demzufolge erscheint am Ausgang der um 1 erhöhte Wert der oberen vier Adressbits.

Auf die ganze Adresse gesehen entspricht das einer Erhöhung um \$1000 (4096), da ja mit dem höchsten Digit einer 2-Byte-Adresse 4k-Schritte gezählt werden.

Wenn jetzt also, um bei unserem Beispiel zu bleiben, der Z80 beim Start die Adresse 0 ausgibt, um sich von dort den ersten Befehl zu holen, entspricht dies durch die Addition einer effektiven Adresse von \$1000 (4096).

Dort kann ohne weiteres ein für den Z80 brauchbares Programm hinterlegt sein, ohne das Betriebssystem des 6510 zu stören. Aus der beschriebenen Methode folgt auch, daß einer Z80-Adresse von \$F000 (61440) eine effektive Adresse von 0 entspricht, da bei der Addition ein Übertrag auf die nächste Stelle unberücksichtigt bleibt.

So wie oben beschrieben, wird auch tatsächlich bei der echten Arbeit mit dem Modul verfahren: Nach dem Einschalten des Rechners wird ein kleines Startprogramm in den Speicher geladen (natürlich ab \$1000), und nach Setzen des Flipflops FF beginnt der Z80 dort sein Programm zu bearbeiten, welches für gewöhnlich darin besteht, das Betriebssystem CP/M zu laden und dann die weiter vom Anwender gegebenen Anweisungen auszuführen.

Dieses Verfahren sollten Sie auch anwenden, wenn Sie eigene Z80-Programme laufen lassen wollen. Das von Ihnen geschriebene Programm legen Sie einfach bei 4096 (\$1000) ab und schalten wie beschrieben die Cartridge ein.

Da sicher ein solches Programm nicht Selbstzweck ist, sondern auch irgendeinen Output hervorbringen sollte, sei es auf dem Bildschirm oder einem Drucker, ist natürlich die genaue Kenntnis der C64-Hardwareadressen unerlässlich, damit Sie vom Z80-Programm aus die richtigen Aktionen durchführen. Berücksichtigen Sie dabei aber immer den auf dem Modul vorgenommenen Offset auf die Adresse. Wollen Sie z.B. Daten auf den Userport bringen, sollte die Adresse in Ihrem Z80-Programm \$CD01 (52481) lauten, da der Userport vom 6510 aus mit \$DD01 (56577) angesprochen wird. Ziehen Sie also von den gewünschten Adressen in Ihrem Programm immer \$1000 (4096) ab.

Wie kommt man aber aus der Z80-Betriebsart wieder heraus?

Nichts einfacher als das: Falls Sie den Basicinterpreter geladen haben, sagen Sie POKE 52736,0 um FF wieder rückzusetzen. Das hat die weiteren Folgen, daß BUSREQ wieder

=0 wird, der Z80 also seinen Betrieb einstellt, und gleichzeitig DMA =1 wird, woraufhin der 6510 an der Stelle fortfährt, an der er zuvor aufgehört hatte.

Die Pokeadresse muß natürlich um 4096 niedriger liegen als vom 6510 her gesehen, da dieser Wert ja durch ADD wieder hinzuaddiert wird. Es ist allerdings nicht empfehlenswert, so vorzugehen, da der 6510 bei der Wiederaufnahme seines Betriebes kein geeignetes Programm vorfindet, denn im Speicher stehen ja noch Programmteile, die für den Z80-Betrieb gedacht waren, und mit diesen kann der 6510 nichts anfangen. Der Rechner stürzt ab.

Der Vollständigkeit halber sei noch erwähnt, daß der Z80 selbstverständlich auf sämtliche im C64 eingebaute Hardware zugreifen kann, da er ja den kompletten Adressraum von 64k überstreicht.

Bisher kam die Leitung BA ein wenig zu kurz. Das soll jetzt nachgeholt werden.

An und für sich ist BA nur ein Hilfssignal, welches den Verkehr auf dem Datenbus regelt. Auf das CP/M-Modul hat es jedoch einen schwerwiegenden Einfluß, wie aus der Verknüpfung mit & hervorgeht, denn unabhängig von der Stellung des FF hält der Z80 bei BA=0 auf jeden Fall an.

Wenn wir die Herkunft von BA näher untersuchen, wird klar, warum das so sein muß.

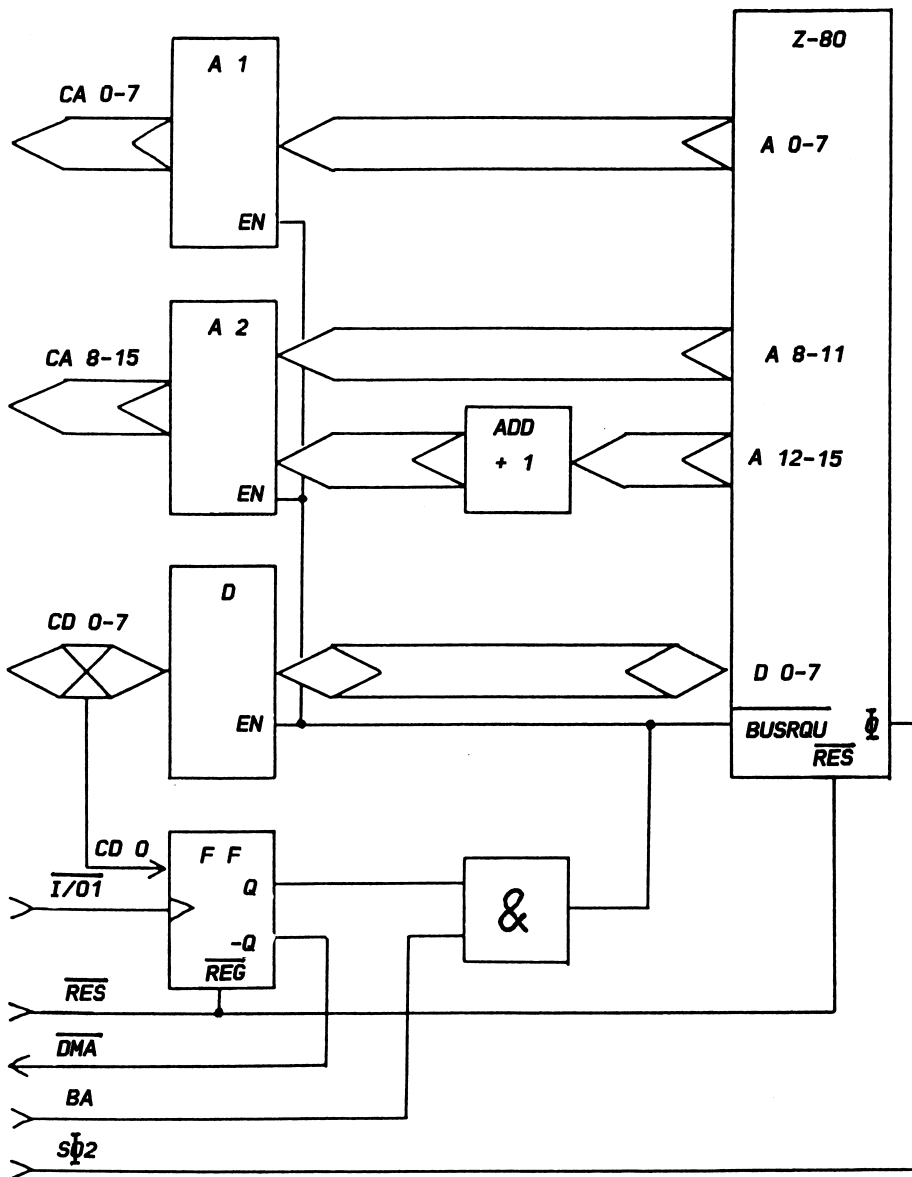
BA wird vom Videocontroller im C64 erzeugt. Da der Bildschirmspeicher im gewöhnlichen Adressraum liegt und der Videocontroller zyklisch zur Wiederauffrischung des Bildes darauf zugreifen muß, darf während dieser Zugriffszeit niemand anders den Systembus benutzen.

Normalerweise ist das auch nicht der Fall, denn der Controller nutzt für seinen Zugriff geschickt die Lücken aus, in denen der Prozessor sowieso nicht den Systembus benötigt. Es gibt allerdings einige Ausnahmen. Eine davon ist der Fall, wenn mit Sprites gearbeitet wird. Zur

Darstellung von Sprites nämlich reichen die erwähnten Lücken nicht aus, da hier mehrmals hintereinander auf den Speicher zugegriffen werden muß.

Diesen Umstand signalisiert der Videocontroller mit BA=0 und auch der Z80 muß dann den Systembus freigeben.

Das Blockschaltbild auf der folgenden Seite haben wir bewußt einfach gehalten, obwohl die Vorgänge im Bereich des Datenbuspuffers D wesentlich komplexer sind. Zum Verständnis der Interaktionen zwischen 6510 und Z80 ist es jedoch vollauf ausreichend.



7.4 SYNTHESIZER IN STEREO

Wenn Sie des Öfteren mit dem Synthesizer Ihres C64 gearbeitet haben, werden Sie sich auch sicher schon über den etwas mageren Klang geärgert haben, den Ihr Fernsehgerät produziert. Das mag auch von der Qualität des Tonteiles im Fernseher abhängen.

Eine evtl. vorhandene Stereoanlage fordert einen geradezu heraus, hier Abhilfe zu schaffen und einen etwas volleren Sound zu produzieren.

Wie Sie hierzu im einzelnen vorgehen können, möchten wir Ihnen jetzt zeigen.

Da eine Stereoanlage bekanntlich über zwei Kanäle verfügt, müssen wir uns zunächst Gedanken darüber machen, wie wir den einkanaligen Ausgang des Synthesizers auf diese beiden Kanäle aufteilen.

Leider sind die einzelnen Stimmen des Gerätes nicht an separaten Anschlüssen verfügbar, sonst wäre eine Aufteilung schon gegeben.

Also sind wir hingegangen und haben das Tonsignal in zwei Frequenzbereiche aufgeteilt. Die Aufteilung in unteren und oberen Bereich geschieht bei 300 Hz.

Damit wird der vom Synthesizer überstrichene Bereich gehörrechtig in der Mitte geteilt, nämlich in drei Oktaven unterhalb (bis 36 Hz) und vier Oktaven oberhalb (bis 4800 Hz) 300 Hz.

Realisiert wird dies mit zwei Doppel-T-Filtern mit einer Flankensteilheit von 6dB/Oktave und einer Grenzfrequenz von 300 Hz (Tiefpaß), bzw. 3 kHz (Hochpaß).

Die Grenzfrequenzen können Sie natürlich nach Geschmack durch Verwendung anderer Kondensatoren verändern, jedoch sollten Sie die Widerstandswerte beibehalten, da diese auf die Impedanz der angeschlossenen Geräte abgestimmt sind.

Bei gegebener Grenzfrequenz F errechnet sich der benötigte Kondensatorwert mit $C=1/(3300 \cdot F)$.

Haben sie irgendwelche Kondensatoren in Ihrer Bastelkiste und wollen wissen, welche Grenzfrequenz sich daraus ergibt, so ist die Formel $F=1/(3300 \cdot C)$ anzuwenden.

Sie können allerdings davon ausgehen, daß die Schaltung mit den angegebenen Werten ein Optimum darstellt, welches wir in unserem Labor durch ausgiebige Messungen und Hörtests erreicht haben.

Genügt Ihnen eine Flankensteilheit des Filters von 3dB/Oktave, so können Sie die Bauteile R_2 , C_2 , C_4 , R_6 auch weglassen. Dabei wird der Klang über einen weiteren Bereich akustisch zwischen den Lautsprechern stehen.

Wie Sie sehen, ist die Filterschaltung extrem einfach. Sie wird zweckmäßig auf einem kleinen Stück Lochrasterplatte aufgebaut und in die Verbindungsleitung zur Stereoanlage eingeschleift. Wer ein Übriges tun will, kann diese Anordnung mitsamt der angeschlossenen Kabelenden in Silikonkautschuk eingießen. Damit erhalten Sie eine Lösung, die auch optisch einwandfrei ist.

Wir wollen Ihnen nun ein kurzes Programm vorstellen, das einen 'Sweep' mit der Dreieckschwingung produziert.

Hierbei können Sie deutlich hören, wie der Ton von einem Lautsprecher zum anderen wandert.

Wir haben die Dreieckschwingung deshalb gewählt, weil sie relativ obertonarm ist und Sie deshalb den Effekt besser beobachten können.

Bei komplexeren, obertonreichen Klängen, z.B. Sägezahnschwingung, erscheinen die Obertöne auf dem einen Kanal, während Sie die Grundschiwingung aus dem anderen Lautsprecher hören, sofern diese 300 Hz nicht übersteigt.

Eines werden Sie aber sicher feststellen, daß nämlich der Klang wesentlich voluminöser geworden ist.

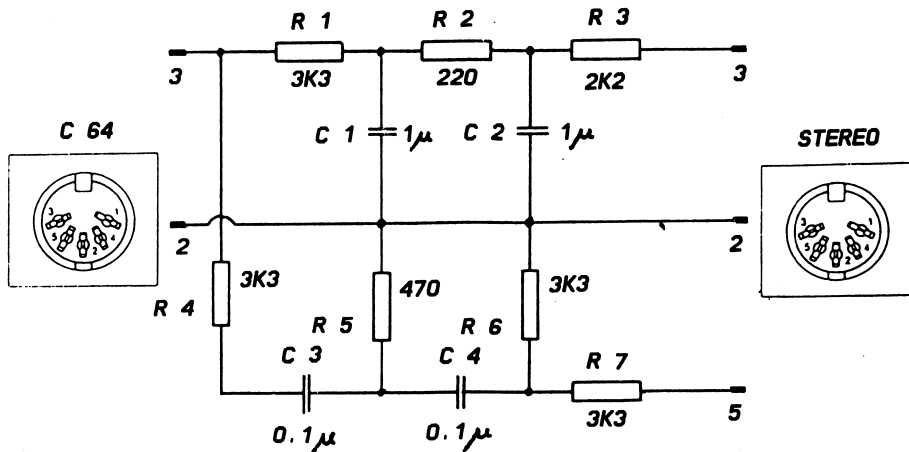
Hier nun das Progrämmchen:

```

10 s1=54272
20 s2=54279
30 s3=54286
60 rs=54295
70 pl=54296
80 poke s1+4,0:poke s2+4,0:poke s3+4,0
100 a=9:d=9:s=9:r=9:h=30
110 poke rs,0:poke pl,15
120 poke s3+5,16*a+d:poke s3+6,16*s+r
130 poke s3+4,17
140 for i=0 to h:poke s3+1,peek(54300):next i
150 poke s3+4,16
160 for i=0 to r*4:poke s3+1,peek(54300):next i

```

Auf die untenstehende Schaltung gehen wir ihrer Einfachheit wegen nicht näher ein. Die stereoseitige Steckerbelegung ist zum Anschluß an den Phonoeingang gedacht.



KAPITEL 8: DATEIVERWALTUNG: KEIN BUCH MIT SIEBEN SIEGELN

Einführung

Wer stand nicht schon häufiger vor der Aufgabe verschiedene Daten zu erfassen und sie an einem späteren Zeitpunkt wieder aufzurufen? Doch gerade diese Aufgabe ist sehr oft das komplizierteste Thema bei der Programmierung. Egal ob BASIC, FORTRAN, PASCAL oder andere Sprachen - Dateiverwaltung, und alles das, was mit Aufarbeitung von Daten zu tun hat, ist überall ein mehr oder weniger aufwendiges Problem. Man sollte erwarten, daß zumindest die Handbücher der Computer oder der Programmiersprachen genauere Auskunft über dieses Thema geben. Doch leider wird die Dateiverwaltung zu kurz, oder überhaupt nicht besprochen.

Wir wollen Ihnen in diesem Kapitel einen Einblick über die Dateiverwaltung auf dem Commodore 64 geben. Wir werden nicht nur trockene Theorie vermitteln, sondern auch Beispiele geben. So können auch Sie den Commodore 64 noch besser verstehen und anwenden. Zunächst müssen aber erst einmal ein paar Grundbegriffe aus dem Bereich der Dateiverwaltung erklärt werden.

DATEI

Alle Welt redet von Dateiverwaltung und Dateien - aber was sind eigentlich Dateien ? Am leichtesten läßt sich der Begriff erklären, wenn wir ihn durch einen Begriff ersetzen, den jeder kennt: KARTEI. Wie Sie wissen, besteht eine Kartei, z.B. in einer Bibliothek, aus mehreren Karteikarten. Auf diesen Karten steht jeweils die gesamte Information über einen bestimmten Gegenstand oder einer Person. So weiß man jederzeit, um bei unserem Beispiel zu bleiben, welcher Leser gerade welches Buch liest, oder welches Buch verliehen ist. Diese Karteikarten sind nach einem ganz bestimmten Muster geordnet. Die häufigste Methode ist das alphabetische Sortieren. Eine andere Möglichkeit ist die

Sortierung nach Artikelnummern oder nach dem Datum. Alle Karten zusammen ergeben nun eine Kartei. Ähnlich ist das bei einer Datei (im EDV-Sinne). Dort besteht eine Datei aus mehreren Datensätzen (Aufzeichnungen s.u.). In diesen Datensätzen stehen nun die einzelnen Informationen - also so wie auf einer Karteikarte.

Der große Vorteil einer DATEI gegenüber einer KARTEI (so wie wir sie eben erklärt haben) ist die unglaubliche Flexibilität der DATEI. Dazu gehört vor allen Dingen der Zeitgewinn beim Suchen und Sortieren aller Daten; aber auch die Platzersparnis spielt heute eine große Rolle. Auf den verwendeten Speichermedien der Kleincomputer finden oftmals schon mehr als 1 Million Zeichen Platz ! Können Sie sich vorstellen, wie viele Karteikarten benötigt würden, um diese Informationsmenge aufzunehmen ?

DATENSATZ

Wie wir oben schon besprochen haben, kann man einen Datensatz gut mit einer Karteikarte aus einer großen Kartei vergleichen. In diesem Datensatz stehen alle Daten, die sonst auf einer Karteikarte stehen würden, aufgeteilt auf eins oder mehrere FELDER.

FELDER

Auch hier kann man wieder das Beispiel der Karteikarte verwenden. Wenn man sich einen Datensatz als Karteikarte vorstellen kann, so sind die Felder die einzelnen Informationen auf der Karte. Die Verbindung der 3 Begriffe kann man sich ungefähr so vorstellen:

DATEI -> DATENSATZ -> FELD

Wenn ich also eine bestimmte Information über eine Sache oder eine Person haben will, z.B. den Namen, so muß zunächst auf die entsprechende Datei zugegriffen werden, dann aus dieser Datei den Datensatz und zum Schluß aus dem Datensatz noch das entsprechende Feld gelesen werden. Graphisch kann das folgendermaßen

dargestellt werden:

DATEI: ADRESSDATEI

	FELD NAME	FELD VORNAME	FELD STRASSE	FELD PLZ	FELD WOHNORT
REC.: 1	MEIER	HEINZ	A-STR. 1	1234	BITDORF
REC.: 2	MÜLLER	EGON	B-STR. 5	6502	CBMSTADT
REC.: 3	SCHULZE	HANS	C-STR. 2	4711	DUFTDORF

An diesem Beispiel einer Datei kann man recht deutlich die Unterschiede und Zusammenhänge zwischen Datei, Datensatz und Feld sehen. Diese Begriffe sollte man schon verstanden haben, bevor man mit der eigentlichen Programmierung von Programmen, die Dateien verwalten, beginnen kann. Im weiteren werden wir noch verschiedene Zugriffs- und Speichermethoden darstellen - die Grundlage ist aber auf jeden Fall das Verständnis des oben besprochenen.

8.1 CASSETTE - DISKETTE

Nach dieser etwas ausführlicheren Einführung wollen wir uns nun mit der eigentlichen Programmierung von Dateiverwaltungs-Programmen beschäftigen. Dazu sollten wir uns zunächst mit den Geräten befassen, die uns beim Commodore 64 zur Speicherung zur Verfügung stehen: Datasette und Diskettenlaufwerk.

Worin unterscheiden sich diese beiden Geräte und ihre Datenträger Cassette und Diskette ? Wozu lassen Sie sich verwenden ? Um diese Fragen zu klären wollen wir zunächst einen Ausflug in die Anfänge der EDV machen.

Noch vor ein paar Jahren waren Begriffe wie "Floppy" oder "Magnetplatte" so gut wie unbekannt. Aber auch damals konnte man natürlich nicht auf einen Datenträger, also ein Gerät, das Daten abspeichern und wieder lesen kann, verzichten. Aus diesem Zweck wurden die Lochkarten entwickelt. Mit ihnen hatte man eine einfache und preisgünstige Möglichkeit gefunden, Daten zu jeder beliebigen Zeit zu speichern und wieder dem Rechner zuzuführen. Man erkannte aber schon recht früh einen gewaltigen Nachteil der zur Arbeit mit der Lochkarte notwendigen Geräte, nämlich des Lochkartenstanzers (zur Informationsausgabe auf Lochkarten) und des Lochkartenlesers. Beide waren als rein mechanische Geräte viel zu langsam. Da man aber im Laufe der Zeit immer mehr auf rasche und sichere Datenspeicherung angewiesen war, mußte man etwas besseres entwickeln. Das Ergebnis: Das Magnetband. Und damit können wir uns auch gleich schon mit der heutigen Cassette befassen, deren Arbeitsprinzip dem des vor allem in der Groß-EDV noch recht häufig verwendeten Magnetbandes sehr ähnlich ist.

Das Prinzip der Cassette als Datenträger ist recht einfach (wenn man zunächst nur die Grundzüge betrachtet). Sehen wir uns zunächst die Datenspeicherung an. Der Commodore 64 hat für die Speicherung auf Cassette eine besondere Gerätezuweisung: Die Gerätenummer ist 1. Das Kommando zum Schreiben ebenfalls 1. Möchte man nun eine Datei auf dem Cassettenrecorder eröffnen, so

lautet der gesamte Befehl hierfür:

OPEN 1, 1, 1, "CBM 64 DATEI"

Die erste '1' ist die Dateinummer für den Commodore 64. Wollen Sie also mehrere Dateien im Commodore 64 öffnen, so müssen Sie für Drucker, Cassette oder Floppy andere Dateinummern wählen. Insgesamt sind alle Zahlen von 1 bis 255 zugelassen. Wenn der Cassettenrecorder auf RECORD & PLAY eingestellt ist, wird der Commodore 64 einen besonderen Vorspann auf das Band schreiben. Dieser Vorspann beinhaltet bei einer Datei nur den Dateinamen. Bei einem Programm wird noch zusätzlich die Startadresse mit abgespeichert. Dieser sogenannte Programm- oder der Dateikopf wird zweimal hintereinander abgespeichert. Danach stoppt das Band. Nun können die Daten abgespeichert werden (bzw. das Programm).

Eine zusätzliche Möglichkeit zum Abspeichern einer Datei besteht in der Sequenz:

OPEN 1, 1, 2, "CBM 64 DATEI"

Mit diesem Befehl schreiben Sie nach der Datei noch eine zusätzliche Information auf das Band. Diese Information, auch EOT (End Of Tape) oder "Ende des Bandes" genannt, sagt dem Commodore beim späteren Lesen, daß das Band hier beendet ist - es kann also nicht weiter beschrieben werden.

Daten die irgendwann einmal abgespeichert wurden, sollen natürlich auch wieder gelesen werden. Der entsprechende Befehl hierfür lautet:

OPEN 1, 1, 0, "CBM 64 DATEI"

Der Commodore 64 sucht nun solange nach einer entsprechenden Datei, bis er sie entweder gefunden hat, oder aber er auf einen EOT gestoßen ist.

WICHTIG: Beim Schreiben sucht der Commodore 64 nicht nach einer Datei mit dem angegebenen Namen. Er schreibt "ohne Rücksicht auf Verluste" genau an die Stelle, wo sich das Band gerade befindet. Man sollte deshalb nach Möglichkeit nicht mehr als eine Datei pro Cassette speichern, um unbeabsichtigtes Zerstören gespeicherter Daten und Programme zu verhindern.

Nachdem auch diese Magnetbänder nicht mehr ausreichten (warum, dazu kommen wir gleich noch), verfeinerte man die Technik. Magnetplatten wurden nun benutzt. Auch der Commodore 64 kann auf eine ähnliche Methode zurückgreifen. Hier kann man eine oder mehrere sogenannte Diskettenlaufwerke anschließen. Die dazugehörigen Datenträger, die Disketten, kann man am ehesten mit Schallplatten vergleichen. Auch hier gibt es verschiedene Spuren. Allerdings ist das Material der Disketten magnetisch. So können diese "Schallplatten" nicht nur gelesen sondern auch beschrieben werden. Die Syntax (Schreibweise) der Befehle zum Lesen bzw. Schreiben von Daten sehen folgendermaßen aus:

OPEN 2, 8, 2, "O:CBM 64 DATEI,S,W"

oder

OPEN 2, 8, 2, "O:CBM 64 DATEI,S,R"

Die erste '2' ist wieder die Commodore interne Dateinummer, '8' ist normalerweise die Gerätenummer für eine Floppy (kann aber auch z.B. auf '9' gestellt werden) und die zweite '2' die Kanalnummer. Das interessanteste ist aber der Namensteil. Hier finden wir zunächst die Angabe des Laufwerks (0 oder 1), dann den Dateinamen, dann ein 'S' für 'Sequentielle Datei' (dazu später mehr) und dann entweder ein 'W' für 'Write' (schreiben) oder 'R' für 'Read' (lesen). Somit können wir nun auch Dateien auf einer Diskette öffnen.

Die wichtigsten Unterschiede zwischen Diskette und Cassette

bestehen in:

- Kosten

Von den Anschaffungskosten her ist der Cassettenrecorder klar das billigste Speichermedium, auch wenn Diskettenlaufwerke in der letzten Zeit schon erheblich preiswerter geworden sind, und man eine VC-1541 Floppy schon für unter 1000 DM bekommt. Anders sieht es bei den Kosten für die eigentlichen Datenträger aus. Um z.B. die 170.000 Zeichen zu speichern, die auf eine Diskette des Commodore Laufwerks VC-1541 passen, braucht man ca. 4 C-60 Cassetten. Hier bietet der Cassettenrecorder also keinen Preisvorteil mehr.

- Zugriffszeit

Hier zeigen sich deutlich die Vorteile der Diskette. So dauert z.B. das Einlesen eines 10 K Programmes beim Cassettenrecorder 200 Sekunden, bei der Floppy VC-1541 aber nur 20 Sekunden. Ähnlich sieht es beim Durchsuchen einer sequentiellen Datei aus. Um z.B. eine Datei bestehend aus 50 Adressen, die je 100 Zeichen lang sind, zu lesen, benötigt man beim Cassettenrecorder 180 Sekunden, beim Diskettenlaufwerk nur 18 Sekunden.

- Zugriffsarten und Programmierkomfort

Während beim Cassettenrecorder nur Programmdateien und sequentielle Dateien zugelassen sind, bietet die Diskette mit relativen Dateien und Direktzugriff erheblich mehr Möglichkeiten.

Damit ist klar, daß der Cassettenrecorder lediglich ein preiswertes Gerät für Einsteiger ist. Wer mehr mit seinem Commodore 64 machen bzw. ihn sogar für kommerzielle Zwecke nutzen möchte, der kommt an einem Diskettenlaufwerk nicht vorbei. Im Folgenden beschäftigen wir uns zunächst mit den technischen Grundlagen der Datenspeicherung auf Cassette und wenden uns dann den einzelnen Zugriffsarten und Dateiformen zu.

Die Datasette - oder WIE KOMMEN DIE BITS AUFS BAND ?

Nachdem wir nun das Prinzip der Dateien erklärt haben (und Sie es hoffentlich auch verstanden haben), wollen wir an dieser Stelle noch erläutern, wie die einzelnen Informationen überhaupt aufs Band gelangen. Wir müssen zwar nun etwas technisch werden, aber wer weiß schon, wann welche Informationen einmal benötigt werden können. Und da uns auch dieses Thema sehr wichtig erschien, finden wir, daß auch Sie diese Informationen bekommen sollten.

Das von Commodore verwendete Verfahren zur Aufzeichnung der Informationen (Bits) auf das Cassettenband nennen wir PPM. PPM bedeutet PULSE POSITION MODULATION. Dies bedeutet, daß der Commodore 64, übrigens genau wie seine anderen Commodore-Brüder, die Digitalsignale direkt, und nicht in Form zweier Tonfrequenzen, aufs Band schreibt. Diese Digitalsignale werden in drei verschiedenen Zeiten übermittelt: K (Kurz), L (Lang) und M (Mittel). Aus diesen drei Zeiten werden nun auch 3 verschiedene Kombinationen gebildet, die die folgende Bedeutung haben:

LLMM = BYTE ; dieser Kombination geht jedem BYTE MMKK (=1), KKMM (=0) voraus.

Das heißt, daß zum Beispiel der Buchstabe "A" auf dem Band in dieser Form abgespeichert wird:

LLMM	MMKK	KKMM	KKMM	KKMM	KKMM	KKMM	MMKK	KKMM	MMKK
BYTE	1	0	0	0	0	0	1	0	1
BIT#	0	1	2	3	4	5	6	7	Parity ODD

Die ganze Abspeicherung sieht wie folgt aus:

Programme	Dateien
-----	-----
Programmkopf	Dateikopf
Start- Endadresse, Name	Name
Programmkopf (Wiederholung)	Dateikopf (Wiederholung)

Programm (ein Block)	Datenblock (ein Block)
Programm (Wiederholung)	Datenblock (Wiederholung)
Endblock	Endblock

Der Kopf ist jeweils so aufgebaut:

Programme	Dateien
-----	-----
Startadresse (xxxx)	Startadresse (0000)
Endadresse (xxxx)	Endadresse (0000)
Programmname (16 Zeichen)	Dateiname (16 Zeichen)
Füllzeichen (für Prg. Name)	Füllzeichen (für Dat. Name)

Ein Block besteht aus: (Programme und Dateien)

ca. 2.00 Sekunden Vorspann

9 BYTES Count Down (\$89 \$88 \$87 \$86 \$85 \$84 \$83 \$82 \$81)
beim ersten Block

(\$09 \$08 \$07 \$06 \$05 \$04 \$03 \$02 \$01)
bei den Wiederholungen

Daten

Prüfsumme (EXOR Prüfsumme über die gesamten Daten)

Endmarkierung (LLKK KKKK KKKK KKKK KKKK)

ca. 0.16 Sekunden Nachspann

Wie schon gesagt, ist das Aufzeichnungsverfahren bei allen Commodore Geräten gleich. Das Problem mit dem Datenaustausch zwischen Commodore 64 und VC-20 liegt nur in den unterschiedlichen Frequenzen. Die Taktzahl des VC-20 liegt über der des CBM, und die Taktzahl des CBM wieder über der des Commodore 64. Das bedeutet für Sie, daß Sie zwar auf den großen Commodore-Geräten die Programme des VC-20 und des Commodore 64

laufen lassen können, nicht aber ein VC-20 Programm direkt auf dem Commodore 64 und umgekehrt. Falls Sie aber Cassetten zwischen den beiden Rechnern austauschen wollen, so müssen Sie den Umweg über einen großen CBM machen. Der Vorgang könnte dabei so aussehen:

Sie haben eine Cassette, auf der eins oder mehrere VC-20 Programme (oder Dateien) enthalten sind. Diese Programme (Dateien) wollen Sie auf Ihren Commodore 64 übertragen. Dazu nehmen Sie eine normale Datasette, schließen diese an einen großen CBM-Rechner an und laden das erste Programm (oder die erste Datei) wie gewohnt. Dann nehmen Sie eine neue Cassette, und tauschen diese mit der VC-20 Cassette im Recorder aus. Nun speichern Sie wieder Ihr Programm (oder die Datei) mit den üblichen Befehlen aufs neue Band. Falls Sie noch mehrere Programme (oder Daten) haben, beginnen Sie wieder von vorne.

Nachdem alle Programme übertragen, oder sagen wir besser konvertiert, sind, haben Sie eine Cassette, die sowohl auf dem VC-20 als auch auf dem Commodore 64 geladen werden kann. Unter Umständen müssen dann noch ein paar Änderungen im Programm vorgenommen werden, wie Bildschirmanpassung, verschiedene POKE's und dergleichen, aber das versteht sich ja schon von selbst.

Leider ist auch uns zur Zeit noch keine softwareseitige Lösung des Problems der Programmkonvertierung zwischen VC-20 und Commodore 64 bekannt, aber vielleicht haben Sie ja schon eine Idee. Über derartige Anregungen sind wir immer sehr erfreut.

Ein Trostpflaster aber noch zum Schluß. Programme, die zum Beispiel auf den Commodore Rechnern der 3000 oder 4000 Serie geschrieben wurden, lassen sich direkt laden - und welcher anderer Computer kann dann schon auf eine derartige Menge an Software zurückgreifen ?

8.2 DAS PRINZIP DER DATEIVERWALTUNG: SEQUENTIELLE DATEIEN

Wir haben uns eben ausführlich mit der "Geschichte" der Datenträger beschäftigt. Wenden wir uns in unseren Betrachtungen jetzt nur den beiden letztgenannten Datenträgern, also Magnetband und -Platte zu. An Hand dieser beiden Geräte lassen sich leicht die unterschiedlichen Methoden des Dateizugriffs erklären. In diesem Kapitel wollen wir uns nun zunächst nur der sequentiellen Methode zuwenden.

Sequentiell bedeutet übersetzt soviel wie "nacheinander". Genauso, also nacheinander befinden sich die einzelnen Datensätze in der Datei. Das kann man vergleichen mit der Aufnahme eines Musikstücks mit Hilfe eines Cassettenrecorders. Sie spulen das Band an eine bestimmte Stelle, und betätigen dann die Aufnahmetasten des Recorders. Dann wird das Musikstück aufgezeichnet. Wenn Sie ein bestimmtes Musikstück wieder abspielen wollen, so müssen Sie in der Regel erst ganz an den Anfang des Bandes spulen, um dann, beim Vorspulen, am Zählwerk ablesen zu können, welches Musikstück Sie gerade "passieren".

Ungefähr so kann man sich auch die Aufzeichnung von Daten vom Computer auf Band vorstellen. Wenn Sie also Daten aufzeichnen wollen, die Sie später anhand einer bestimmten Zählerposition wiederfinden wollen, müssen Sie zunächst exakt die Zählerposition feststellen. Diese stimmt natürlich nur dann, wenn Sie das Band vorher zurückgespult und anschließend die Zählwerk-Rückstell Taste betätigt haben. Erst dann können Sie mit der eigentlichen Aufzeichnung beginnen. Dieses etwas umständliche Verfahren ermöglicht es Ihnen später zumindest, mit Hilfe der schnellen Vor- und Rücklauftasten schneller an Ihre Daten zu kommen. Problematisch wird es aber trotzdem dann, wenn Sie innerhalb einer Datei nach einem bestimmten Datensatz suchen. Wenn Sie eine Datei voller Adressen haben, und Sie suchen nach dem Namen MÜLLER, so kann es natürlich vorkommen, daß Sie mehrere MÜLLER in der selben Datei haben. Oftmals können Sie aber beim Abspeichern der Daten nicht jedesmal den Zählwerksstand notieren (ausserdem

wollen wir ja gerade auf solche manuellen Arbeiten verzichten, sonst können wir gleich Karteikarten nehmen). Wir müssen also etwas anders vorgehen. Dazu spulen wir das Band wieder ganz an den Anfang zurück, öffnen die Datei zum Lesen (s.o.) und gehen nun Satz für Satz durch, bis wir den richtigen MÜLLER gelesen haben. Das bedeutet natürlich einen gewissen Aufwand - bei 2000 Datensätzen in einer Datei kann man vielleicht zwischendurch gemütlich Kaffee trinken gehen. Aber trotzdem kann man auch mit so etwas einfachem wie einem Cassettenrecorder durchaus eine kleine Dateiverwaltung betreiben, zumal dieser Cassettenrecorder eine wirklich preisgünstige Möglichkeit ist.

Wer nur einen Cassettenrecorder besitzt, aber trotzdem schnell und effizient Daten verwalten möchte, der sollte folgendes Verfahren anwenden: Gestalten Sie Ihre Dateien so, daß diese grundsätzlich in den freien Speicher des Commodore 64 passen. Bevor Sie Daten in Ihrer Datei ändern, löschen, hinzufügen, oder nur einfach auf dem Bildschirm anzeigen wollen, laden Sie die komplette Datei in den Speicher. Jetzt sind Sie bei Dateizugriffen nicht mehr von der geringen Geschwindigkeit des Recorders abhängig, sondern können die volle Rechengeschwindigkeit Ihres Commodore ausnutzen. Nach der Arbeit mit Ihrer Datei speichern Sie die komplette Datei wieder auf Cassette ab. Dieses einfache und praktische Verfahren können Sie selbstverständlich auch bei größeren Dateien anwenden. Spalten Sie Ihre Datenbestände einfach auf mehrere kleine Dateien auf. So können Sie z.B. eine umfangreiche Adressdatei durchaus auch nach Namen aufspalten in A-C, D-F etc., und so Dateien erhalten, die in den Speicher des Commodore 64 passen. Mit etwas Geschick und Organisation läßt sich so auch mit einem Recorder eine ganze Menge Komfort in der Dateiverwaltung erreichen.

Wenn man sich den Vorgang der Datenspeicherung bildlich vorstellt, wird man leicht einsehen, warum auf einem Kassettenrecorder nur eine sequentielle Datenspeicherung möglich ist. Alle Daten werden hintereinander abgespeichert und auch hintereinander wieder in den Computer eingelesen.

Auch bei der Diskette steht die sequentielle Datei als Dateiart zur Verfügung, allerdings sind hier die Möglichkeiten ihrer Handhabung größer als beim Cassettenrecorder. Auf sequentielle Dateien kann ohne Suchen schnell und direkt zugegriffen werden. Über das Disketten-Inhaltsverzeichnis, in dem alle auf einer Diskette befindlichen Programme und Daten verzeichnet sind, erhält der Commodore 64 sofort den Platz, an dem sich die Datei befindet. Damit entfällt das umständliche Suchen, wie es beim Cassettenrecorder notwendig ist.

Wie behandeln wir nun eine sequentielle Datei auf dem Commodore 64 ? Zunächst muß die Datei geöffnet werden (s.o.). Dazu sind Dateinummer, Gerätenummer, Kanalnummer und Name notwendig. Wenn diese Datei so geöffnet wurde, kann mit einem Befehl in die Datei geschrieben oder aus ihr gelesen werden, aber nie beides auf einmal !

Allerdings kann nicht ohne weiteres in eine bestehende Datei geschrieben werden. Wenn Sie z.B. die Datei "CBM 64 DATEI" mit dem Befehl

```
OPEN 2, 8, 2, "O:CBM 64 DATEI,S,W"
```

eröffnen und eine Datei mit diesem Namen befand sich bereits auf der Diskette, dann erhalten Sie eine FILE EXISTS Fehlermeldung. Der Befehl muß deshalb modifiziert werden, indem vor die Laufwerksnummer das Zeichen "@" gesetzt wird (in der Fachsprache auch als "Klammeraffe" bekannt). Der Befehl lautet dann

```
OPEN 2, 8, 2, "@O:CBM 64 DATEI,S,W"
```

und bewirkt, daß die betreffende Datei überschrieben wird. Dabei gehen die bisher in dieser Datei auf der Diskette gespeicherten Daten verloren !

Wichtig ist deshalb: Auch bei einer Diskette können in einer bestehenden, sequentiellen, Datei keine Daten geändert werden.

Stets muß zum Ändern von Daten die sequentielle Datei komplett in den Speicher gelesen und nach Durchführung der Änderung wieder neu auf Diskette abgespeichert werden. Wer trotz der Möglichkeit des direkten Zugriffs seine Dateiverwaltung mit der Diskette auf sequentielle Dateien aufbauen möchte, der sollte das weiter oben für Cassettenrecorder beschriebene Verfahren anwenden.

Allerdings bieten sequentielle Dateien auf der Diskette gegenüber dem Cassettenrecorder außer der erheblich höheren Geschwindigkeit noch einen weiteren Vorteil, nämlich das Anbinden (APPEND) an eine Datei. Dieses Anbinden hat einen großen Vorteil: Sie müssen nicht alle Daten wieder in den Commodore 64 lesen und dann wegschreiben, sondern Sie können durch einfaches ändern des OPEN-Befehls den Commodore 64 dazu veranlassen, alle Daten die nun auf die Diskette geschrieben werden sollen, an eine vorhandene Datei anzubinden. Der OPEN-Befehl lautet dann:

```
OPEN 2, 8, 2, "O:CBM 64 DATEI,S,A"
```

Nun werden alle Daten, die jetzt noch eingegeben werden, an die Datei angehängt. Dadurch können Sie zum Beispiel eine bestehende Adressdatei leicht erweitern, ohne die anderen Daten zu überschreiben bzw. erst in eine neue Datei kopieren zu müssen.

Bei einem Cassettenrecorder ist die Möglichkeit des Anbindens an eine sequentielle Datei leider nicht gegeben. Sie müssen also stets erst alle Daten in den Commodore 64 einlesen, dann modifizieren oder erweitern und dann erst wieder auf die Cassette abspeichern. Verstehen Sie nun, warum wir sagten, daß eine Dateiverwaltung mit der Diskette weitaus sinnvoller ist als auf Cassette ?

Im Folgenden finden Sie anhand von Musterprogrammen eine kleine Dateiverwaltung mit sequentiellen Dateien für Cassettenrecorder und Diskettenlaufwerk. Die einzelnen Programme können Sie leicht für Ihre individuelle Anwendung modifizieren. Zunächst nun die Version für Cassettenrecorder.

1. Schreiben der Daten

```
10 REM *****
20 REM SCHREIBEN VON NAME UND VORNAME AUF BAND
30 REM VERSION FUER DATASETTE / COMMODORE 64
40 REM *****
50 PRINT CHR$(147): REM BILDSCHIRM LOESCHEN
52 PRINT "OEFFNEN DER DATEI ZUM SCHREIBEN"
54 PRINT
56 OPEN 1,1,1,"CBM 64 DATEI"
60 INPUT "NAME      : ";NA$
70 INPUT "VORNAME  : ";VN$
80 PRINT
90 PRINT "SCHREIBEN - NAME = ";NA$
100 PRINT "          - VORNAME = ";VN$
110 PRINT
120 PRINT#1,NA$
130 PRINT#1,VN$
140 PRINT
150 INPUT "WEITER (J/N) ";JN$
155 PRINT
160 IF JN$="J" THEN GOTO 60
170 IF JN$="N" THEN GOTO 200
180 PRINT "FALSCHE EINGABE !"
190 GOTO 140
200 CLOSE 1
210 END
```

Dieses Programm ist dafür gedacht, eine beliebige Anzahl von Namen und Vornamen auf Band zu speichern. Es braucht nicht darauf hingewiesen zu werden, daß dieses Programm nur mit einem Cassettenrecorder angewendet werden kann. Als nächstes Programm folgt nun das "Gegenstück" zu dem Vorherigen. Dieses Programm liest die Daten in den Commodore 64 ein und zeigt Sie auf dem Bildschirm (oder Drucker) an. Zuvor müssen Sie aber das Band wieder an den Anfang zurückspulen. Wenn dies geschehen ist, können Sie das Programm anwenden.

2. Lesen der Daten

```
10 REM *****
20 REM LESEN VON NAME UND VORNAME VON BAND
30 REM VERSION FÜR DATASETTE / COMMODORE 64
40 REM *****
50 PRINT CHR$(147): REM BILDSCHIRM LOESCHEN
52 PRINT "OEFFNEN DER DATEI ZUM LESEN"
54 PRINT
56 OPEN 1,1,0,"CBM 64 DATEI"
60 INPUT#1,NA$
70 INPUT#1,VN$
80 IF ST AND 64 THEN GOTO 130: REM DATEIENDE ?
90 PRINT "LESEN - NAME = ";NA$
100 PRINT "      - VORNAME = ";VN$
110 PRINT
120 GOTO 60
130 PRINT "DATEIENDE - NAME = ";NA$
140 PRINT "      - VORNAME = ";VN$
150 CLOSE 1
160 END
```

Dieses Programm liest nun alle Daten, die mit dem ersten Programm auf Cassette geschrieben wurden und gibt Sie auf den Bildschirm aus. Wollen Sie dagegen Ihre Daten statt auf dem Bildschirm auf einem Drucker ausgeben, so müssen Sie ein paar Zeilen ändern:

```
58 OPEN 4,4
90 PRINT#4,"LESEN - NAME = ";NA$
100 PRINT#4,"      - VORNAME = ";VN$
110 PRINT#4
130 PRINT#4,"DATEIENDE - NAME = ";NA$
140 PRINT#4,"      - VORNAME = ";VN$
155 CLOSE 4
```

Die anderen Zeilen bleiben unverändert. Nun gibt es noch eine

andere Möglichkeit, nämlich das Anfügen von Daten. Wie oben besprochen, kann man bei der Verwendung des Cassettenrecorders als Datenträger zusätzliche Daten nicht einfach anhängen. Die Datei muß also zunächst vollständig in den Speicher eingelesen werden. Anschließend spulen Sie das Band zurück, eröffnen die Datei zum Schreiben und schreiben die eben gelesenen Daten wieder auf Band. Zum Schluß hängen Sie die neuen Daten an. Nach dem selben Verfahren lassen sich dann natürlich auch die einzelnen Daten ändern oder löschen.

3. Anfügen von Daten

```
10 REM *****
20 REM ANFUEGEN VON NAME UND VORNAME AN BAND
30 REM VERSION FUER DATASETTE / COMMODORE 64
40 REM *****
50 PRINT CHR$(147): REM BILDSCHIRM LOESCHEN
52 PRINT "OEFFNEN DER DATEI ZUM LESEN"
54 PRINT
56 OPEN 1,1,0,"CBM 64 DATEI"
60 DIM NA$(100),VN$(100): I=1: REM MAXIMAL 100 NAMEN
70 INPUT#1,NA$: NA$(I)=NA$
80 INPUT#1,VN$: VN$(I)=VN$
90 IF ST AND 64 THEN GOTO 130
100 IF I=100 THEN GOTO 130
110 I=I+1
120 GOTO 70
130 ENDE=I
135 PRINT
140 PRINT "SPULEN SIE DAS BAND ZURÜCK !"
150 PRINT
160 INPUT "FERTIG (J/N) ";JN$
170 IF JN$="N" THEN GOTO 130
180 IF JN$="J" THEN 210
190 PRINT "FALSCHE EINGABE !"
200 GOTO 150
210 PRINT "OEFFNEN DER DATEI ZUM ANFUEGEN"
```

```

220 PRINT
230 OPEN 1,1,1,"CBM 64 DATEI"
240 FOR I=1 TO ENDE
250 PRINT#1,NA$(I)
260 PRINT#1,VN$(I)
270 NEXT I
280 PRINT "ANFÜGEN DER DATEN:"
290 PRINT
300 INPUT "NAME      : ";NA$
310 INPUT "VORNAME  : ";VN$
320 PRINT
330 PRINT "SCHREIBEN - NAME = ";NA$
340 PRINT "          - VORNAME = ";VN$
350 PRINT
360 PRINT#1,NA$
370 PRINT#1,VN$
380 PRINT
390 INPUT "WEITER (J/N) ";JN$
400 IF JN$="J" THEN GOTO 300
410 IF JN$="N" THEN GOTO 440
420 PRINT "FALSCH EINGABE !"
430 GOTO 380
440 CLOSE 1
450 END

```

Jetzt haben Sie schon eine eigene, kleine Adressverwaltung. Es fehlen zwar noch die Angaben über die Adressen, aber jeder, der ein wenig Erfahrung in der Programmierung hat, müßte sich die Datei um die entsprechenden Angaben erweitern bzw. das Programm ändern können.

Nun wollen wir uns etwas ausführlicher mit der sequentiellen Dateiverwaltung auf der Floppy beschäftigen. Auch hier wollen wir wieder die drei Beispiele, die wir schon bei der Datasette besprochen haben anwenden. So werden Ihnen auch schnell der Unterschied, aber auch die Gemeinsamkeiten in der Anwendung deutlich.

1. Schreiben der Daten

Diese Routine ähnelt natürlich der Routine, die wir schon bei der Benutzung des Cassettenrecorders kennengelernt haben. Dies ist ja auch ganz verständlich, da das Prinzip der sequentiellen Datei auch auf der Floppy dasselbe ist. Die Routine lautet also:

```
10 REM *****
20 REM SCHREIBEN VON NAME UND VORNAME AUF DISK
30 REM VERSION FUER FLOPPY 1541 / COMMODORE 64
40 REM *****
50 PRINT CHR$(147): REM BILDSCHIRM LOESCHEN
52 PRINT "OEFFNEN DER DATEI ZUM SCHREIBEN"
54 PRINT
56 OPEN 2,8,2,"CBM 64 DATEI,S,W"
60 INPUT "NAME      : ";NA$
70 INPUT "VORNAME  : ";VN$
80 PRINT
90 PRINT "SCHREIBEN - NAME = ";NA$
100 PRINT "          - VORNAME = ";VN$
110 PRINT
120 PRINT#2,NA$
130 PRINT#2,VN$
140 PRINT
150 INPUT "WEITER (J/N) ";JN$
155 PRINT
160 IF JN$="J" THEN GOTO 60
170 IF JN$="N" THEN GOTO 200
180 PRINT "FALSCH EINGABE !"
190 GOTO 140
200 CLOSE 2
210 END
```

Genau wie das schon oben besprochene Programm zum Schreiben auf den Cassettenrecorder, so schreibt dieses Programm eine beliebige Anzahl von Namen und Vornamen auf die Diskette, und das in sequentieller Form. Natürlich geht das nur solange, bis die

Diskette, oder besser gesagt der Raum für eine Datei voll ist. Um solch eine Diskette voll zu schreiben, benötigt man zwar schon eine ganze Menge an Daten, man sollte aber trotzdem schon bei der Programmierung darauf achten, daß es zu keinem Fehler, oder gar Programmabsturz kommt, weil die Diskette plötzlich voll ist. Damit Sie einen gewissen Eindruck von der Kapazität der Diskette bekommen, wollen wir an dieser Stelle einmal ein kleines Rechenbeispiel zeigen:

Die Floppy 1541 hat insgesamt Platz für 174.848 Bytes (Zeichen). Davon können für die Dateien benutzt werden:

Sequentielle Dateien : 168.656 Zeichen
Relative Dateien : 167.132 Zeichen

Es können maximal 144 Programme (Dateien) abgespeichert werden.

Stellen Sie sich nun vor, Sie wollen eine komplette Adressverwaltung programmieren. Dazu benötigen Sie beispielsweise folgende Daten:

BEZEICHNUNG	LÄNGE
-----	-----
Nummer	3
Vorname	20
Nachname	20
Strasse, Nr.	25
PLZ	4
Wohnort	25
Land	3
Telefon	16
Bemerkung	50
-----	-----
Summe:	166

Ihr Datensatz ist also 166 Zeichen lang. Dazu kommen jetzt noch die Abschlußzeichen (CHR\$(13)). Wir müssen also für jede

Bezeichnung noch ein Zeichen mehr rechnen. Das ergibt dann insgesamt 175 Zeichen. Wieviele Datensätze können wir nun auf der Diskette abspeichern ?

Die Rechnung, die wir hierzu benötigen, sieht also folgendermaßen aus:

MAX: BYTES FÜR SEQUENTIELLE DATEI / LÄNGE EINES RECORDS

oder in unserem Beispiel:

$$168.656 / 175 = 963,74857$$

Da wir aber schlecht 963,74857 Datensätze anlegen können, und ein bisschen Platz auf der Diskette nie schadet, würden wir in diesem Fall 950 Datensätze anlegen können. Diese Anzahl reicht für die üblichen Anwendungen auch normalerweise aus. Wenn Sie aber trotzdem noch mehr Datensätze anlegen möchten, so müssen Sie entweder das Programm in der Weise umschreiben, daß Sie mehrere Daten-Disketten benutzen können, oder aber eine größere Floppy verwenden. So erhöht sich zum Beispiel die Kapazität bei Verwendung einer Commodore 8250 Floppy um den Faktor 6 pro Laufwerk. Das würde also bedeuten, daß Sie bei der Commodore 8250 Floppy mehr als 5500 Adressen speichern könnten.

Und diese Möglichkeit haben Sie sogar bei Ihrem Commodore 64. Alles was Sie dafür benötigen, ist die Floppy selber und den notwendigen IEC-Bus. Zusammen mit dem DATA BECKER IEC-Bus haben Sie gleichzeitig auch noch die Diskettenbefehle des BASIC-4, also des BASIC, das auch die großen Commodore sprechen.

2. Lesen der Daten

Nun aber zum Lesen der Daten. Auch hier ist das Programm quasi identisch mit der Cassetten-Version. Der Übersicht halber wollen wir aber auch dieses Programm noch einmal vorstellen:

```

10 REM *****
20 REM LESEN VON NAME UND VORNAME VON DISK
30 REM VERSION FUER FLOPPY 1541 / COMMODORE 64
40 REM *****
50 PRINT CHR$(147): REM BILDSCHIRM LOESCHEN
52 PRINT "OEFFNEN DER DATEI ZUM LESEN"
54 PRINT
56 OPEN 2,8,2,"CBM 64 DATEI,S,R"
60 INPUT#2,NA$
70 INPUT#2,VN$
80 IF ST AND 64 THEN GOTO 130: REM DATEI ENDE ?
90 PRINT "LESEN - NAME = ";NA$
100 PRINT "      - VORNAME = ";VN$
110 PRINT
120 GOTO 60
130 PRINT "DATEIENDE - NAME = ";NA$
140 PRINT "      - VORNAME = ";VN$
150 CLOSE 2
160 END

```

Wie Sie sehen, gibt es auch in diesem Programm keine bedeutenden Unterschiede zur Cassettenversion. Wir sagten ja auch schon, daß es bei der sequentiellen Bearbeitung von Dateien keine größeren Abweichungen in der Programmierung gibt. Der einzige Unterschied besteht nur im Anfügen von Daten. Dies geht mit einer Floppy bedeutend einfacher als bei der Cassette.

3. Anfügen von Daten

```

10 REM *****
20 REM ANFUEGEN VON NAME UND VORNAME AN DATEI
30 REM VERSION FUER FLOPPY 1541 / COMMODORE 64
40 REM *****
50 PRINT CHR$(147): REM BILDSCHIRM LOESCHEN
52 PRINT "OEFFNEN DER DATEI ZUM ANFUEGEN"
54 PRINT
56 OPEN 2,8,2,"CBM 64 DATEI,S,A"

```

```

60 INPUT "NAME      : ";NA$
70 INPUT "          : ";VN$
80 PRINT
90 PRINT "SCHREIBEN - NAME = ";NA$
100 PRINT "          - VORNAME = ";VN$
110 PRINT
120 PRINT#2,NA$
130 PRINT#2,VN$
140 PRINT
150 INPUT "WEITER (J/N) ";JN$
155 PRINT
160 IF JN$="J" THEN GOTO 60
170 IF JN$="N" THEN GOTO 200
180 PRINT "FALSCH EINGABE !"
190 GOTO 140
200 CLOSE 2
210 END

```

Wie Sie sicherlich schon festgestellt haben werden, ähnelt diesem Programm sehr stark dem Programm zum Schreiben der Daten - mit einer Ausnahme: Der OPEN-Befehl wurde von

```
OPEN 2,8,2,"CBM 64 DATEI,S,W"
```

auf

```
OPEN 2,8,2,"CBM 64 DATEI,S,A"
```

verändert. Dies bewirkt, daß alle Daten, die nun mit PRINT#2 in die Datei geschrieben werden, nicht die anderen Daten überschreiben, sondern an die schon bestehende Datei angehängt werden. Diese Besonderheit ermöglicht also eine relativ bequeme Handhabung von sequentiellen Dateien.

Lassen Sie uns zum Schluß dieses Kapitels sinnvolle Anwendungsbereiche für sequentielle Dateien aufzeigen. Für eine Dateiverwaltung sind sie, wenn es auf schnellen Zugriff und

leichte Änderungsmöglichkeit von Daten ankommt, nur sehr bedingt geeignet. Man benutzt deshalb sequentielle Dateien in der EDV hauptsächlich dann, wenn eine Datei für einen ganz bestimmten Zweck einmal in einer ganz bestimmten Form erstellt werden soll. Ein Beispiel wäre zum Beispiel Datenträgeraustausch mit anderen Computern. Sequentielle Dateien lassen sich, sofern sie im gleichen Aufzeichnungsformat gespeichert sind (ASCII), im Prinzip von jedem anderen Computer aus leicht lesen. Bei relativen bzw. Direktzugriffsdateien ist dies wegen der oft von Betriebssystem zu Betriebssystem unterschiedlichen Struktur solcher Dateien meist nicht möglich. Ein anderes Beispiel sind Protokolldateien, die fortlaufend erstellt werden und in denen Änderungen nicht vorkommen oder sogar nicht vorkommen dürfen, z.B. das Buchungsjournal bei der Finanzbuchhaltung.

8.3 KOPIEREN VON DATEIEN MIT EINEM UND ZWEI LAUFWERKEN

Wie oben schon beschrieben, gibt es verschiedene Möglichkeiten, um bestehende sequentielle Dateien zu erweitern oder zu ändern/löschen. Sequentielle Dateiverwaltung kann zwar einfach sein - aber ist nicht nur als primitive Möglichkeit anzusehen, Daten abzuspeichern und wieder einzulesen.

Außerdem ist es oftmals sehr sinnvoll oder sogar notwendig Daten oder Dateien zu duplizieren, damit auch nach einer Bearbeitung der entsprechenden Datei alle Daten noch im originalzustand vorliegen, bzw. damit für den Fall eines Datenverlustes eine Kopie vorliegt.

Wir wollen uns zunächst mit der Kopie von Dateien befassen. In unserem Beispiel nehmen wir an, daß es sich um eine Datei handelt, die sequentiell abgespeichert wurde. Um diese Datei nun zu kopieren, gibt es mehrere Möglichkeiten.

Zunächst könnten wir die gesamte Datei in den Arbeitsspeicher des Commodore 64 lesen, um diese Datensätze dann in die neue Datei zu kopieren. Das erfordert allerdings im Normallfall einen sehr großen Arbeitsspeicher (die Diskette fast ja bekanntlich ca. 170.000 Zeichen, der Commodore 64 dagegen unter BASIC + Programm nur ca. 30.000) oder ist beschränkt auf kleinere Dateien. Außerdem wäre ein "Mittelweg" denkbar, bei dem die Datensätze in Blöcken zu einer bestimmten Anzahl eingelesen werden und dann eben in diesen Blöcken auch wieder abgespeichert werden. Wir wollen uns aber an die einfache Methode halten und jeden Satz einzelnen einlesen und wegschreiben.

Hierzu müssen wir eine zweite Datei erstellen, die nach dem Kopiervorgang identisch mit der Ursprungsdatei ist. Das einzige Problem was hier auftritt ist, daß wir von Anfang an wissen müssen, wieviele Felder jeder Datensatz hat. Um das Programm noch etwas komfortabler zu machen, wollen wir auch noch anzeigen, welchen Datensatz wir gerade lesen, und wie die Felder lauten.

```

50000 AD$="ORIGINALDATEI": REM NAME DER DATEI (BELIEBIG WAEHLB.)
50010 ND$="NEUEDATEI": REM NAME DER NEUEN DATEI (      "      )
50011 ND$=" :"+ND$+",S,W": REM NEUE DATEI ZUM SCHREIBEN
50020 INPUT "WIEVIELE FELDER PRO DATENSATZ ";AF$
50030 AF=VAL(AF$)
50040 DIM FB$(AF): REM DIMENSIONIERUNG DER FELDBEZEICHNUNGEN
50045 DIM DF$(AF): REM      "      " DATENFELDER
50050 FOR I=1 TO AF: REM EINGEBE ALLER FELDTITEL
50060 INPUT FB$(I)
50070 NEXT I
50080 PRINT
50090 PRINT "KOPIERVORGANG LAEUFT !"
50100 OPEN 1,8,2,AD$: REM DATEI ZUM LESEN OEFFNEN
50110 OPEN 2,8,3,ND$: REM DATEI ZUM SCHREIBEN OEFFNEN
50112 DS=1: REM BEGINNEN MIT DATENSATZ 1
50115 PRINT "LESEN DATENSATZ NR.";DS: PRINT
50120 FOR I=1 TO AF: REM ALLE DATENFELDER EINLESEN
50130 INPUT#1,DF$(I)
50140 PRINT FB$(I);" : ";DF$(I)
50145 DZ=ST: REM DZ = DATEISTATUS
50150 NEXT I
50160 PRINT
50170 PRINT "SCHREIBEN DATENSATZ NR. ";DS: PRINT
50180 FOR I=1 TO AF: REM ALLE DATENFELDER SCHREIBEN
50190 PRINT#2,DF$(I)
50200 NEXT I
50210 PRINT
50220 IF DZ AND 64 THEN 51000: REM ENDE DER DATEI ?
50230 DS=DS+1: REM NAECHSTER DATENSATZ
50240 GOTO 50115
51000 PRINT "ALLE DATENSAETZE UEBERTRAGEN !"
51010 PRINT
51020 CLOSE 2: REM SCHLIESSEN DER DATEIEN
51030 CLOSE 1
51040 PRINT "ENDE."
51050 END

```

Mit diesem Programm können Sie nun leicht Ihre vorhandenen sequentiellen Dateien kopieren, sofern Sie den Aufbau - genauer gesagt die Anzahl der Datenfelder pro Datensatz - kennen.

Ein Problem hat diese Routine aber doch. Wenn Sie nämlich eine sehr große Datei kopieren wollen, werden Sie schon sehr bald an die Grenzen der Floppy stoßen. Sie werden einsehen, daß Sie Dateien die z.B. 100.000 Zeichen lang sind (ca. 100 KB) mit diesem Programm nicht so einfach kopieren können, da Sie ja eine Zielfeile erzeugen, die nicht nur den selben Aufbau der Ursprungsdatei hat, sondern auch genau die selbe Größe.

Um aber nun auch größere Dateien kopieren zu können, müssen Sie entweder mit 2 Laufwerken arbeiten, oder aber mit 2 verschiedenen Disketten.

Am leichtesten und sichersten ist solch eine Duplizierung mit 2 Laufwerken zu bewerkstelligen. Hierzu muß eines Ihrer Laufwerke als Einheit 8 und das andere als Einheit 9 definiert sein. Dieses geht sowohl softwaremäßig umzustellen, als auch durch einen kleinen Eingriff in eine der Floppys, den der Techniker leicht durchführen kann.

Wenn Sie nun eine Floppy als Gerät 9 haben, so können Sie das oben vorgegebene Programm derart abändern, daß auch große Dateien kopiert werden können.

50110 OPEN 2,2,3,ND\$: REM DATEI ZUM SCHREIBEN OEFFNEN

Nun werden Daten von Laufwerk 8 gelesen, auf dem Bildschirm dargestellt und auf Laufwerk 9 wieder gespeichert.

Noch ein Tip am Rande. Sehr viele Programmierer benutzen bei Ihren Dateien den Datensatz 0 (oder 1) oder eine zweite Datei, als speziellen Datensatz (oder Datei) mit Informationen über den Aufbau der Datei. Es wäre zum Beispiel sehr nützlich, wenn Sie als erste Information die Feldanzahl und die Datensatzanzahl

gespeichert hätten. Dabei würde dann eine manuelle Eingabe entfallen. Außerdem wüßten Sie nun auch zu jedem Zeitpunkt wieviele Datensätze noch zu lesen sind, bzw. wie groß die Date überhaupt ist. Wenn Sie diese beiden Variablen gespeichert haben (in der Datei ORIGINALDATEI) müssen Sie das Programm natürlich noch etwas abändern.

```
50012 OPEN 1,8,2,AD$: REM ORIGINALDATEI ZUM LESEN OEFFNEN
50013 INPUT#1,AF$: REM ANZAHL DER FELDER
50014 INPUT#1,AD$: REM ANZAHL DER DATENSÄTZE
50015 CLOSE 1
```

Zeile 50020 löschen.

```
50035 AD=VAL(AD$)
50111 INPUT#1,AF$: INPUT#2,AD$: PRINT#2,AF$: PRINT#2,AD$
50112 FOR DS=1 TO AD
```

Zeilen 50145 und 50220 löschen.

```
50230 NEXT DS
```

Zeile 50240 löschen.

Eine derart parametrisierte Datei ist wesentlich leichter und übersichtlicher zu bearbeiten.

Das Anfügen von Datensätzen geht nach dem selben Prinzip wie das Kopieren. Zunächst wird die Datei mit dem oben gezeigten Programm kopiert, dann wird Datei 2, also die Neuedatei nicht geschlossen, sondern mit den üblichen PRINT#-Befehlen weiter beschrieben. Wenn diese Datei so erweitert wurde, kann Sie wieder auf die ursprüngliche Datei zurückkopiert werden.

8.4 SO GEHT'S SCHNELLER: RELATIVE DATEIEN

Die größeren Commodore Computer mit ihrem BASIC 4.0 und alle, die auf ihrem Commodore 64 MAXI oder den DATA BECKER IEC-BUS einsetzen, haben mit den relativen Dateien eine wesentlich komfortablere und leistungsfähigere Dateiarart für die Datenspeicherung zur Verfügung. Bei einer relativen Datei trägt jeder Datensatz eine Nummer, die gleichzeitig seinen Platz auf der Diskette relativ zum Anfang der Datei angibt. So können Sie z.B. bei einer relativen Datei eine Dateiverwaltung mit Direktzugriff aufbauen, so gibt es grundsätzlich zwei Möglichkeiten:

1.) Sie nutzen das Ordnungskriterium der relativen Datei, nämlich die fortlaufend vergebene Nummer, analog als Zugriffsschlüssel für Ihre Datei. So können Sie z.B. in einer Rechnungsdatei die Satznummer mit der Rechnungsnummer gleichsetzen. So wäre über die Rechnungsnummer stets ein schneller, direkter Zugriff auf die jeweilige Rechnung möglich. Analog gilt das Gleiche auch für Artikelnummern und andere Schlüssel, die sich willkürlich vergeben lassen.

2.) Sie bauen eine entsprechende Tabelle auf, in der Satznummer und Schlüssel gegenübergestellt sind. Haben Sie zum Beispiel Ihre Adressdatei nach Namen geordnet und wollen die Adresse mit dem Namen MÜLLER herausfinden, so suchen Sie zunächst aus der Tabelle die Satznummer heraus, die zum Namen MÜLLER gehört, und greifen dann direkt auf diesen Datensatz zu. Dieses Verfahren ist erheblich schneller und eleganter als das Lesen einer sequentiellen Datei bis zum Auffinden des Namens.

Leider kann aber der Besitzer eines serienmäßigen Commodore 64, der nicht z.B. über MAXI 64, IEC-BUS 64 oder MASTER 64 um die Diskettenbefehle des BASIC 4.0 ergänzt wurde, normalerweise mit den so leistungsfähigen relativen Dateien nicht arbeiten. Die Floppy VC-1541 kennt zwar von ihrem Betriebssystem her, dem DOS,

die relativen Dateien, doch sind die notwendigen Befehle im BASIC des serienmäßigen Commodore 64 nicht enthalten.

Wir möchten Ihnen nachfolgend einen Weg zeigen, wie man trotzdem relative Dateien mit dem Commodore 64 nutzen kann.

Es gibt die Möglichkeit, durch verschiedene CHR\$-Folgen dem Computer mitzuteilen, welchen Datensatz man nun lesen oder schreiben möchte. Der ganze Vorgang besteht also aus zwei verschiedenen Teilen:

1.) Öffnen der relativen Datei mit dem gewohnten OPEN-Befehl:

OPEN Dateinummer, Geräteadresse, Kanalnummer, "Name,L,"+CHR\$(Länge)

Der erste Teil dieses OPEN-Befehls ist uns schon von der sequentiellen Datei her bekannt. Nach der Angabe des Namens folgt allerdings nun ein 'L'. Dieses 'L' steht für LÄNGE - die Floppy 1541 weiß nun, daß eine relative Datei eröffnet werden soll. Danach folgt ein sehr wichtiger CHR\$-Befehl. Dieser Befehl gibt nun die Länge eines Datensatzes an. Wenn Sie sich noch an unsere sequentielle Dateiverwaltung erinnern, mit der wir eine Adressenverwaltung bearbeiten wollten, so müßten wir hier die Länge 175 angeben. Der Commodore 64, bzw. das Betriebssystem der Floppys erlaubt eine maximale Datensatzlänge von 254 Zeichen. Wenn ein Datensatz mehr als 254 Zeichen haben soll, so muß entweder eine andere Datei eröffnet, und der eine Datensatz in zwei Datensätze aufgeteilt werden, oder aber Sie schreiben in die selbe Datei, und betrachten jeden zweiten Datensatz als den ersten Teil des 'Groß-Datensatzes'.

2.) Die Positionierung des Datensatz-Zeigers:

PRINT # Kanalnummer,"P"+CHR\$(Kanalnummer)+CHR\$(low)+CHR\$(high)

Die Besonderheit dieses Befehls beginnt eigentlich schon nach der Angabe der Dateinummer. Das 'P' bedeutet hier POSITION, und sagt dem Betriebssystem, daß die nachfolgende CHR\$-Folge den

Datensatz-Zeiger auf den Datensatz setzen soll, der durch LOW und HIGH angegeben wird (wie LOW und HIGH berechnet werden, zeigen wir Ihnen später noch).

Der Befehl kann sogar noch erweitert werden. Wenn Sie an den Schluß der CHR\$-Folge noch ein weiteren CHR\$-Befehl setzen, so gibt dieser an, an welche Stelle innerhalb des Datensatzes positioniert werden soll. So können Sie also gezielt nach einem ganz bestimmten Zeichen oder Wort suchen.

Ganz wichtig ist noch eine Besonderheit, auf die bei der Programmierung sowohl der sequentiellen, als auch der realativen Datei geachtet werden sollte:

Nach jedem FELD, das in den Datensatz geschrieben wurde, muß ein Abschlußzeichen (CHR\$(13)) gesetzt werden. Ansonsten wird der Computer nicht zwischen den unterschiedlichen Feldern unterscheiden können. Aus diesem Grund haben wir auch immer die PRINT#-Befehle in zwei Zeilen gesetzt, da so automatisch ein Carriage Return, also CHR\$(13), mit abgespeichert wird.

An einem konkreten Beispiel sieht man natürlich leichter, was wir hier zu erklären versuchen. Deshalb haben wir anschließend an dieses Kapitel ein voll funktionsfähiges Lagerverwaltungsprogramm gefügt. So können Sie leichter die ganzen Funktionen dieses Kapitels in einem Programm angewendet sehen. Wir glauben, daß sich die Mühe dieses Programm einzugeben lohnt, wenn man bedenkt, daß sich schon aus kleinen Änderungen an diesem Programm hieraus eine Adressverwaltung, Video-Kassetten-Verwaltung und dergleichen machen läßt.

Nun aber noch etwas zu den oben schon angesprochenen HIGH und LOW Adressen. Diese HIGH und LOW Adressen ergeben zusammen die eigentliche Datensatznummer. Die Formel hierfür lautet:

$$\text{Datensatznummer} = \text{HIGH} * 256 + \text{LOW}$$

Somit können wir auch Datensätze lesen, deren Adresse größer als 255 ist. Soll also z.B. der 78te Satz gelesen werden, so muß zunächst LOW und HIGH berechnet werden:

$$\text{HIGH} = \text{INT} (\text{Datensatznummer} / 256)$$

$$\text{LOW} = \text{Datensatznummer} - \text{HIGH} * 256$$

oder an einem konkreten Beispiel:

$$\text{HIGH} = \text{INT}(78/256); \text{LB} = 78 - \text{HB} * 256$$

und als Ergebnis für HIGH und LOW:

$$\text{HIGH} = 0, \text{LOW} = 78$$

Diese Berechnung ist an für sich recht witzlos, da wir ja so wie so einen Datensatz lesen wollen, der kleiner als 256 ist. Aber trotzdem zeigt dieses Beispiel, wie eine solche Berechnung aussehen muß.

Dieses Ergebnis muß nun noch in die Formel zum Positionieren der Zeiger eingesetzt werden. Diese Formel lautet danach so:

```
PRINT # Kanalnummer, "P";CHR$(Kanalnummer)+CHR$(0)+CHR$(78)
```

In unserem Lagerverwaltungsprogramm finden Sie zum Beispiel folgende Struktur:

```
PRINT#15, "P"+CHR$(3)+CHR$(231)+CHR$(3)+CHR$(1)
```

Das bedeutet nun, daß der Zeiger auf den 999 Datensatz gesetzt werden soll, und zwar auf das erste Zeichen.

Bevor man allerdings die Datei benutzen kann, muß Sie erst für die relative Anwendung vorbereitet werden. Dies geschieht, indem

zuerst der Zeiger im Kommandokanal (15) geöffnet wird (s.o.), und dann dieser Datensatz mit dem Zeichen 255 (CHR\$(255)) beschrieben wird. Dieses Zeichen sagt dem Betriebssystem, daß an dieser Stelle ein Datensatz steht, der zwar schon existiert, in dem aber noch nichts gespeichert ist. In unserem Beispiel werden also alle 999 Datensätze mit dieser 255 gekennzeichnet.

Nun können die Datensätze in diese Datei geschrieben werden - natürlich nicht mehr, als wie bei der Eröffnung der Datei angegeben wurden. Versucht man dagegen einen Datensatz zu schreiben, der außerhalb des erlaubten Bereichs liegt, so meldet sich der Rechner mit der Fehlermeldung RECORD NOT PRESENT - dieser Datensatz existiert also nicht.

Nun aber noch etwas zu unserem Programm. Nachdem Sie es durch RUN gestartet haben, erscheint die Frage, ob das Diskettengerät bereit ist. Diese Meldung wird solange erscheinen, bis Sie die 'J'-Taste drücken. Danach werden Sie gefragt, ob Sie eine neue Diskette verwenden wollen. Neu heißt in dem Fall nicht nur, daß die Diskette unformatiert sein kann, sondern auch, daß noch keine Datei vorhanden ist. Aber Vorsicht, die Diskette wird in jedem Fall formatiert. Benutzen Sie also nicht Ihr Programm- oder Ihre Datendiskette hierfür. Wenn der Vorgang beendet ist, erscheint das Hauptmenü. Hier können Sie nun eine der 6 möglichen Funktionen aufrufen.

Wenn Sie einen Datensatz gestalten wollen, so denken Sie daran, daß die Eingabe nicht länger sein darf, als in den DATA-Zeilen (30-82) festgelegt ist. Diese Zeilen sind so aufgebaut, daß zuerst der Name und dann die Länge der Bezeichnung aufgeführt wird. Wollen Sie einen bereits eingegebenen Datensatz wieder löschen, so gehen Sie in die Routine AENDERN, und schreiben als erstes Zeichen der Bezeichnung WARENART einen Klammeraffen (@). Somit wird dieser Satz als gelöscht gekennzeichnet.

Wenn Sie in irgendeinem Unterprogramm aufgefordert werden, eine Artikelnummer einzugeben, so können Sie durch die Eingabe von

"ENDE" wieder ins Haupt- oder Übermenü zurückgelangen.

Die Zeilen 11 bis 14 stellen den heutigen Mehrwertsteuersatz dar. ME(1) und ME(2) sind die beiden alten Sätze von 6,5% und 13%. Die nächsten beiden sind die neuen Sätze von 7% und 14%. Werden Sie also nach dem Mehrwertsteuersatz gefragt, so geben Sie einfach eine '1' für 6,5%, eine '2' für 13%, eine '3' für 7% und eine '4' für 14% ein. Der Computer berechnet sich daraus dann den jeweiligen Mehrwertsteuersatz.

Eine Besonderheit ist bei diesem Programm aber noch zu beachten. Artikelnummern, die von 800 bis 999 gehen (999 ist die größte mögliche Nummer), werden nicht als Artikel sondern als verschiedene Ausgaben oder Einnahmen gewertet. So können Sie gleichzeitig zu den verschiedenen Artikeln auch noch Ihre Privatkonten, Bankkonten usw. überwachen. Die Eingabe dieser Konten erfolgt im Unterprogramm EINGABE DER KASSENZETTEL. Wenn die erste Eingabe, also der erste "Kassenzettel", genauer gesagt die Rechnung, eine Zahl größer als 799 war, so erwartet der Computer weitere Eingaben dieser Art. Das heißt, daß Sie nicht abwechselnd Artikel und Rechnungen eingeben können, sondern nur jeweils getrennt.

Außerdem ist darauf zu achten, daß bei der Arbeit mit diesem Programm Drucker und Floppy immer angeschaltet sind. Haben Sie aber keinen Drucker, so müssen Sie das Programm umschreiben. Sie sehen ja an den Print#4-Befehlen, in welchen Zeilen eine Ausgabe auf dem Drucker erfolgen soll.

Wenn Sie noch weitere Filialen oder Personen mit diesem Programm unterstützen, so empfiehlt es sich, für jede Filiale oder Person eine neue Diskette anzulegen. Dann wissen Sie zu jeder Zeit über Ihre finanzielle Lage Bescheid.

Wir hoffen, daß Sie mit Hilfe dieses Programmes einen Einblick in die Datenverarbeitung, und besonders in die Datenverarbeitung mit relativen Dateien bekommen haben. Es sieht auf den ersten Blick schwieriger aus als es ist. Mit ein wenig Übung werden auch Sie rasch solche Programme selbst entwerfen können.


```

10 CLR
15 ME(1)=1.065: ME(2)=1.13: ME(3)=1.07 :ME(4)=1.14
20 FOR I=1 TO 7: READ TD$(I),TD(I): NEXT I
30 DATA "1) ARTIKEL NR.      :",3
32 DATA "2) WARENART        :",20
34 DATA "3) ANZAHL          :",3
36 DATA "4) EINZELPREIS EK:",7
38 DATA "5) GESAMTPREIS EK:",8
40 DATA "6) EINZELPREIS VK:",7
42 DATA "7) GESAMTPREIS VK:",8
50 FOR I=1 TO 3: READ TI$(I),TI(I): NEXT I
56 DATA "1) ARTIKEL NR.      :",3
58 DATA "2) WARENART        :",20
60 DATA "3) MWST GRUPPE     :",1
70 FOR I=1 TO 4: READ TT$(I),TT(I): NEXT I
76 DATA "1) FILIALE NR.      :",1
78 DATA "2) DATUM           :",8
80 DATA "3) RECHNUNGS NR.    :",8
82 DATA "4) BELEG NR.        :",8
100 PRINT CHR$(147)
110 PRINT "*****";
120 PRINT "      DATEIVERWALTUNGS PROGRAMM 1.0      ";
130 PRINT "*****";
140 PRINT: PRINT
150 PRINT "DISKETTEN GERAET ANGESCHLOSSEN (J/N)? ";
160 GET A$: IF A$="" THEN 160
170 IF A$<>"J" THEN 160
180 PRINT A$
190 OPEN 15,8,15,"IO": CLOSE 15
200 PRINT "NEUE DATENDISKETTE ANLEGEN (J/N)? ";
210 GET A$: IF A$="" THEN 210
220 IF A$<>"J" THEN 300
222 PRINT A$
230 OPEN 15,8,15,"N:DATENDISKETTE,AH"
240 OPEN 1,8,3,"O:ARTDAT,L,"+CHR$(64)
250 PRINT#15,"P"+CHR$(3)+CHR$(231)+CHR$(3)+CHR$(1)
260 PRINT#1,CHR$(255);

```

```

270 RM=INT(167132/64)
280 CLOSE 1: CLOSE 15
300 PRINT CHR$(147)
310 PRINT "*****";
320 PRINT " *      DATEIVERWALTUNGS PROGRAMM 1.0      *";
330 PRINT "*****";
340 PRINT: PRINT
345 PRINT TAB(15);"HAUPTMENU": PRINT: PRINT
350 PRINT "      1) EINGEBEN DER ARTIKELDATEI": PRINT
355 PRINT "      2) AENDERN DER ARTIKELDATEI": PRINT
360 PRINT "      3) EINGEBEN DER KASSENZETTEL": PRINT
365 PRINT "      4) DRUCKEN DER ARTIKELLISTE": PRINT
370 PRINT "      5) DRUCKEN DER AUSWERTUNG": PRINT
375 PRINT "      6) ENDE DES PROGRAMMES": PRINT: PRINT
380 PRINT "BITTE WAEHLEN SIE (1-6) : ";
390 GET A$: IF A$="" THEN 390
400 A=VAL(A$): IF A<1 OR A>6 THEN 390
410 PRINT A$
420 FOR I=1 TO 1000: NEXT
430 ON A GOTO 1000, 2000, 3000, 4000, 5000, 6000
1000 OPEN 15,8,15: OPEN 8,8,8,"O:ARTDAT"
1002 GOSUB 12000
1005 PRINT CHR$(147)
1010 PRINT "*****";
1020 PRINT " *      DATEIVERWALTUNGS PROGRAMM 1.0      *";
1030 PRINT "*****";
1040 PRINT: PRINT
1050 PRINT TAB(7);"EINGEBEN DER ARTIKELDATEI": PRINT: PRINT
1060 FOR I=1 TO 3
1065 TE$(I)=""
1070 PRINT TI$(I);
1080 INPUT TE$(I)
1090 PRINT
1092 IF TE$(1)="ENDE" THEN 1200
1095 IF LEN(TE$(I))>TI(I) THEN 1065
1100 NEXT
1102 FOR I=4 TO 8: TE$(I)="": NEXT

```

```

1110 RN=VAL(TE$(1))
1120 IF RN<1 OR RN>999 THEN 1005
1130 GOSUB 10000
1140 GOSUB 10070
1150 GOTO 1005
1200 CLOSE 8: CLOSE 15
1220 GOTO 300
2000 OPEN 15,8,15: OPEN 8,8,8,"O:ARTDAT"
2002 GOSUB 12000
2005 PRINT CHR$(147)
2010 PRINT "*****";
2020 PRINT " *      DATEIVERWALTUNGS PROGRAMM 1.0      *";
2030 PRINT "*****";
2040 PRINT: PRINT
2050 PRINT TAB(8);"AENDERN DER ARTIKELDATEI": PRINT: PRINT
2055 TE$(1)=" "
2060 PRINT TI$(1);
2070 INPUT TE$(1)
2080 PRINT
2090 IF TE$(1)="ENDE" THEN 2400
2100 IF LEN(TE$(1))>TI(1) THEN 2055
2110 RN=VAL(TE$(1))
2120 IF RN<1 OR RN>999 THEN 2005
2130 GOSUB 10000
2140 GOSUB 10030
2142 IF VAL(TE$(1))<>RN THEN 2005
2150 PRINT CHR$(147)
2160 PRINT "*****";
2170 PRINT " *      DATEIVERWALTUNGS PROGRAMM 1.0      *";
2180 PRINT "*****";
2190 PRINT: PRINT
2200 PRINT TAB(8);"AENDERN DER ARTIKELDATEI": PRINT: PRINT
2210 FOR I=1 TO 3
2220 PRINT TI$(I);"? ";
2230 PRINT TE$(I)
2240 PRINT CHR$(145);
2250 PRINT TI$(I);

```

```

2260 INPUT TE$(I)
2270 PRINT
2280 IF TE$(1)="ENDE" THEN 2400
2290 IF LEN(TE$(I))>TI(I) THEN 2250
2300 NEXT
2310 RN=VAL(TE$(1))
2320 IF RN<1 OR RN>999 THEN 2005
2330 GOSUB 10000
2340 GOSUB 10070
2400 CLOSE 4: CLOSE 8: CLOSE 15
2430 GOTO 300
2530 GOTO 3005
3000 OPEN 15,8,15: OPEN 8,8,8,"O:ARTDAT"
3002 GOSUB 12000
3005 PRINT CHR$(147)
3010 PRINT "*****";
3020 PRINT "*      DATEIVERWALTUNGS PROGRAMM 1.0      *";
3030 PRINT "*****";
3040 PRINT: PRINT
3050 PRINT TAB(7);"EINGEBEN DER KASSENZETTEL": PRINT: PRINT
3060 TE$(1)=" "
3070 PRINT TE$(1);
3080 INPUT TE$(1)
3090 PRINT
3100 IF TE$(1)="ENDE" THEN 3700
3110 IF LEN(TE$(1))>TI(1) THEN 3060
3120 RN=VAL(TE$(1))
3130 IF RN<1 OR RN>999 THEN 3005
3132 IF DW=1 AND RN>799 THEN 3005
3134 IF DW=2 AND RN<800 THEN 3000
3140 GOSUB 10000
3150 GOSUB 10030
3152 IF VAL(TE$(1))<>RN THEN 3005
3154 IF LEFT$(TE$(2),1)="@" THEN 3005
3160 PRINT CHR$(147)
3170 PRINT "*****";
3180 PRINT "*      DATEIVERWALTUNGS PROGRAMM 1.0      *";

```

```

3290 PRINT "*****";
3200 PRINT: PRINT
3210 PRINT TAB(7);"EINGEBEN DER KASSENZETTEL": PRINT: PRINT
3212 FOR I=1 TO 5
3214 TH$(I)=TE$(I+3)
3216 NEXT
3220 FOR I=1 TO 2
3230 PRINT TD$(I);"? ";TE$(I)
3235 TX$(I)=TE$(I)
3240 PRINT
3250 NEXT
3255 TX$(3)=TE$(3)
3260 PRINT TD$(3);
3270 INPUT TX$(4)
3275 TE$(4)=TX$(4)
3280 PRINT
3285 IF VAL(TE$(4))<-999 OR VAL(TE$(4))>999 THEN 3260
3287 IF LEN(TE$(4))>TD(3) THEN 3260
3290 PRINT TD$(4);
3295 TE$(5)=" "
3300 INPUT TX$(5)
3305 TE$(5)=TX$(5)
3310 PRINT
3315 IF LEN(TE$(5))>TD(4) THEN 3290
3320 PRINT TD$(6);
3325 TE$(7)=" "
3330 INPUT TX$(7)
3335 TE$(7)=TX$(7)
3340 PRINT
3345 IF LEN(TE$(7))>TD(6) THEN 3320
3350 TH=VAL(TE$(5))*VAL(TE$(4))
3351 TX$(6)=STR$(TH)
3352 TH=TH+VAL(TH$(3))
3355 TE$(6)=STR$(TH)
3360 TH=VAL(TE$(7))*VAL(TE$(4))
3361 TX$(8)=STR$(TH)
3362 TH=TH+VAL(TH$(5))

```

```

3365 TE$(8)=STR$(TH)
3370 TH=VAL(TE$(5))
3371 IF VAL(TE$(4))<1 THEN TH=-TH
3372 TH=TH+VAL(TH$(2))
3375 TE$(5)=STR$(TH)
3380 TH=VAL(TE$(7))
3381 IF VAL(TE$(4))<1 THEN TH=-TH
3382 TH=TH+VAL(TH$(4))
3385 TE$(7)=STR$(TH)
3390 TH=VAL(TE$(4))
3392 TH=TH+VAL(TH$(1))
3395 TE$(4)=STR$(TH)
3460 RN=VAL(TE$(1))
3470 GOSUB 10000
3480 GOSUB 10070
3485 FOR I=1 TO 8: TE$(I)=TX$(I): TX$(I)="": NEXT
3490 IF DW=0 AND VAL(TE$(1))<800 THEN DW=1: GOSUB 5360: GOTO 3510
3500 IF DW=0 AND VAL(TE$(1))>799 THEN DW=2: GOSUB 7005: GOTO 3520
3510 IF DW=1 THEN GOSUB 5520: GOTO 3530
3520 IF DW=2 THEN GOSUB 7120
3530 GOTO 3005
3700 IF DW=1 THEN GOSUB 5590: GOTO 3800
3710 IF DW=2 THEN GOSUB 7190
3800 DW=0: DV=0
3999 CLOSE 4: CLOSE 8: CLOSE 15: GOTO 300
4000 OPEN 15,8,15: OPEN 8,8,8,"O:ARTDAT"
4002 GOSUB 12000
4005 PRINT CHR$(147)
4010 PRINT "*****";
4020 PRINT " *      DATEIVERWALTUNGS PROGRAMM 1.0      *";
4030 PRINT "*****";
4040 PRINT: PRINT
4050 PRINT TAB(8);"DRUCKEN DER ARTIKELLISTE": PRINT: PRINT
4060 FOR I=1 TO 2
4070 TE$(I)=" "
4080 PRINT TT$(I);
4090 INPUT TE$(I)

```

```

4100 PRINT
4110 IF TE$(1)="ENDE" THEN 4999
4120 IF LEN(TE$(1))>TT(1) THEN 4070
4130 NEXT
4135 TE$(3)="": TE$(4)=" "
4140 IF DV=1 THEN RETURN
4200 PRINT CHR$(147)
4210 PRINT "*****";
4220 PRINT " *      DATEIVERWALTUNGS PROGRAMM 1.0      * ";
4230 PRINT "*****";
4240 PRINT: PRINT
4250 PRINT TAB(8);"DRUCKEN DER ARTIKELLISTE": PRINT: PRINT
4260 PRINT "IST DER DRUCKER EINGESCHALTET (J/N)? ";
4270 GET A$: IF A$="" THEN 4270
4280 IF A$<>"J" THEN 4270
4290 PRINT A$
4300 OPEN 4,4
4310 PRINT#4,"FILIALE NR. ";TE$(1);
4330 PRINT#4,CHR$(16);"20";"DATUM ";TE$(2);
4340 PRINT#4,CHR$(16);"40";"RECHNUNGS NR. ";TE$(3);
4345 PRINT#4,CHR$(16);"60";"BELEG NR. ";TE$(4)
4350 PRINT#4
4360 PRINT#4,"ARTIKEL NR. ";
4375 PRINT#4,CHR$(16);"20";"WARENART";
4385 PRINT#4,CHR$(16);"60";"MWST GRUPPE"
4390 PRINT#4,"-----";
4405 PRINT#4,CHR$(16);"20";"-----";
4415 PRINT#4,CHR$(16);"60";"-----"
4420 FOR RN=1 TO 999
4430 GOSUB 10000
4440 GOSUB 10030
4442 IF VAL(TE$(1))<>RN THEN 4480
4444 IF LEFT$(TE$(2),1)="@ " THEN 4480
4450 PRINT#4,TE$(1);
4460 PRINT#4,CHR$(16);"20";TE$(2);
4470 PRINT#4,CHR$(16);"60";TE$(3)
4480 NEXT

```

```

4490 FOR I=1 TO 3
4492 PRINT#4
4494 NEXT
4999 CLOSE 4: CLOSE 8: CLOSE 15: GOTO 300
5000 OPEN 15,8,15: OPEN 8,8,8,"O:ARTDAT"
5002 GOSUB 12000
5005 PRINT CHR$(147)
5010 PRINT "*****";
5020 PRINT "      DATEIVERWALTUNGS PROGRAMM 1.0      ";
5030 PRINT "*****";
5040 PRINT: PRINT
5050 PRINT TAB(9);"DRUCKEN DER AUSWERTUNG": PRINT: PRINT
5060 FOR I=1 TO 4
5070 TE$(I)=" "
5080 PRINT TT$(I);
5090 INPUT TE$(I)
5100 PRINT
5110 IF TE$(1)="ENDE" THEN 5999
5120 IF LEN(TE$(I))>TT(I) THEN 5070
5125 IF DV=0 AND I=2 THEN I=4
5130 NEXT
5140 IF DV=1 THEN RETURN
5200 PRINT CHR$(147)
5210 PRINT "*****";
5220 PRINT "      DATEIVERWALTUNGS PROGRAMM 1.0      ";
5230 PRINT "*****";
5240 PRINT: PRINT
5250 PRINT TAB(9);"DRUCKEN DER AUSWERTUNG": PRINT: PRINT
5252 SA=0: SE=0: SG=0: SF=0: SH=0
5255 FOR I=1 TO 4: M1(I)=0: M2(I)=0: NEXT
5260 PRINT "IST DER DRUCKER EINGESCHALTET (J/N)? ";
5270 GET A$: IF A$="" THEN 5270
5280 IF A$<>"J" THEN 5270
5290 PRINT A$
5300 OPEN 4,4
5310 PRINT#4,"FILIALE NR. ";TE$(1);
5330 PRINT#4,CHR$(16);"20";"DATUM ";TE$(2);

```



```

5335 IF DV=0 THEN PRINT#4: GOTO 5350
5340 PRINT#4,CHR$(16);"40";"RECHNUNGS NR. ";TE$(3);
5345 PRINT#4,CHR$(16);"60";"BELEG NR. ";TE$(4)
5350 PRINT#4
5355 IF DV=1 THEN RETURN
5360 PRINT#4,"ART. NR. ";
5370 PRINT#4,CHR$(16);"10";"ANZ. ";
5375 PRINT#4,CHR$(16);"15";"WARENART";
5380 PRINT#4,CHR$(16);"40";"EINZ. EK";
5390 PRINT#4,CHR$(16);"50";"GESA. EK";
5400 PRINT#4,CHR$(16);"60";"EINZ. VK";
5410 PRINT#4,CHR$(16);"70";"GESA. VK"
5420 PRINT#4,"-----";
5425 PRINT#4,CHR$(16);"10";"----";
5430 PRINT#4,CHR$(16);"15";"-----";
5440 PRINT#4,CHR$(16);"40";"-----";
5450 PRINT#4,CHR$(16);"50";"-----";
5460 PRINT#4,CHR$(16);"60";"-----";
5470 PRINT#4,CHR$(16);"70";"-----"
5475 IF DV=1 THEN RETURN
5480 FOR RN=1 TO 799
5490 GOSUB 10000
5500 GOSUB 10030
5510 IF VAL(TE$(1))<>RN THEN 5580
5515 IF LEFT$(TE$(2),1)="@" THEN 5580
5520 PRINT#4,TE$(1);
5525 PRINT#4,CHR$(16);"10";TE$(4);
5530 PRINT#4,CHR$(16);"15";TE$(2);
5540 PRINT#4,CHR$(16);"40";TE$(5);
5550 PRINT#4,CHR$(16);"50";TE$(6);
5560 PRINT#4,CHR$(16);"60";TE$(7);
5570 PRINT#4,CHR$(16);"70";TE$(8)
5571 SA=SA+VAL(TE$(4))
5572 SE=SE+VAL(TE$(5))
5573 SG=SG+VAL(TE$(6))
5574 SF=SF+VAL(TE$(7))
5575 SH=SH+VAL(TE$(8))

```

```

5576 IF VAL(TE$(3))<1 OR VAL(TE$(3))>4 THEN 5579
5577 M1(VAL(TE$(3)))=M1(VAL(TE$(3)))+(VAL(TE$(6))*ME(VAL(TE$(3)))
      -VAL(TE$(6)))
5578 M2(VAL(TE$(3)))=M2(VAL(TE$(3)))+(VAL(TE$(8))-VAL(TE$(8))/ME(
      VAL(TE$(3))))
5579 IF DV=1 THEN RETURN
5580 NEXT
5590 PRINT#4,"-----";
5600 PRINT#4,CHR$(16);"10";"----";
5605 PRINT#4,CHR$(16);"15";"-----";
5610 PRINT#4,CHR$(16);"40";"-----";
5620 PRINT#4,CHR$(16);"50";"-----";
5630 PRINT#4,CHR$(16);"60";"-----";
5640 PRINT#4,CHR$(16);"70";"-----"
5650 PRINT#4,"SUMMEN:";
5660 PRINT#4,CHR$(16);"09";SA;
5670 PRINT#4,CHR$(16);"39";SE;
5680 PRINT#4,CHR$(16);"49";SG;
5690 PRINT#4,CHR$(16);"59";SF;
5700 PRINT#4,CHR$(16);"69";SH
5710 FOR I=1 TO 3: PRINT#4: NEXT
5720 PRINT#4,"MWST. EK  6,5 % = ";M1(1);
5725 PRINT#4,CHR$(16);"40";"MWST. EK  7 % = ";M1(3)
5730 PRINT#4,"MWST. EK 13,0 % = ";M1(2);
5735 PRINT#4,CHR$(16);"40";"MWST. EK 14 % = ";M1(4)
5740 PRINT#4,"MWST. VK  6,5 % = ";M2(1);
5745 PRINT#4,CHR$(16);"40";"MWST. VK  7 % = ";M2(3)
5750 PRINT#4,"MWST. VK 13,0 % = ";M2(2);
5755 PRINT#4,CHR$(16);"40";"MWST. VK 14 % = ";M2(4)
5756 FOR I=1 TO 3: PRINT#4: NEXT
5760 PRINT#4,"GESAMT EK. = ";SG+M1(1)+M1(2)+M1(3)+M1(4)
5770 PRINT#4,"GESAMT VK. = ";SH-M2(1)-M2(2)-M2(3)-M2(4)
5775 FOR I=1 TO 3: PRINT#4: NEXT
5777 IF DV=1 THEN RETURN
5780 GOSUB 7000
5999 CLOSE 4: CLOSE 8: CLOSE 15: GOTO 300
6000 CLOSE 4: CLOSE 8: CLOSE 15

```

```

6030 PRINT CHR$(147)
6040 END
7000 S1=0: S2=0
7005 PRINT#4,"ART. NR.";
7010 PRINT#4,CHR$(16);"15";"ART";
7020 PRINT#4,CHR$(16);"40";"AUSGABEN";
7030 PRINT#4,CHR$(16);"60";"EINNAHMEN";
7040 PRINT#4,"-----";
7050 PRINT#4,CHR$(16);"15";"---";
7060 PRINT#4,CHR$(16);"40";"-----";
7070 PRINT#4,CHR$(16);"60";"-----";
7075 IF DV=1 THEN RETURN
7080 FOR RN=800 TO 999
7090 GOSUB 10000
7100 GOSUB 10030
7110 IF VAL(TE$(1))<>RN THEN 7180
7120 PRINT#4,TE$(1);
7130 PRINT#4,CHR$(16);"15";TE$(2);
7140 PRINT#4,CHR$(16);"40";TE$(5);
7150 PRINT#4,CHR$(16);"60";TE$(7)
7160 S1=S1+VAL(TE$(5))
7170 S2=S2+VAL(TE$(7))
7171 IF VAL(TE$(3))<1 OR VAL(TE$(3))>4 THEN 7175
7172 M2(VAL(TE$(3)))=M2(VAL(TE$(3)))+(VAL(TE$(7))-VAL(TE$(7))/ME(
    VAL(TE$(3))))
7173 M1(VAL(TE$(3)))=M1(VAL(TE$(3)))+(VAL(TE$(5))-VAL(TE$(5))/ME(
    VAL(TE$(3))))
7175 IF DV=1 THEN RETURN
7180 NEXT
7190 PRINT#4,"-----";
7200 PRINT#4,CHR$(16);"15";"---";
7210 PRINT#4,CHR$(16);"40";"-----";
7220 PRINT#4,CHR$(16);"60";"-----"
7230 PRINT#4,"SUMMEN:";
7240 PRINT#4,CHR$(16);"39";S1;
7250 PRINT#4,CHR$(16);"59";S2
7260 FOR I=1 TO 3: PRINT#4: NEXT

```

```

7270 PRINT#4,"MWST. EK 6,5 % = ";M1(1);
7280 PRINT#4,CHR$(16);"40";"MWST. EK 7 % = ";M1(3)
7290 PRINT#4,"MWST. EK 13,0 % = ";M1(2);
7300 PRINT#4,CHR$(16);"40";"MWST. EK 14 % = ";M1(4)
7310 PRINT#4,"MWST. VK 6,5 % = ";M2(1);
7320 PRINT#4,CHR$(16);"40";"MWST. VK 7 % = ";M2(3)
7330 PRINT#4,"MWST. VK 13,0 % = ";M2(2);
7340 PRINT#4,CHR$(16);"40";"MWST. VK 14 % = ";M2(4)
7350 FOR I=1 TO 3: PRINT#4: NEXT
7360 PRINT#4,"AUSGABEN = ";S1-M1(1)-M1(2)-M1(3)-M1(4)
7370 PRINT#4,"EINNAHMEN " ";S2-M2(1)-M2(2)-M2(3)-M2(4)
7380 FOR I=1 TO 3: PRINT#4: NEXT
7999 RETURN
8000 PRINT CHR$(147)
8010 PRINT "*****";
8020 PRINT "*      DATEIVERWALTUNGS PROGRAMM 1.0      *";
8030 PRINT "*****";
8040 PRINT: PRINT
8050 PRINT TAB(7);"EINGEBEN DER KASSENZETTEL": PRINT: PRINT
8060 SA=0: SE=0: SG=0: SF=0: SH=0: S1=0: S2=0
8070 FOR I=1 TO 4
8072 M1(I)=0
8074 M2(I)=0
8076 NEXT
8080 PRINT "IST DER DRUCKER EINGESCHALTET (J/N)? ";
8090 GET A$: IF A$="" THEN 8090
8100 IF A$<>"J" THEN 8090
8110 PRINT A$
8120 OPEN 4,4
8130 RETURN
8500 PRINT CHR$(147)
8510 PRINT "*****";
8520 PRINT "*      DATEIVERWALTUNGS PROGRAMM 1.0      *";
8530 PRINT "*****";
8540 PRINT: PRINT
8550 PRINT TAB(7);"EINGEBEN DER KASSENZETTEL": PRINT: PRINT
8560 GOSUB 5060

```

```

8570 GOSUB 5310
8580 RETURN
10000 HB=INT(RN/256): LB=RN-HB*256
10010 PRINT#15,"P"+CHR$(8)+CHR$(LB)+CHR$(HB)+CHR$(1)
10015 GOSUB 12000
10020 RETURN
10030 INPUT#8,TE$(1),TE$(2),TE$(3),TE$(4),TE$(5),TE$(6),TE$(7),TE
$(8)
10060 RETURN
10070 TE$=TE$(1)+CHR$(13)+TE$(2)+CHR$(13)+TE$(3)+CHR$(13)+TE$(4)+
CHR$(13)
10072 TE$=TE$+TE$(5)+CHR$(13)+TE$(6)+CHR$(13)+TE$(7)+CHR$(13)+TE$
(8)
10080 PRINT#8,TE$
10110 RETURN
12000 INPUT#15,X,X$,Y$,Z$
12010 IF X=0 OR X=50 THEN RETURN
12020 PRINT "FEHLER: ";X,X$;Y$;Z$
12030 CLOSE 8: CLOSE 15
12040 FOR I=1 TO 6000: NEXT
12060 GOTO 100

```

Viel Spaß mit diesem Programm

8.5 EINE ANDERE METHODE: DIREKTZUGRIFF

Diese Art des Zugriffs auf Daten und Diskette wird leider oft völlig vernachlässigt. Zwar ist sie auch sehr kompliziert und aufwendig, hat aber auch einige sehr interessante Aspekte. Was läßt sich nun mit dem Direktzugriff alles verwirklichen ?

1.) Der Zugriff auf Dateien - Random-Dateien

Diese Methode hat einiges mit der sequentiellen Dateiverwaltung zu tun, ohne aber deren Nachteile zu haben, und einiges mit der relativen Dateiverwaltung.

2.) Der Zugriff auf einzelne Spuren der Diskette

Mit diesem Zugriff stehen Ihnen Möglichkeiten zur Verfügung, an die Sie bisher noch gar nicht dachten, und deren Zweck Sie vielleicht noch gar nicht einsehen. Mehr dazu noch später.

1. Random-Dateien

Im Unterschied zu den sequentiellen und relativen Dateien, ist hier der einzelne Block immer 256 Bytes lang - auf eine Diskette gehen insgesamt 664 solcher Blöcke. Trotzdem können Sie auch kleinere Datensätze abspeichern. Es ist beispielsweise denkbar, daß Sie in einen Block insgesamt 4 Datensätze a' 64 Bytes Länge speichern. Sie müssen dann nur darauf achten, daß Sie innerhalb eines solchen Blocks auch genau auf den Satz zugreifen, den Sie tatsächlich verarbeiten wollen. Sie müssen zunächst eine sequentielle Datei öffnen, um dort die Spuren zu vermerken, in denen Sie die Datensätze gespeichert haben. Insgesamt benötigen Sie also 3 Dateien:

- 1.) Sequentielle Datei für die Zeiger**
- 2.) Kommando-Datei**
- 3.) Daten-Datei zur direkten Speicherung**

```

10 OPEN 4,8,4,"CBM 64 DATEI,S,W": REM SEQUENT. DATEI
20 OPEN 15,8,15: REM KOMMANDOKANAL
30 OPEN 5,8,5,"#": REM DATEN-DATEI
40 TE$="DATA BECKER GMBH"
50 PRINT#5,TE$;",";1: REM TEXT, RECORD#
60 T=1: S=1: REM TRACK=1, SECTOR=1
70 PRINT#15,"B-A: ";O,T,S: REM DRIVE, TRACK, SECTOR
80 INPUT#15,ER,NA$,TR,BL: REM LESEN FEHLER
90 IF ER=65 THEN T=TR: S=BL: GOTO 70
100 PRINT#15,"B-W: ";5,O,T,S: REM SCHREIBE RECORD
110 PRINT#4,T;",";S
120 CLOSE 5
130 CLOSE 15
140 CLOSE 4
150 END

```

Was macht nun dieses Programm ? Zunächst werden die 3 notwendigen Dateien geöffnet, dann ein Text definiert, der direkt abgespeichert werden soll. Dieser Text wird in der Daten-Datei zunächst zwischengespeichert. Nun geht das Betriebssystem auf die Suche nach einem freien Block auf der Diskette. Die Suche beginnt bei Track 1, Sektor 1, also den Start der Diskette. Dazu wird die Zeile 70 benötigt. "B-A:" heißt BLOCK-ALLOCATE oder übersetzt soviel wie BLOCK-SUCHE. Es wird also ein Block auf der Diskette gesucht, der noch nicht beschrieben wurde, bzw. wieder freigegeben wurde. Um zu sehen, ob der gesuchte Block frei ist, muß der Kommandokanal gelesen werden. Hat der Fehlercode den Wert 65, so bedeutet das, daß dieser Block nicht frei ist und der Computer fährt mit seiner Suche fort. Hat er jedoch einen freien Block gefunden, so schreibt er den Text, der in der Daten-Datei bereit gestellt wurde, mit den gefundenen Blockadressen, auf die Diskette. Danach werden die Adressen noch in der sequentiellen Datei abgespeichert, damit der Satz später wiedergefunden wird. Gleichzeitig sperrt das Betriebssystem diese Spur automatisch, damit sie nicht unbeabsichtigt überschrieben wird. Die Dateien werden geschlossen und das Programm ist beendet.

Genauso interessant ist das Wiedereinlesen der Daten:

```
10 OPEN 4,8,4,"CBM 64 DATEI"  
20 OPEN 15,8,15  
30 OPEN 5,8,5,"#"  
40 INPUT#4,T,S: REM LESEN DER ADRESSEN  
50 PRINT#15,"B-R: ";5,0,T,S  
60 INPUT#5,TE$,RE  
70 PRINT#15,"B-F: ";0,T,S  
80 CLOSE 5  
90 CLOSE 4  
100 PRINT#15,"S:CBM 64 DATEI"  
110 CLOSE 15
```

Und auch zu diesem Programm eine kurze Erläuterung: Nachdem die Dateien geöffnet wurden, wird die Adresse des Blocks eingelsen, in dem die eigentlichen Daten, in unserem Fall der Text, gespeichert sind. Nachdem dies geschehen ist, wird dieser Block durch den BLOCK-FREE Befehl wieder freigegeben. Dies darf natürlich nur dann geschehen, wenn der Block gelöscht werden kann. Danach werden die sequentielle- und die Daten-Datei wieder geschlossen, die sequentielle Datei wieder gelöscht und auch der Kommandokanal wieder geschlossen.

2. Der direkte Diskettenzugriff

Dieser Zugriff ermöglicht, wie sein Name schon sagt, den direkten Zugriff auf beliebige TRACKS und SEKTOREN, also das Lesen von Informationen direkt von Disk, ohne erst Dateien zu eröffnen. So können Sie zum Beispiel leicht das Inhaltsverzeichnis lesen, ohne den LOAD "\$",8 Befehl anwenden zu müssen und damit ein Programm, das im Speicher steht vielleicht zu zerstören. Oder Sie können ein Programm auf Diskette ändern, ohne es laden zu müssen - selbst zerstörte Programme können unter Umständen wieder repariert werden.

Diese Zugriffe sind aber so gefährlich, daß wir jeden Programmierer davor warnen möchten, sie ohne genaues Wissen über den Aufbau der Diskette, der Directory, BAM oder des Programms selber anzuwenden. Es können so schnell ganze Dateien und Disketteninhalte verloren gehen. Lesen Sie dazu also auch genaustens das Handbuch zur Commodore Floppy 1541 durch. Sie werden verstehen, daß wir an dieser Stelle nicht auch noch ausführlich auf den ganzen Aufbau von Directory, BAM etc. eingehen können. Das würde den Umfang dieses Buches sprengen. Wenn Sie aber gerne mit diesen Möglichkeiten experimentieren möchten, so empfehlen wir mit einer Testdiskette zu arbeiten - dann können zumindest keine "lebenswichtigen" Daten verloren gehen.

Zur Verdeutlichung der angewendeten Befehle hier noch eine Liste der Befehle, die direkt auf einen Block zugreifen können:

Bezeichnung	Anwendung
-----	-----
BLOCK-LESEN	"B-R: ";Kanal;Laufwerk;Track;Block
BLOCK-SCHREIBEN	"B-W: ";Kanal;Laufwerk;Track;Block
BLOCK-BELEGEN	"B-A: ";Laufwerk;Track;Block
BLOCK-FREIGEBEN	"B-F: ";Laufwerk;Track;Block
PUFFER-ZEIGER	"B-P: ";Kanal;Position

Dieses sind die wichtigsten Befehle zum direkten Zugriff. Allen ist gemeinsam, daß Sie direkt auf den Floppy-Controller zugreifen. So haben Sie also Möglichkeiten, die Sie bei einer gewöhnlichen Zugriffsart nie haben werden.

8.6 RETTEN EINER NICHT ORDNUNGSGEMÄß GESCHLOSSENEN DATEI

Zugegeben, es passiert nicht oft. Aber wenn es passiert, ist der damit verbundene Ärger und Arbeitsaufwand um so größer. Was ist "es"?

Mit "es" meinen wir folgenden Vorgang:

Sie haben mit viel Mühe Ihre Schallplattensammlung geordnet und möchten Sie zwecks schneller Auffindung bestimmter Titel auf einer Diskette verewigen.

Das wohl gebräuchlichste Verfahren hierzu besteht einfach darin, daß man den Plattennamen samt der darauf befindlichen Titel auf einer oder mehreren sequentiellen Dateien abspeichert.

Sie sind also im schönsten Zuge, die gewünschten Daten über die Tastatur einzugeben und sind auch fast fertig (500 Titel haben Sie schon aufgenommen), da fliegt die Sicherung raus oder Ihre Frau (oder Mann) stolpert über den Netzstecker.

Macht nichts, denken Sie, die bisher eingegebenen Daten habe ich ja sicher auf der Diskette.

Sie gehen wieder in Ihr Programm, ändern das OPEN-Kommando für Ihre sequentielle Datei für APPEND (A anstelle von W) und wollen weitermachen.

Schreck laß' nach, die Fehlerlampe an der Diskettenstation leuchtet auf.

Ein nachfolgendes Abfragen des Fehlerkanals bringt des Rätsels Lösung: WRITE FILE OPEN.

Wenn Sie sich nun das Inhaltsverzeichnis der Diskette anschauen, so finden Sie vor der Typenkennzeichnung der Datei ein '*'.
Das bedeutet, daß die Datei noch zum Schreiben geöffnet ist,

d.h. nach einem Schreibzugriff ist kein CLOSE erfolgt, und genau das ist bei dem Stromausfall passiert. Das gleiche geschieht übrigens auch, wenn Sie die Diskette aus dem

Laufwerk entfernen, ohne vorher einen CLOSE abgesetzt zu haben.

Sie haben mit gewöhnlichen Methoden nun keine Chance mehr, an Ihre Daten heranzukommen. Schade um die Plattensammlung. Da auch uns dies oft genug passiert ist, haben wir aus der Not eine Tugend gemacht und ein kleines Programm entwickelt, welches es ermöglicht, derart gestörte Dateien wenigstens wieder lesbar zu machen.

Wenn Sie sich parallel zu dem nun folgenden noch einmal den Abschnitt 8.1 zu Gemüte führen, werden Sie auch verstehen, was genau in diesem Programm gemacht wird.

Selbstverständlich werden wir wieder jeden Programmschritt genau erklären.

Zuerst seien Ihnen die verwendeten Variablen vorgestellt:

- E Position von Filename innerhalb des Directorysektors
- S Sektornummer für die Direktbefehle
- T Tracknummer für die Direktbefehle
- TY Dateityp (aus T\$ abgeleitet)
- X Laufvariable zur Ermittlung der tatsächlichen des
 Filenamens
- A\$ Interimvariable zum Aufbau von S\$
- F\$ gewünschter Filename
- S\$ vollständiger Sektor
- T\$ Dateityp
- X\$ Aus dem Directory gelesener, auf 16 Zeichen expandierter
 Filename, später echter Filename
- X1\$ Duplikat von X\$

Nun das Programm Schritt für Schritt:

- 70 Öffnen eines Datenkanals für den Direktzugriff
- 80 Öffnen des Kommandokanals
- 100 Eingabe des zu behandelnden Dateinamens
- 110 Zuweisung von Track- und Sektornummer.
Bei der Floppy VC-1541 und der CBM4040 beginnt das Inhaltsverzeichnis auf Spur 18, Sektor 1.
Bei der Floppy CBM8050, die Sie bei vorhandenem IEC-Bus ebenfalls behandeln können, liegt es auf Spur 39. In diesem Falle muß diese Zeile entsprechend geändert werden.
- 120 In dieser Zeile wird der durch T und S spezifizierte Sektor von der Diskette (Drive 0) zunächst in den floppyinternen Puffer gelesen.
- 150 Der Pufferinhalt wird über A\$ nach S\$ gebracht.
- 160 Ein Sektor kann acht Directoryeinträge aufnehmen.
Dieser wird zunächst nach dem gewünschten Dateinamen durchsucht, bevor der nächste Sektor eingelesen wird.
- 170 Hier wird der Dateiname aus dem Eintrag isoliert, und zwar in der vollen Länge von 16 Bytes, und nach X\$ gebracht.
- 200-210 Das Endekriterium für die echte Länge des Filenamens ist CHR\$(160) (shifted space). Hiernach wird der Dateiname abgesucht und dann in der echten Länge wieder nach X\$ gebracht.
- 220-230 Wurde der gesucht Name gefunden, geht es bei 300 weiter, ansonsten werden die übrigen noch in diesem Sektor enthaltenen Einträge untersucht.
- 240-260 Am Beginn jedes Sektors steht die Track- und Sektoradresse des Folgeblockes, falls es einen gibt. Gibt es keinen, d.h. ist das Directory zu Ende, so ist die Tracknummer =0.

- 300-310 Der Dateityp (das ist das Byte, aus dem die Dateiart für den Bildschirm generiert wird) wird isoliert und nach T\$, bzw. als numerischer Wert nach T gebracht.
- 320 T=0 kennzeichnet einen leeren Eintrag im Directory.
- 360 Das Bit, welches hier gesetzt wird, ist der Urheber des ganzen Übels. Dieses Bit kennzeichnet nämlich, ob eine Datei zum Schreiben geöffnet ist oder nicht. Aus diesem Bit wird auch das '*' für den Bildschirm abgeleitet.
- 370 Nun wird der ganze Sektor mit der Kennung für eine geschlossene Datei wieder zusammengebaut.
- 390-410 Jetzt wird der Pufferzeiger in der Floppy wieder zurückgesetzt, der Sektor in den Puffer gebracht und der Pufferinhalt auf die Diskette zurückgeschrieben.
- 420-490 Diese Zeilen sollen Ihnen als Gedächtnisstütze dienen, wie nun weiter verfahren werden kann.

Die Anwendung des Programmes ist recht einfach:

Sie laden das Programm, stecken die zu behandelnde Diskette in das Laufwerk (bei Doppeldrives muß es Laufwerk 0 sein), starten es und geben den zu behandelnden Dateinamen ein.

Befindet sich die Datei auf der Diskette, so können Sie mit einem erfolgreichen Ausgang der Aktion rechnen.

Eine Einschränkung gibt es allerdings:

Bei relativen Dateien ist dieses Verfahren leider nicht anwendbar, da deren Struktur von der der übrigen Dateiarten gänzlich abweicht. Die Organisation relativer Dateien kann nur mit einem weitaus größeren Aufwand rekonstruiert werden, wobei intime Kenntnisse über deren Aufbau unbedingte Voraussetzung sind.

Haben Sie nun also Ihre Datei gerettet, so sollten Sie sie Satz für Satz lesen und jeden zu dieser Datei gehörenden

Eintrag auf eine neue Datei übertragen.

Das ist deswegen erforderlich, weil zwar die Daten gerettet sind, jedoch das logische Ende nicht mehr erkannt werden kann, d.h. im Anschluß an Ihre Daten mögen noch weitere auftauchen, die mit Ihrer Datei nichts mehr zu tun haben.

An diesem Punkt sollten Sie den Vorgang abbrechen und die neue Datei unbedingt schließen und die defekte Datei löschen.

Jetzt erst können Sie beruhigt mit der Aufnahme Ihrer Schallplattensammlung fortfahren.

Wir hoffen, Ihnen mit diesem Programm ein nützliches Werkzeug zur Verfügung gestellt zu haben, welches Sie hoffentlich nicht zu oft einsetzen müssen.

```
10 print chr$(147);"  
20 print chr$(5);  
70 open 2,8,2,"#":rem direktzugriff  
80 open 15,8,15:rem kommandokanal  
90 print:print  
100 input "filename ";f$:print:print  
110 t=18:s=1:rem 1541 directory ** t=39 bei cbm8050  
120 print#15,"u1 2 0"t;s:rem lesen  
130 s$="":rem variable fuer gelesenen  
140 :      :rem sektor  
150 for i=1 to 255:get#2,a$:s$=s$+left$(a$+chr$(0),1):next  
160 for i=0 to 7:rem 8 eintraege  
170 x$=mid$(s$,i*32+6,16):x1$=x$  
180 rem filenamen isolieren  
190 x=1  
200 if mid$(x$,x,1)<>chr$(160) then x=x+1:if x<17 then 200  
210 x$=left$(x$,x-1)
```

```

220 if x$=f$ then e=i:goto 300
230 next i
240 t=asc(s$):s=asc(mid$(s$,2,1))
250 rem naechsten sektor lesen
260 if t<>0 then 120
270 rem ende
280 print"file "f$" nicht auf dieser diskette"
290 close 2:close 15:end
300 t$=mid$(s$,e*32+3)
310 ty=asc(t$) and 15
320 if ty=0 then next i:goto 240
330 if ty<>4 then 340
335 print"relative files koennen nicht verarbeitet werden"
337 goto 290
340 ty$="delseqprgusrrel"
350 print"file "x1$" "mid$(ty$,ty*3+1,3):print
360 t$=chr$(asc(t$) or 128)
370 s$=left$(s$,e*32+1)+t$+mid$(s$,e*32+3)
380 rem * loeschen und rueckschreiben
390 print#15,"b-p 2 0"t;s
400 print#2,s$;
410 print#15,"u2 2 0"t;s
420 close 2:close 15
425 print"filedaten koennen jetzt gelesen werden."
430 print"nach dem kopieren der gueltigen daten"
440 print"sollten die folgenden befehle gegeben"
450 print"werden :":print
460 print"open 15,8,15"
470 print chr$(17)"print#15,"chr$(34)"s:"f$chr$(34)
480 print chr$(17)"print#15,"chr$(34)"v0"chr$(34)
490 print chr$(17)"close 15"
500 end

```

8.7 - DER BLOCKVERFOLGER -

Hatten Sie nicht auch schon immer einmal den Wunsch sich den Aufbau Ihrer Diskette und die Organisation der einzelnen Dateien anzusehen? Wollen Sie wissen, wie lang ein Programm genau ist? Vielleicht möchten Sie einen Blick auf die gespeicherten Daten werfen, um festzustellen, warum die eigene Dateiverwaltung immer mit einem "String too long Error" aussteigt? Das und noch einige andere Funktionen sind mit dem Blockverfolger möglich.

Das Programm ist weitgehend modular aufgebaut. Somit lassen sich weitere Funktionen ohne großen Aufwand zusätzlich implementieren.

```
100 HX$="0123456789ABCDEF":DIMK$(75):FOR
I=1TO75:READK$(I):NEXTI
110 REM *****
120 REM BLOCKVERFOLGUNG
130 REM *****
140 PRINT CHR$(147)
150 PRINT"(1) DIREKTORY "
160 PRINT"(2) BLOCKVERKETTUNG "
170 PRINT"(3) BLOCKAUSGABE ASC"
180 PRINT"(4) BLOCKAUSGABE TOKENS"
190 PRINT"(5) FEHLERKANAL"
200 PRINT:PRINT"TREFFEN SIE IHRE WAHL"
210 GOSUB 990
220 IF A < 1 OR A > 5 THEN GOTO 210
230 ON A GOSUB 250,450,540,710,880
240 GOTO 140
250 REM DIREKTORY *****
260 T=18 : S=1
270 OPEN1,8,15 : OPEN2,8,2,"#"
280 PRINTCHR$(147);"EINTRAG          TR
SE"
290 PRINT#1,"U1:"2;0;T;S
300 GOSUB 1030 : T = TX : S = SX
310 FOR I = 0 TO 7
320 PRINT#1,"B-P:"2;I*32+2
330 GET#2,A$:IFA$=""THENA$=CHR$(0)
340 IFASC(A$) < 128 THEN 410
350 GOSUB1030
360 FOR U = 1TO16
370 GET#2,A$:IFA$=""THENA$=" "
380 IF ASC(A$) =160 THENA$=" "
390 N$ = N$ + A$ : NEXT U
```



```

810 NEXT U
820 PRINT N$:N$=""
830 NEXT I
840 PRINT:PRINT"BITTE EINE TASTE ODER FO
LGE BLOCK MIT 'F'"
850 GOSUB 990 : IF A$ <> "F" THEN CLOSE1
:CLOSE2:RETURN
860 T=TX: S= SX : IF T=0 THEN CLOSE1:CLOSE
2:RETURN
870 GOTO 730
880 REM FEHLERKANAL *****
890 OPEN1,8,15
900 INPUT#1,A$,S$,D,F
910 PRINT CHR$(147)
920 PRINT"FEHLERNUMMER :";A
930 PRINT:PRINT"Typ : ";S$
940 PRINT:PRINT"BLOCK :";D
950 PRINT:PRINT"SEKTOR :";F
960 CLOSE1
970 PRINT:PRINT"BITTE EINE TASTE"
980 GOSUB 990 : RETURN
990 REM ZEICHEN HOLEN *****
1000 GET A$ : IF A$ = "" THEN GOTO 1000
1010 A = VAL(A$)
1020 RETURN
1030 REM 2 BYTE HOLEN *****
1040 GET#2,A$: IFA$="" THEN A$=CHR$(0)
1050 TX = ASC(A$)
1060 GET#2,A$: IFA$="" THEN A$=CHR$(0)
1070 SX = ASC(A$)
1080 RETURN
1090 REM EINGABE TRACK SEKTOR *****
1100 PRINT CHR$(147)
1110 INPUT"SPUR (TRACK) ";T
1120 INPUT"SEKTOR      ";S
1130 OPEN1,8,15:OPEN2,8,2,"#"
1140 RETURN
1150 REM BLOCK LESEN *****
1160 PRINT#1,"U1:"2;0;T;S
1170 GOSUB1030:PRINT#1,"B-P:"2;0
1180 RETURN
1190 REM HEXWANDEL
1200 A=ASC(A$)
1210 A1=INT(A/16)
1220 A2=A-A1*16
1230 H$=MID$(HX$,A1+1,1)
1240 H$=H$+MID$(HX$,A2+1,1)
1250 RETURN

```

```

400 PRINTN$;TX;SX:N$=""
410 NEXT I
420 IF T<> 0 GOTO 290
430 PRINT:PRINT"BITTE EINE TASTE"
440 CLOSE1:CLOSE2:GOSUB990:RETURN
450 REM VERKETTUNG *****
460 GOSUB 1090:C=1
470 PRINT"BLOCK NR.;"C;" TR:";T;" SE
:";S
480 PRINT#1,"U1:"2;0;T;S
490 GOSUB 1030:T=TX:S=SX:C=C+1:IFT=0THEN
GOTO510
500 GOTO 470
510 PRINT"LETZTER BLOCK HAT :";S;"BYTES"
520 CLOSE1:CLOSE2:PRINT:PRINT"BITTE EINE
TASTE"
530 GOSUB 990 : RETURN
540 REM AUS ASC *****
550 GOSUB 1090
560 GOSUB 1150
570 FOR I = 0 TO 31
580 FOR U = 0 TO 7
590 GET#2,A$:IFA$=""THENA$=CHR$(0)
600 GOSUB 1190
610 IFASC(A$)<32ORASC(A$)>127ANDASC(A$)<
160THENA$="."
620 N$=N$+A$
630 PRINTH$;" ";
640 NEXT U
650 PRINT N$:N$=""
660 NEXT I
670 PRINT:PRINT"BITTE EINE TASTE ODER FO
LGEBLOCK MIT 'F'"
680 GOSUB 990 : IF A$ <> "F" THEN CLOSE1
:CLOSE2:RETURN
690 T=TX: S=SX :IF T=0 THEN CLOSE1:CLOSE
2:RETURN
700 GOTO 560
710 REM AUS TOKEN *****
720 GOSUB 1090
730 GOSUB 1150
740 FOR I = 0 TO 31
750 FOR U = 0 TO 7
760 GET#2,A$:IFA$=""THENA$=CHR$(0)
770 GOSUB 1190
780 IF ASC(A$)<32THENA$="."
790 IFASC(A$)>127ANDASC(A$)<203THENA$=K$
(ASC(A$)-127)
800 N$=N$+A$:PRINTH$;" ";

```

```

1260 DATAEND,FOR,NEXT,DATA,INPUT#,INPUT,
DIM,READ,LET,GOTO,RUN,IF,RESTORE,GOSUB
1270 DATARETURN,REM,STOP,ON,WAIT,LOAD,SA
VE,VERIFY,DEF,POKE,PRINT#,PRINT,CONT
1280 DATALIST,CLR,CMD,SYS,OPEN,CLOSE,GET
,NEW,TAB<,TO,FN,SPC<,THEN,NOT,STEP
1290 DATA+,-,/,*,^,AND,OR,>,<,>SGN,INT,
ABS,USR,FRE,POS,SQR,RND,LOG,EXP,COS,SIN
1300 DATATAN,ATN,PEEK,LEN,STR$,VAL,ASC,C
HR$,LEFT$,RIGHT$,MID$,,

```

Erklärung zum Listing

```

-----
100      String zur Hexadezimalumwandlung und Aufbau der
        Basic-Schlüsselworte
110-200  Ausgabe des Menues
210-240  Aufruf des gewählten Unterprogramms mit
        anschließendem Rücksprung zum Menue
250-300  Vorlauf zum Lesen der Directory
310-330  Positionierung auf einen Eintrag
340      Prüfung auf "file scratched"
350-410  Aufbau und Ausgabe eines Eintrags
420      Folgeblock der Directory auslesen
430-440  Nachlauf
450-460  Vorlauf der Blockverkettung
470      Ausgabe der Blocknummer mit den zugehörigen
        Werten
480-500  Folgeblock holen
510-530  Nachlauf
540-580  Vorlauf ASC Ausgabe
590      Ein Byte holen
600      In hexadezimalen Format wandeln
610      Prüfung auf druckbaren ASC Wert
620-660  Ausgabe der Hexwerte und Aufbau des ASC Strings
670-700  Nachlauf
710-750  Vorlauf Token Ausgabe
760      Ein Byte holen
770      In hexadezimalen Format wandeln

```

780-790 Prüfen ob Token oder ASC Wert
800-830 Ausgabe der Hexwerte und des ASC/Token Strings
840-870 Nachlauf
880-980 Fehlerkanal lesen, ausgeben und Nachlauf
990-1020 Unterprogramm 1 - Warten auf eine Taste und deren
numerischen Wert ermitteln
1030-1080 1. und 2. Byte eines Blocks holen und in die
Adresse des Folgeblocks umrechnen
1090-1140 Einfache Input Abfrage für den frei wählbaren
Startpunkt der Blockverfolgung
1150-1180 Einen Block lesen und den Pufferzeiger rücksetzen
1190-1250 Umwandlung eines 1 Byte Dezimalwertes in hexadezi-
male Form
1260-1300 Bezeichnung der Basic-Schlüsselwerte

KAPITEL 9 POKE'S UND ANDERE NÜTZLICHE ROUTINEN

9.1 DER KASSETTENPUFFER ALS PROGRAMMSPEICHER

Will man kleine Maschinenprogramme zusammen mit BASIC verwenden, stellt sich immer die Frage in welchem Speicherbereich man solche Programme ablegt.

Bei der Auswahl muß beachtet werden, daß dieser Bereich nicht von BASIC-Programmen oder Variablen überschrieben werden darf. Unter diesem Gesichtspunkt gibt es zwei grundsätzliche Möglichkeiten zum Ablegen von Maschinenprogrammen.

Zum ersten kann man Speicherbereiche auswählen, die vom BASIC-Interpreter nicht benutzt werden, zum zweiten ist das Verlegen des BASIC-Programmstarts oder des Endes des BASIC-Speichers möglich.

Unbenutzt von BASIC sind wieder drei Bereiche. Als erstes bietet sich der Kassettenpuffer an. Er liegt von Adresse 828 bis 1019 (\$033C bis \$03FB). Dieser Bereich wird vom Programm nur dann benutzt, wenn Sie Daten mit dem Kassettenrekorder abspeichern oder laden. Ansonsten ist dieser Bereich unbenutzt. Er eignet sich damit hervorragend zur Speicherung kleiner Maschinenprogramme bis 192 Bytes Länge. Falls Sie die Sprites 13, 14 oder 15 benutzen, ist der Kassettenpuffer durch die Sprites belegt. Ein weiterer kleiner Bereich steht Ihnen von Adresse 704 bis 767 (\$02C0 bis \$02FF) zur Verfügung (64 Bytes). Dieser Bereich wird jedoch von Sprite 11 belegt. Einen großen Bereich von 4 KBytes haben Sie im RAM oberhalb des BASIC-Interpreters von Adresse 49152 bis 53247 (\$C000 bis \$CFFF), der auch für längste Maschinenprogramme ausreichen dürfte.

Braucht man lediglich ein paar Speicherzellen, um z.B.

Variablenwert abzulegen, die auch nach einem CLR- oder NEW-Befehl noch erhalten bleiben sollen, so hat man dazu 16 Bytes 'hinter' dem Bildschirmspeicher zur Verfügung. Der 64er hat für den Bildschirmspeicher 1K Byte = 1024 Byte zur Verfügung; für das Video-RAM werden jedoch nur $25 \times 40 = 1000$ Bytes benötigt. 24 Bytes sind also übrig. Davon werden jedoch die letzten 8 Bytes als Zeiger für die Sprites benutzt. Es bleiben also 16 Bytes übrig, die Sie für Ihre Zwecke nutzen können. Es sind dies die Adressen 2024 bis 2039 (hexadezimal \$07E9 bis \$07F8).

Sind die oben beschriebenen Bereiche belegt oder brauchen Sie mehr Speicherplatz, bietet es sich an, den Bereich für BASIC-Programme einzuschränken und damit für Maschinenprogramme nutzbar zu machen. Hierbei können Sie das BASIC-Ende runtersetzen (die allgemein übliche Methode) oder den BASIC-Start nach oben legen. Sehen wir uns jetzt an, wie man dies macht.

Der BASIC-Interpreter hat zwei Zeiger, die jeweils auf Start und Ende des BASIC-Speichers zeigen. Der BASIC-Start steht in 43/44 (\$2B/\$2C), das Ende in 55/56 (\$37/\$38). Mit

```
PRINT PEEK(43)+256*PEEK(44) bzw.
```

```
PRINT PEEK(55)+256*PEEK(56)
```

lassen sich die Werte holen; im Normalfall steht dort 2049 bzw. 40960. Wollen wir Platz für ein 1000 Byte langes Maschinenprogramm schaffen, können wir das BASIC-Ende um 1000 nach unten legen. Das Ende müßte also jetzt bei 39960 liegen. Jetzt können wir den neuen Wert mittels POKE setzen.

```
HB = INT (39960/256) : LB = 39960 - HB*256
```

```
POKE 55,LB : POKE 56,HB : CLR
```

Der CLR-Befehl ist unbedingt erforderlich, da Sie sonst falsche Variablenwerte erhalten können. Wollen wir den Start auf 3049 legen, sind folgende Befehle erforderlich:

```
HB = INT (3049/256) : LB = 3049 - HB*256  
POKE 43, LB : POKE 44,HB : POKE 3049-1,0 : NEW
```

Hier ist der NEW-Befehl erforderlich, um alle anderen BASIC-Zeiger richtig zu setzen.

9.2 SORTIEREN VON STRINGS

Eine Aufgabe, auf die jeder Programmierer des öfteren stößt, ist das Sortieren irgendwelcher Daten. Das können z.B. Namen, Adressen und Zahlenreihen sein. Zum Sortieren gibt es verschiedene bekannte Algorithmen, die jedoch alle sehr zeitaufwendig sind, sobald eine größere Anzahl von Daten zu sortieren sind. Die einfachsten Verfahren sind im allgemeinen auch die langsamsten. Braucht man entschieden schnellere Sortierverfahren, so muß man die Algorithmen nicht in BASIC, sondern in Maschinensprache formulieren. Bei solchen Aufgaben sind Lösungen in Maschinensprache gut 100mal so schnell wie eine BASIC-Routine. Das folgende Programm ist zur Sortierung eines Stringarrays geeignet. Um das Programm möglichst kurz zu halten, wurde von folgenden Voraussetzungen ausgegangen:

1. Das zu sortierende Feld muß als erstes mit DIM dimensioniert werden.
2. Nach dem letzten zu sortierenden Arrayelement muß ein Leerstring folgen.

Punkt 2 hat zudem den Vorteil, daß auch ein nur zum Teil gefülltes Array sortiert werden kann, ohne daß sämtliche Leerstrings nach dem Sortieren am Anfang des Arrays stehen. Mit diesen Vereinbarungen wird das Programm so kurz, daß es im Kassettenpuffer Platz hat. Es wird einfach mit SYS 828 aufgerufen. Das Programm überprüft, ob das erste Array ein eindimensionales Stringarray ist. Ist dies nicht der Fall, wird das Maschinenprogramm direkt beendet.

100:	033c		*= 828
105:	033c		.opt p1
110:	033c a0 00		ldy #0
120:	033e b1 2f		lda (\$2f),y ; erster Buchstabe
130:	0340 30 0d		bmi 11
135:	0342 c8		iny
140:	0343 b1 2f		lda (\$2f),y ; zweites Zeichen
150:	0345 10 08		bpl 11
160:	0347 a0 04		ldy #4
170:	0349 b1 2f		lda (\$2f),y ; Dimensionen
180:	034b c9 01		cmp #1
190:	034d f0 01		beq 12
200:	034f 60	11	rts
210:	0350 18	12	clc
220:	0351 a5 2f		lda \$2f ; Arraystart
230:	0353 69 07		adc #7 ; plus 7
240:	0355 85 6e		sta \$6e
250:	0357 a5 30		lda \$30
260:	0359 69 00		adc #0
270:	035b 85 6f		sta \$6f
280:	035d a0 00	13	ldy #0
290:	035f b1 6e		lda (\$6e),y
300:	0361 f0 ec		beq 11 ; Länge null, fertig
310:	0363 85 22		sta \$22
320:	0365 c8	14	iny
330:	0366 b1 6e		lda (\$6e),y
340:	0368 99 22 00		sta \$22,y ; Zeiger auf String
350:	036b c0 02		cpy #2
360:	036d d0 f6		bne 14
370:	036f a5 6e		lda \$6e
380:	0371 85 71		sta \$71
390:	0373 a5 6f		lda \$6f
400:	0375 85 72		sta \$72
410:	0377 18	15	clc

420:	0378 a5 71		lda \$71	
430:	037a 69 03		adc #3	; drei addieren
440:	037c 85 71		sta \$71	
450:	037e 90 02		bcc 16	
460:	0380 e6 72		inc \$72	
470:	0382 a0 00	16	ldy #0	
480:	0384 b1 71		lda (\$71),y	
490:	0386 f0 3d		beq 113	
500:	0388 85 4d		sta \$4d	; Länge
510:	038a c5 22		cmp \$22	; mit erster Länge vergleichen
520:	038c 90 02		bcc 17	
530:	038e a5 22		lda \$22	
540:	0390 85 55	17	sta \$55	; Vergleichslänge
550:	0392 c8	18	iny	
560:	0393 b1 71		lda (\$71),y	
570:	0395 99 4d 00		sta \$4d,y	
580:	0398 c0 02		cpy #2	
590:	039a d0 f6		bne 18	
600:	039c a0 00		ldy #0	
610:	039e b1 23	19	lda (\$23),y	; Stringvergleich
620:	03a0 d1 4e		cmp (\$4e),y	
630:	03a2 f0 04		beq 110	; gleich, dann weiter
640:	03a4 b0 0b		bcs 111	; größer dann Tausch
650:	03a6 90 cf		bcc 15	; kleiner dann nächsten String
660:	03a8 c8	110	iny	
670:	03a9 c4 55		cpy \$55	; alle Zeichen gleich ?
680:	03ab d0 f1		bne 19	
690:	03ad c4 22		cpy \$22	; erster String länger
700:	03af b0 c6		bcs 15	; nein dann ok
710:	03b1 a0 02	111	ldy #2	
720:	03b3 b1 6e	112	lda (\$6e),y	; Stringzeiger vertauschen
730:	03b5 aa		tax	
740:	03b6 b1 71		lda (\$71),y	
750:	03b8 91 6e		sta (\$6e),y	

```

760: 03ba 99 22 00      sta $22,y
770: 03bd 8a            txa
780: 03be 91 71          sta ($71),y
790: 03c0 88             dey
800: 03c1 10 f0           bpl 112
810: 03c3 30 b2           bmi 15
820: 03c5 18             113      clc           ; Zeiger auf nächsten String
830: 03c6 a5 6e           lda $6e
840: 03c8 69 03           adc #3
850: 03ca 85 6e           sta $6e
860: 03cc 90 8f           bcc 13
870: 03ce e6 6f           inc $6f
880: 03d0 d0 8b           bne 13

```

Hier wieder das Ladeprogramm in BASIC.

```

100 for i = 828 to 977
110 read x : poke i,x : s=s+x : next
120 data 160, 0,177, 47, 48, 13,200,177, 47, 16, 8,160
130 data 4,177, 47,201, 1,240, 1, 96, 24,165, 47,105
140 data 7,133,110,165, 48,105, 0,133,111,160, 0,177
150 data 110,240,236,133, 34,200,177,110,153, 34, 0,192
160 data 2,208,246,165,110,133,113,165,111,133,114, 24
170 data 165,113,105, 3,133,113,144, 2,230,114,160, 0
180 data 177,113,240, 61,133, 77,197, 34,144, 2,165, 34
190 data 133, 85,200,177,113,153, 77, 0,192, 2,208,246
200 data 160, 0,177, 35,209, 78,240, 4,176, 11,144,207
210 data 200,196, 85,208,241,196, 34,176,198,160, 2,177
220 data 110,170,177,113,145,110,153, 34, 0,138,145,113
230 data 136, 16,240, 48,178, 24,165,110,105, 3,133,110
240 data 144,143,230,111,208,139
250 if s <> 17663 then print "fehler in datas !!" : end
260 print "ok !"

```

Die Geschwindigkeit des Maschinenprogramms können Sie mit einem kleinen Testprogramm demonstrieren.

Das Programm erzeugt eine vorgegebene Anzahl von String aus Zufallsbuchstaben mit einer ebenfalls einzugebenden maximalen Länge, zeigt sie am Bildschirm an, sortiert sie dann und gibt sie zusammen mit der Zeit für das Sortieren wieder aus.

```
100 input "anzahl, laenge ";a,l
110 dim a$(a):a=a-1
120 for i=0 to a
130 for j=1 to rnd(1)*l
140 a$(i) = a$(i)+chr$(rnd(1)*26+65)
150 next:next
160 for i=0 to a : print a$(i) : next
170 t=ti : sys 828 : t=ti-t
180 print "sortierzeit " t/60 "sekunden"
190 for i=0 to a : print a$(i) : next
```

Lassen Sie das Programm einmal mit verschiedenen Längen und Anzahl von Strings laufen, und beachten Sie die Sortierzeit. 100 Strings werden unter einer Sekunde sortiert. Ein entsprechendes BASIC-Programm würde dazu Minuten brauchen.

Wenn Sie das Programm für Ihre Zwecke einsetzen wollen, beachten Sie bitte, daß das letzte Element (bis zu dem sortiert werden soll) ein Leerstring sein muß und daß das Array als erstes dimensioniert werden muß.

9.3 MINIMUM UND MAXIMUM VON FELDVARIABLEN

Bei Berechnungen mit dimensionierten Variablen benötigt man oft den kleinsten oder größten Wert des Feldes. Die Berechnung läßt sich zwar mit einer kleinen BASIC-Schleife lösen, dauert jedoch bei größeren Feldern relativ lange. Dies ist ein typischer Fall für die Maschinenprogrammierung. Das Programm verwendet denselben Algorithmus wie die entsprechende BASIC-Variante.

```
100 DIM A(N)
...
200 GOSUB 1000
...
1000 MIN = A(0)
1010 FOR I=1 TO N
1020 IF A(I) < MIN THEN MIN = A(I)
1030 NEXT
1040 RETURN
```

Ein Feld A wird von 0 bis N dimensioniert. Durch Aufruf des Unterprogramms ab Zeile 1000 wird das Minimum berechnet und in der Variablen MIN zurückgegeben. Ist das Maximum gesucht, ersetzt man lediglich Zeile 1020 durch

```
1020 IF A(I) > MAX THEN MAX = A(I)
```

In Zeile 1000 wird entsprechend $MAX = A(0)$ geschrieben.

Das Maschinenprogramm hat gegenüber dem obigen BASIC-Programm noch den Vorteil, daß es nicht auf die Variable A beschränkt

ist, sondern daß der Variablenname als Parameter der
 USR-Funktion übergeben wird. Das Programm kann sowohl reelle
 Zahlen (A) als auch Integerarrays (A%) verarbeiten und ist im
 freien RAM-Bereich ab Adresse \$C800 angesiedelt.

```

                                ; Min/Max Funktion
120:  c800      intflg  =   14      ; Flag für Integervariable
130:  c800      store  =   $26
140:  c800      arrtab  =   $2f      ; Zeiger auf Arraytabelle
150:  c800      arrend  =   $31      ; Zeiger auf Ende Arrays
160:  c800      varnam  =   $45      ; Variablenname
170:  c800      temp   =   $5f
180:  c800      setarr  =   $b196    ; Zeiger auf erstes Arrayelement
190:  c800      memfac  =   $bba2    ; Konstante in FAC holen
200:  c800      compare =   $bc5b    ; Konstante mit FAC vergleichen
210:  c800      errout  =   $a445
220:  c800      int     =   $14      ; Speicher für Integervariable
230:  c800      intfloat = $b391     ; Integer nach FAC
240:  c800      *=      =   $c800
300:  c800 a6 2f      minmax  ldx  arrtab
310:  c802 a5 30      lda  arrtab+1 ; Zeiger auf Beginn der Arraytabelle
320:  c804 86 5f      13      stx  temp
330:  c806 85 60      sta  temp+1 ; laufender Zeiger
340:  c808 c5 32      cmp  arrend+1
350:  c80a d0 04      bne  11
360:  c80c e4 31      cpx  arrend ; Ende der Arraytabelle ?
370:  c80e f0 1d      beq  notfound
380:  c810 a0 00      11      ldy  #0
390:  c812 b1 5f      lda  (temp),y ; erster Buchstabe des Namens
400:  c814 c8          iny
410:  c815 c5 45      cmp  varnam ; mit gesuchtem Namen vergleichen
420:  c817 d0 06      bne  12
430:  c819 a5 46      lda  varnam+1

```

```

440:  c81b d1 5f          cmp (temp),y ; zweites Zeichen vergleichen
450:  c81d f0 17          beq found ; gefunden ?
460:  c81f c8             12    iny
470:  c820 b1 5f          lda (temp),y
480:  c822 18             clc
490:  c823 65 5f          adc temp ; Offset für nächstes Array addieren
500:  c825 aa             tax
510:  c826 c8             iny
520:  c827 b1 5f          lda (temp),y
530:  c829 65 60          adc temp+1
540:  c82b 90 d7          bcc l3
550:  c82d a9 b8          notfound lda #<errmsg ; Zeiger auf Fehlermeldung
560:  c82f 85 22          sta $22
570:  c831 a9 c8          lda #>errmsg
580:  c833 4c 45 a4        jmp errout ; Fehlermeldung ausgeben
590:  c836 c8             found  iny
600:  c837 18             clc
600:  c838 b1 5f          lda (temp),y
610:  c83a 65 5f          adc temp
620:  c83c 85 26          sta store
630:  c83e c8             iny
640:  c83f b1 5f          lda (temp),y
650:  c841 65 60          adc temp+1
660:  c843 85 27          sta store+1 ; Zeiger auf Ende des Arrays
670:  c845 c8             iny
680:  c846 b1 5f          lda (temp),y ; Anzahl der Dimensionen
690:  c848 20 96 b1        jsr setarr ; Zeiger auf erstes Arrayelement
700:  c84b 85 5f          sta temp
710:  c84d 84 60          sty temp+1 ; Zeiger merken
720:  c84f 24 0e          bit intflg ; Typ testen
730:  c851 30 24          bmi integer
740:  c853 10 09          bpl lp1
770:  c855 20 5b bc 15     jsr compare ; Arrayelement vergleichen
785:  c858 10 07          bpl loop

```

```

790:  c85a a5 5f          lda  temp
800:  c85c a4 60          ldy  temp+1
810:  c85e 20 a2 bb lp1    jsr  memfac ; Arrayelement als Min/Max merken
820:  c861 18             loop  clc
830:  c862 a5 5f          lda  temp
840:  c864 69 05          adc  #5      ; Zeiger auf nächstes Element
850:  c866 85 5f          sta  temp
860:  c868 90 02          bcc  14
870:  c86a e6 60          inc  temp+1
880:  c86c a4 60          ldy  temp+1
890:  c86e c5 26          cmp  store ; Ende des Arrays
900:  c870 d0 e3          bne  15
910:  c872 c4 27          cpy  store+1
920:  c874 d0 df          bne  15
930:  c876 60             rts
950:  c877 a0 00          integer ldy  #0      ; Integerarray
960:  c879 b1 5f          lda  (temp),y
970:  c87b aa             tax
980:  c87c c8             iny
980:  c87d b1 5f          lda  (temp),y
990:  c87f 85 15          sta  int+1 ; ersten Wert nach int holen
1000: c881 86 14         stx  int
1010: c883 18             112    clc
1020: c884 a5 5f          lda  temp
1030: c886 69 02          adc  #2      ; Zeiger auf nächstes Element
1040: c888 85 5f          sta  temp
1050: c88a 90 02          bcc  110
1060: c88c e6 60          inc  temp+1
1070: c88e c5 26          110    cmp  store
1080: c890 d0 0d          bne  111
1090: c892 a5 60          lda  temp+1
1100: c894 c5 27          cmp  store+1 ; Ende schon erreicht ?
1110: c896 d0 07          bne  111
1120: c898 a5 14          lda  int      ; Integerwert holen

```



```

1130: c89a a4 15          ldy int+1
1140: c89c 4c 91 b3       jmp intfloat ; nach FAC wandeln
1150: c89f a0 00      111   ldy #0
1160: c8a1 b1 5f          lda (temp),y
1180: c8a3 c5 14          cmp int ; High Byte vergleichen
1180: c8a5 d0 07          bne 114
1190: c8a7 c8             iny
1190: c8a8 b1 5f          lda (temp),y
1200: c8aa e5 15          sbc int+1 ; Lo Byte vergleichen
1210: c8ac f0 d5          beq 112
1220: c8ae a9 01      114   lda #1 ; Flag für größer
1230: c8b0 90 02          bcc 113
1230: c8b2 a9 ff          lda #$ff ; Flag für kleiner
1240: c8b4 30 c1      113   bmi integer
1245: c8b6 10 cb          bpl 112
1250: c8b8 41 52 52 errmsg .asc "array not found"
1260: c8bb 41 59 20 4f 54 20 46 50 55 4f c4

```

Hier wieder das BASIC-Ladeprogramm:

```

100 FOR I = 51200 TO 51398
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 166, 47,165, 48,134, 95,133, 96,197, 50,208, 4
130 DATA 228, 49,240, 29,160, 0,177, 95,200,197, 69,208
140 DATA 6,165, 70,209, 95,240, 23,200,177, 95, 24,101
150 DATA 95,170,200,177, 95,101, 96,144,215,169,184,133
160 DATA 34,169,200, 76, 69,164,200, 24,177, 95,101, 95
170 DATA 133, 38,200,177, 95,101, 96,133, 39,200,177, 95
180 DATA 32,150,177,133, 95,132, 96, 36, 14, 48, 36, 16
190 DATA 9, 32, 91,188, 16, 7,165, 95,164, 96, 32,162
200 DATA 187, 24,165, 95,105, 5,133, 95,144, 2,230, 96
210 DATA 164, 96,197, 38,208,227,196, 39,208,223, 96,160
220 DATA 0,177, 95,170,200,177, 95,133, 21,134, 20, 24
230 DATA 165, 95,105, 2,133, 95,144, 2,230, 96,197, 38

```

```

240 DATA 208, 13,165, 96,197, 39,208, 7,165, 20,164, 21
250 DATA 76,145,179,160, 0,177, 95,197, 20,208, 7,200
260 DATA 177, 95,229, 21,240,213,169, 1,144, 2,169,255
270 DATA 48,193, 16,203, 65, 82, 82, 65, 89, 32, 78, 79
280 DATA 84, 32, 70, 79, 85, 78,196
290 IF S <> 22908 THEN PRINT "FEHLER IN DATAS !!": END
300 PRINT "OK !"

```

Die abgedruckte Version berechnet das Maximum eines Arrays. Wollen Sie das Minimum berechnen, müssen Sie die Verzweigungslogik nach den Vergleichen umkehren. Dazu müssen Sie aus BMI-Befehlen ein BPL machen und umgekehrt. Folgende Adressen müssen Sie ändern:

```

C858   aus $10 wird $30
C8B4   aus $30 wird $10
C8B6   aus $10 wird $30

```

Wollen Sie die Funktion benutzen, müssen Sie zuerst die Adresse für die USR-Funktion setzen:

```
POKE 785,0 : POKE 786,200
```

Danach können Sie die Funktion mit `PRINT USR(A)` aufrufen, wobei A der Name des Arrays ist. Integerarrays sind ebenfalls erlaubt. Die Funktion USR kann wie jede andere Funktion aufgerufen werden, z.B. `X = USR(A%)*SIN(3)`.

Folgendes kleine Programm kann zum Test der Funktion dienen.

```

100 POKE 785,0 : POKE 786,200
110 INPUT "ARRAYGROESSE ";N
120 DIM A(N)
130 FOR I=0 TO N

```

```
140 A(I) = RND (1)*1000
150 PRINT A(I)
160 NEXT
170 PRINT
180 PRINT USR(A)
```

Die Umschaltung von MAX auf MIN-Funktion kann durch Ändern der drei obigen Werte mittels POKE geschehen:

```
POKE 51288,48   bzw. 16
POKE 51380,16   bzw. 48
POKE 51382,48   bzw. 16
```

Probieren Sie auch diese POKES für die MIN-Funktion und benutzen Sie auch Integerarrays. Beachten Sie dabei auch den Zeitunterschied (nur bei sehr großen Arrays zu bemerken, z.B. DIM A%(5000)).

9.4 DUMP-BEFEHL ZUR VARIABLENAUSGABE

Das folgende Maschinenprogramm ist sehr nützlich zum Austesten von BASIC-Programmen. Es gibt sämtliche BASIC-Variablen mit den zugehörigen Werten aus. Das Programm liegt im Kassettenpuffer und wird mit SYS 828 aufgerufen.

```
100: 033c          *= 828      ; Kassettenpuffer
110: 033c a5 2d      lda $2d
120: 033e a4 2e      ldy $2e    ; Zeiger auf Variablenstart
130: 0340 85 14 10    sta $14    ; merken
140: 0342 84 15      sty $15
150: 0344 c4 30      cpy $30    ; mit Variablenende vergleichen
160: 0346 d0 02      bne 11
170: 0348 c5 2f      cmp $2f
180: 034a b0 18 11    bcs 13     ; zu Ende, dann fertig
190: 034c 69 02      adc #2     ; Zeiger auf Variablenwert
200: 034e 90 01      bcc 12
210: 0350 c8         iny
220: 0351 85 22 12    sta $22
230: 0353 84 23      sty $23
240: 0355 20 82 03    jsr 17     ; Name ausgeben
250: 0358 20 b6 03    jsr 112    ; "=" ausgeben
260: 035b 8a         txa
270: 035c 10 07      bpl 14
280: 035e 20 bf 03    jsr 113    ; Intgervariable ausgeben
290: 0361 4c 71 03    jmp 16     ; zur Hauptschleife
300: 0364 60 13      rts
310: 0365 98 14      tya
320: 0366 30 06      bmi 15
330: 0368 20 cf 03    jsr 114    ; Fließkommazahl ausgeben
340: 036b 4c 71 03    jmp 16
```

350:	036e 20 d8 03 15	jsr 116	; Stringvariable ausgeben
360:	0371 a9 0d 16	lda #13	; carriage return
370:	0373 20 d2 ff	jsr \$ffd2	; ausgeben
380:	0376 a5 14	lda \$14	
390:	0378 a4 15	ldy \$15	
400:	037a 18	clc	
410:	037b 69 07	adc #7	; 7 addieren für nächste Variable
420:	037d 90 c1	bcc 10	
430:	037f c8	iny	
440:	0380 b0 be	bcs 10	; zur Hauptschleife
450:	0382 a0 00 17	ldy #0	
460:	0384 b1 14	lda (\$14),y	; erster Buchstabe des Namens
470:	0386 aa	tax	
480:	0387 29 7f	and #\$7f	
490:	0389 20 d2 ff	jsr \$ffd2	; ausgeben
500:	038c c8	iny	
510:	038d b1 14	lda (\$14),y	; zweites Zeichen
520:	038f a8	tay	
530:	0390 29 7f	and #\$7f	
540:	0392 f0 03	beq 18	
550:	0394 20 d2 ff	jsr \$ffd2	; ausgeben
560:	0397 8a 18	txa	
570:	0398 10 11	bpl 110	; Typ testen
580:	039a 98	tya	
590:	039b 30 0a	bmi 19	
600:	039d a9 21	lda #"!"	; Funktion, "!" ausgeben
610:	039f 20 d2 ff	jsr \$ffd2	
620:	03a2 68	pla	
630:	03a3 68	pla	
640:	03a4 4c 71 03	jmp 16	; Rücksprung in Hauptschleife
650:	03a7 a9 25 19	lda #"%"	; Intervervariable
660:	03a9 d0 4e	bne 119	
670:	03ab 98 110	tya	
680:	03ac 10 04	bpl 111	

690:	03ae a9 24	lda #"\$"	; Stringvariable
700:	03b0 d0 47	bne 119	
710:	03b2 60	rts	
	111		
720:	03b3 20 d2 ff	jsr \$ffd2	; Zeichen ausgeben
730:	03b6 a9 20	lda #" "	
	112		
740:	03b8 20 d2 ff	jsr \$ffd2	; Leerstelle ausgeben
750:	03bb a9 3d	lda #"="	
760:	03bd d0 3a	bne 119	; ausgeben
770:	03bf a0 00	ldy #0	; Integervariable
	113		
780:	03c1 b1 22	lda (\$22),y	; Lo Byte
790:	03c3 aa	tax	
800:	03c4 c8	iny	
810:	03c5 b1 22	lda (\$22),y	; Hi Byte
820:	03c7 a8	tay	
830:	03c8 8a	txa	
840:	03c9 20 95 b3	jsr \$b395	; nach Fließkomma wandeln
850:	03cc 4c d2 03	jmp 115	; und ausgeben
860:	03cf 20 a6 bb 114	jsr \$bba6	; Fließkommavariable holen
870:	03d2 20 dd bd 115	jsr \$bddd	; in ASCII-String umwandeln
880:	03d5 4c 1e ab	jmp \$ab1e	; und ausgeben
890:	03d8 20 f7 03 116	jsr 118	; String, Hochkomma ausgeben
900:	03db a0 02	ldy #2	
910:	03dd b1 22	lda (\$22),y	; Adresse high
920:	03df 85 25	sta \$25	
930:	03e1 88	dey	
940:	03e2 b1 22	lda (\$22),y	; Adresse low
950:	03e4 85 24	sta \$24	
960:	03e6 88	dey	
970:	03e7 b1 22	lda (\$22),y	; Länge
980:	03e9 85 26	sta \$26	
990:	03eb f0 0a	beq 118	
1000:	03ed b1 24	lda (\$24),y	; Zeichen des Strings
	117		
1010:	03ef 20 d2 ff	jsr \$ffd2	; ausgeben
1020:	03f2 c8	iny	

```

1030: 03f3 c4 26          cpy $26      ; String zu Ende ?
1040: 03f5 d0 f6          bne 117
1050: 03f7 a9 22    118    lda #$22      ; Hochkomma
1060: 03f9 4c d2 ff 119    jmp $ffd2    ; ausgeben

```

Hier wieder das BASIC-Programm zur Eingabe des Maschinenprogramms.

```

100 for i = 828 to 1019
110 read x : poke i,x : s=s+x : next
120 data 165, 45,164, 46,133, 20,132, 21,196, 48,208, 2
130 data 197, 47,176, 24,105, 2,144, 1,200,133, 34,132
140 data 35, 32,130, 3, 32,182, 3,138, 16, 7, 32,191
150 data 3, 76,113, 3, 96,152, 48, 6, 32,207, 3, 76
160 data 113, 3, 32,216, 3,169, 13, 32,210,255,165, 20
170 data 164, 21, 24,105, 7,144,193,200,176,190,160, 0
180 data 177, 20,170, 41,127, 32,210,255,200,177, 20,168
190 data 41,127,240, 3, 32,210,255,138, 16, 17,152, 48
200 data 10,169, 33, 32,210,255,104,104, 76,113, 3,169
210 data 37,208, 78,152, 16, 4,169, 36,208, 71, 96, 32
220 data 210,255,169, 32, 32,210,255,169, 61,208, 58,160
230 data 0,177, 34,170,200,177, 34,168,138, 32,149,179
240 data 76,210, 3, 32,166,187, 32,221,189, 76, 30,171
250 data 32,247, 3,160, 2,177, 34,133, 37,136,177, 34
260 data 133, 36,136,177, 34,133, 38,240, 10,177, 36, 32
270 data 210,255,200,196, 38,208,246,169, 34, 76,210,255
280 if s <> 20988 then print "fehler in datas !!" : end
290 print "ok !"

```

Wenn Sie das folgende kleine Programm laufen lassen, erhalten Sie folgenden Ausdruck:

```
100 a=5
110 def fnx (y) = sin(y) * cos(y)
120 c$ = "programm"
130 b% = -101
140 sys 828
```

```
a = 5
x!
y = 0
c$="programm"
b%=-101
```

Sie können die DUMP-Funktion sowohl vom Programm aus als auch im Direktmodus mit SYS 828 aufrufen. Haben Sie ein Programm unterbrochen, können Sie sich die aktuellen Variableninhalte mit DUMP ansehen und danach mit CONT im Programm fortfahren. Wie Sie im obigen Beispiel gesehen haben, werden definierte Benutzerfunktionen durch ein "!" gekennzeichnet.

9.5 MODIFIZIERTE PEEK-FUNKTION

Das folgende kleine Maschinenprogramm ermöglicht es Ihnen auf elegante Weise, den zusätzlichen RAM-Speicher des Commodore 64 zu nutzen. Gleichzeitig ist damit auch ein Auslesen des Charactergenerators von BASIC aus möglich. Dazu noch ein paar Erläuterungen:

Die Speicherbereiche \$A000 bis \$BFFF (40960 bis 49151) und \$E000 bis \$FFFF (57344 bis 65535) sind jeweils doppelt belegt: Einmal durch je 8K BASIC- bzw. Betriebssystem-ROM sowie zusätzlich noch durch zweimal 8 K RAM. Diese 16 KBytes lassen sich von BASIC jedoch nicht ohne weiteres nutzen. Jeder POKE-Befehl in diesen Speicherbereich schreibt zwar unmittelbar in das RAM, ein Leseversuch mit PEEK liest jedoch immer das eingeschaltete ROM. Hier ersetzen wir nun die PEEK-Funktion durch eine selbst geschriebeneUSR-Funktion. Die Funktion muß folgende Aufgaben durchführen: Bevor der Wert einer Speicherzelle gelesen wird, muß erst die Speicherkonfiguration hergestellt werden, die das RAM anspricht. Jetzt kann der Wert gelesen werden. Anschließend muß die alte Konfiguration wieder hergestellt werden. Zusätzlich soll im Bereich von \$D000 bis \$DFFF der dort liegende Character-generator ausgelesen werden können. Es wird deshalb geprüft, ob die PEEK-Adresse zwischen \$D000 und \$DFFF liegt. Falls ja wird die Speicherkonfiguration so eingestellt, daß sich der Zeichengenerator auslesen läßt. Jetzt wird der Wert gelesen und wieder auf die alte Speicherkonfiguration umgeschaltet.

```
                ; USR - PEEK
110: 033C      ADR      =   $14      ; Integeradresse
120: 033C      FACADR   =   $B7F7
130: 033C      YFAC     =   $B3A2
```

190:	033C		*= 828 ; Kassettenpuffer
200:	033C A5 14		LDA ADR
210:	033E 48		PHA ; Integeradresse retten
220:	033F A5 15		LDA ADR+1
230:	0341 48		PHA
240:	0342 20 F7 B7		JSR FACADR ; FAC in Adressformat wandeln
250:	0345 A5 01		LDA 1
260:	0347 48		PHA ; Konfiguration merken
261:	0348 A5 15		LDA ADR+1
262:	034A C9 D0		CMP #\$D0 ; kleiner \$D000 ?
262:	034C 90 07		BCC RAM
263:	034E C9 E0		CMP #\$E0 ; größer \$E000 ?
263:	0350 B0 03		BCS RAM
270:	0352 A9 31		LDA #\$31 ; Charactergenerator auslesen
270:	0354 2C		.BYT \$2C
275:	0355 A9 34	RAM	LDA #\$34 ; RAM auslesen
280:	0357 78		SEI
280:	0358 85 01		STA 1 ; Speicherkonfiguration setzen
290:	035A A0 00		LDY #0
300:	035C B1 14		LDA (ADR),Y ; Byte lesen
310:	035E A8		TAY ; und nach Y
320:	035F 68		PLA
330:	0360 85 01		STA 1 ; Konfiguration zurückholen
330:	0362 58		CLI
340:	0363 68		PLA
350:	0364 85 15		STA ADR+1
360:	0366 68		PLA ; Adresse zurückholen
370:	0367 85 14		STA ADR
380:	0369 4C A2 B3		JMP YFAC ; Y nach FAC wandeln

Das kleine Programm hat im Kassettenpuffer ab Adresse 828 Platz. Haben Sie das Programm eingegeben, müssen Sie zuerst den USR-Vektor auf 828 legen. Dies geschieht mit zwei

POKE-Befehlen.

POKE 785, 828 AND 255

POKE 786, 828 / 256

Für die 'Nicht-Maschinenprogrammierer' hier wieder ein Ladeprogramm in BASIC, das den USR-Vektor direkt initialisiert.

```
100 FOR I = 828 TO 875
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 165, 20, 72,165, 21, 72, 32,247,183,165, 1, 72
130 DATA 165, 21,201,208,144, 7,201,224,176, 3,169, 49
140 DATA 44,169, 52,120,133, 1,160, 0,177, 20,168,104
150 DATA 133, 1, 88,104,133, 21,104,133, 20, 76,162,179
160 IF S <> 5085 THEN PRINT "FEHLER IN DATAS !!" : END
170 POKE 785, 828 AND 255 : POKE 786, 828/256
180 PRINT "OK !"
```

Wollen Sie nun das RAM oder den Zeichengenerator auslesen, so ersetzen Sie einfach die PEEK-Funktion durch die USR-Funktion. Wollen Sie z.B. die Zeichenmatrix eines Zeichens, so können Sie folgendes kleine Programm benutzen:

```
100 CG=13*4096
110 A = (PEEK(53248+24) AND 2) * 1024
120 INPUT "BILDSCHIRMKODE DES ZEICHENS ";B
120 FOR I=0 TO 7
130 PRINT I, USR(CG+A+8*B+I) : NEXT
140 GOTO 110
```

Zeile 110 wählt die untere bzw. obere Hälfte des Charactergenerators aus, je nachdem ob Groß/Grafik oder Klein/Großschreibung ausgewählt ist.

Den Bildschirmcode können Sie einfach so ermitteln: Setzen Sie das Zeichen in die linke obere Ecke des Bildschirm und fragen Sie dann mit PEEK(1024) den Code ab.

Jetzt haben Sie zusätzliche 16 K RAM, die Sie z.B. zur Datenspeicherung von BASIC aus nutzen können.

9.6 MULTI-TASKING AUF DEM COMMODORE 64

Multi-Tasking ist ein Begriff, der ursprünglich aus der Groß-EDV kommt und das gleichzeitige Verarbeiten mehrerer Programme auf einem Rechner bedeutet. Wie funktioniert so etwas ?

Auch ein Großrechner kann genau genommen immer nur an einem Programm arbeiten, jedoch arbeitet man hier mit einem Trick:

Soll der Rechner z.B. 5 Programme gleichzeitig betreiben, so geht man folgendermaßen vor: Der Rechner beginnt zuerst mit der Bearbeitung des ersten Programms, nach einer bestimmten Zeit (Bruchteile einer Sekunde) unterbricht er die Arbeit daran, speichert alle aktuellen Variablen und beginnt mit der Bearbeitung des nächsten Programms. Auch dieses Programm wird nach einer kurzen Zeit wieder unterbrochen, um mit dem nächsten fort zu fahren. Sind alle Programme einmal dran gewesen, werden die Variablen des ersten Programms zurückgeholt und dieses Programm fortgeführt. Die Rechenzeit der Rechners wird also in kleine 'Zeitscheiben' auf die einzelnen Programme aufgeteilt, daher auch der öfter verwendete Ausdruck 'Timesharing'.

In beschränktem Umfang ist eine solche Vorgehensweise auch auf Ihrem Commodore 64 möglich. In Ihrem Rechner laufen in der Tat zwei Programme gleichzeitig ab: Einmal Ihr BASIC-Programm und zum zweiten das sogenannte Interrupt- oder Unterbrechungsprogramm, das 60mal in der Sekunde durch einen Zeitgeber gesteuert aufgerufen und ausgeführt wird. Während der Ausführung der Ausführung des Unterbrechungsprogramms ist Ihr BASIC-Programm unterbrochen. Das Unterbrechungsprogramm sorgt zum Beispiel dafür, daß die Tastatur regelmäßig abge-

fragt wird und daß Sie überhaupt etwas über die Tastatur eingeben können.

Hier können wir uns nun 'reinhängen' und während des Interrupts den Rechner noch zusätzliche Aufgaben erledigen lassen.

Eine Anwendung dazu wäre z.B. das Ausgeben eines Textes an den Drucker. Bei jedem Interrupt kann z.B. ein Zeichen aus einem Textpuffer an den Drucker geschickt werden. Der Anwender würde dabei in seinem BASIC-Programm nicht merklich gestört; das Programm liefе wie gewöhnlich ab.

Als Beispiel für eine solche Anwendung haben wir einen kleinen Leckerbissen für Sie: Ein Programm, das Ihnen ständig die aktuelle Uhrzeit auf die Zehntelsekunde genau auf dem Bildschirm anzeigt. Das Programm benutzt die eingebaute Hardwareuhr des Commodore 64. Die aktuelle Uhrzeit wird einmal eingegeben und dann automatisch immer in der rechten oberen Ecke des Bildschirms angezeigt. Das Programm ist in Maschinensprache geschrieben, kann jedoch auch mit einem Ladeprogramm in BASIC eingegeben werden.

```
100:  C800                .OPT P1
                        ; TIME, ZEIT ANZEIGEN
                        ; SYS AD,"HHMMSS",COLOR
                        ;
140:  C800      FRMEVL  =   $AD9E   ; BASICAUSDRUCK HOLEN
150:  C800      FRESTR  =   $B6A3
160:  C800      CHKCOM  =   $AEFD   ; AUF KOMMA PRÜFEN
170:  C800      CHRGOT  =   $79
180:  C800      GETBYT  =   $B79E   ; BYTAUSDRUCK HOLEN
```

190:	C800	ILLQUAN =	\$B248 ; 'ILLEGAL QUANTITY'
200:	C800	ADR =	\$22
210:	C800	COLOR =	\$2A7 ; SPEICHER FÜR FARBWERT
220:	C800	VIDEO =	\$288 ; HI-BYTE VIDEO-RAM
230:	C800	TEMP =	\$FB
240:	C800	IRQ =	\$314 ; IRQ-VEKTOR
250:	C800	PNT =	\$FB
260:	C800	IRQVEC =	\$EA31 ; NORMALE IRQ-ROUTINE
270:	C800	FARBE =	\$D800 ; FARBRAM
280:	C800	ZEHNTEL =	\$DC08 ; ECHTZEITUHR CIA 1
290:	C800	SEKUNDE =	ZEHNTEL +1
300:	C800	MINUTE =	SEKUNDE +1
310:	C800	STUNDE =	MINUTE+1
320:	C800	TRIGGER =	STUNDE +3 ; 50/60 HZ
330:	C800	SET =	TRIGGER +1 ; ZEIT/ALARM SETZEN
340:	C800	*=	\$C800
350:	C800 AD OE DC	LDA	TRIGGER
360:	C803 09 80	ORA	#\$10000000 ; 50 HZ-MODUS
370:	C805 8D OE DC	STA	TRIGGER
380:	C808 AD OF DC	LDA	SET
390:	C80B 29 7F	AND	#\$11111111 ; UHRZEIT SETZEN
400:	C80D 8D OF DC	STA	SET
410:	C810 20 79 00	JSR	CHRGOT ; WEITERER CODE ?
420:	C813 F0 65	BEQ	CHGIRQ ; UHR UMSCHALTEN
430:	C815 20 FD AE	JSR	CHKCOM
440:	C818 20 9E AD	JSR	FRMEVL ; STRING HOLEN
450:	C81B 20 A3 B6	JSR	FRESTR ; PARAMETER
460:	C81E C9 06	CMP	#6 ; 6 ZEICHEN ?
470:	C820 D0 6B	BNE	ILL ; ILLEGAL QUANTITY
480:	C822 A0 00	LDY	#0
490:	C824 B1 22	LDA	(ADR),Y ; STD-ZEHNER
500:	C826 38	SEC	
510:	C827 E9 30	SBC	#"0" ; NACH HEX
520:	C829 C9 03	CMP	#3

530:	C82B B0 60	BCS	ILL
540:	C82D 0A	ASL	
540:	C82E 0A	ASL	
540:	C82F 0A	ASL	
540:	C830 0A	ASL	
550:	C831 85 FB	STA	TEMP
560:	C833 C8	INY	
570:	C834 B1 22	LDA	(ADR),Y ; STD-EINER
580:	C836 38	SEC	
590:	C837 E9 30	SBC	#"0"
600:	C839 C9 0A	CMP	#10
610:	C83B B0 50	BCS	ILL
620:	C83D 05 FB	ORA	TEMP
630:	C83F D0 04	BNE	NOTNULL
640:	C841 A9 92	LDA	#\$10000000 + \$12 ; 12 UHR PM
650:	C843 D0 0F	BNE	SETSTD
660:	C845 C9 24	NOTNULL	CMP
670:	C847 B0 44	BCS	ILL
680:	C849 C9 13	CMP	#\$13
690:	C84B 90 07	BCC	SETSTD
700:	C84D 38	SEC	
700:	C84E F8	SED	
710:	C84F E9 12	SBC	#\$12
710:	C851 D8	CLD	
720:	C852 09 80	ORA	#\$10000000 ; PM SETZEN
730:	C854 8D 0B DC	SETSTD	STA
740:	C857 20 FD C8	JSR	GET59 ; MINUTEN HOLEN
750:	C85A 8D 0A DC	STA	MINUTE
760:	C85D 20 FD C8	JSR	GET59
770:	C860 8D 09 DC	STA	SEKUNDE
780:	C863 A9 00	LDA	#0
790:	C865 8D 08 DC	STA	ZEHNTEL ; UHR STARTEN
800:	C868 20 79 00	JSR	CHRGOT
810:	C86B F0 0D	BEQ	CHGIRQ

820:	C86D 20 FD AE	JSR	CHKCOM	
830:	C870 20 9E B7	JSR	GETBYT	; FARBE
840:	C873 E0 10	CPX	#16	
850:	C875 B0 16	BCS	ILL	
860:	C877 8E A7 02	STX	COLOR	; FARBKODE MERKEN
870:	C87A 78	CHGIRQ	SEI	; IRQ-VEKTOR VERTAUSCHEN
870:	C87B AD 14 03	LDA	IRQ	
880:	C87E 49 A1	EOR	#< IRQVEC	↑ TIMIRQ
890:	C880 8D 14 03	STA	IRQ	
900:	C883 AD 15 03	LDA	IRQ + 1	
910:	C886 49 22	EOR	#> IRQVEC	↑ TIMIRQ
920:	C888 8D 15 03	STA	IRQ + 1	
930:	C88B 58	CLI		
940:	C88C 60	RTS		
950:	C88D 4C 48 B2 ILL	JMP	ILLQUAN	
				; ANZEIGE-ROUTINE
970:	C890 A5 FB	TIMIRQ	LDA	PNT
970:	C892 48		PHA	
970:	C893 A5 FC		LDA	PNT+1 ; ZEIGER RETTEN
970:	C895 48		PHA	
970:	C896 AD 88 02		LDA	VIDEO ; HIGHBYT VIDEO-RAM
980:	C899 85 FC		STA	PNT+1
990:	C89B A9 00		LDA	#0
1000:	C89D 85 FB		STA	PNT ; ZEIGER AUF VIDEORAM
1010:	C89F A0 1E		LDY	#30 ; 30. SPALTE
1020:	C8A1 AD 0B DC		LDA	STUNDE
1020:	C8A4 C9 12		CMP	#\$12
1020:	C8A6 F0 11		BEQ	NULLUHR
1020:	C8A8 C9 80		CMP	#\$80
1020:	C8AA 90 0F		BCC	STDOUT ; AM
1030:	C8AC 29 7F		AND	#\$1111111
1030:	C8AE C9 12		CMP	#\$12
1030:	C8B0 F0 09		BEQ	STDOUT
1040:	C8B2 F8		SED	

1040:	C8B3 18	CLC	
1040:	C8B4 69 12	ADC	#\$12
1040:	C8B6 D8	CLD	
1040:	C8B7 D0 02	BNE	STDOUT
1050:	C8B9 A9 00	NULLUHR LDA	#0
1060:	C8BB 20 DB C8	STDOUT JSR	PRINT ; STUNDE ANZEIGEN
1070:	C8BE AD 0A DC	LDA	MINUTE
1080:	C8C1 20 DB C8	JSR	PRINT ; MINUTEN ANZEIGEN
1090:	C8C4 AD 09 DC	LDA	SEKUNDE
1100:	C8C7 20 DB C8	JSR	PRINT ; SEKUNDEN ANZEIGEN
1110:	C8CA AD 08 DC	LDA	ZEHNTEL
1120:	C8CD 09 30	ORA	"0"
1130:	C8CF 20 F3 C8	JSR	PRINT1 ; ZEHNTEL ANZEIGEN
1140:	C8D2 68	PLA	
1140:	C8D3 85 FC	STA	PNT+1
1140:	C8D5 68	PLA	; ZEIGER ZURÜCKHOLEN
1140:	C8D6 85 FB	STA	PNT
1140:	C8D8 4C 31 EA	JMP	IRQVEC ; ZUM ALTEN IRQ
1150:	C8DB 48	PRINT PHA	; ANZEIGE
1160:	C8DC 29 F0	AND	111110000
1160:	C8DE 4A	LSR	
1160:	C8DF 4A	LSR	
1160:	C8E0 4A	LSR	
1160:	C8E1 4A	LSR	
1170:	C8E2 18	CLC	
1170:	C8E3 69 30	ADC	"0"
1180:	C8E5 20 F3 C8	JSR	PRINT1
1190:	C8E8 68	PLA	
1200:	C8E9 29 0F	AND	11111
1210:	C8EB 18	CLC	
1210:	C8EC 69 30	ADC	"0"
1220:	C8EE 20 F3 C8	JSR	PRINT1
1230:	C8F1 A9 3A	LDA	": "
1240:	C8F3 91 FB	PRINT1 STA	(PNT),Y ; ZEICHEN

1250:	C8F5 AD A7 02		LDA	COLOR
1260:	C8F8 99 00 D8		STA	FARBE,Y ; UND FARBE
1270:	C8FB C8		INY	
1270:	C8FC 60		RTS	
1280:	C8FD C8	GET59	INY	
1290:	C8FE B1 22		LDA	(ADR),Y ; ZEHNER
1300:	C900 38		SEC	
1310:	C901 E9 30		SBC	"0"
1320:	C903 C9 06		CMP	#6
1330:	C905 B0 86	ILL1	BCS	ILL
1340:	C907 0A		ASL	
1340:	C908 0A		ASL	
1340:	C909 0A		ASL	
1340:	C90A 0A		ASL	
1350:	C90B 85 FB		STA	TEMP
1360:	C90D C8		INY	
1370:	C90E B1 22		LDA	(ADR),Y ; EINER
1380:	C910 38		SEC	
1390:	C911 E9 30		SBC	"0"
1400:	C913 C9 0A		CMP	#10
1410:	C915 B0 EE		BCS	ILL1
1420:	C917 05 FB		ORA	TEMP
1420:	C919 60		RTS	

Haben Sie keinen Assembler oder Monitor für Ihren 64er zur Verfügung, so finden Sie anschließend wieder ein Ladeprogramm in BASIC.

```

100 FOR I = 51200 TO 51481
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 173, 14,220, 9,128,141, 14,220,173, 15,220, 41
130 DATA 127,141, 15,220, 32,121, 0,240,101, 32,253,174

```

```

140 DATA 32,158,173, 32,163,182,201, 6,208,107,160, 0
150 DATA 177, 34, 56,233, 48,201, 3,176, 96, 10, 10, 10
160 DATA 10,133,251,200,177, 34, 56,233, 48,201, 10,176
170 DATA 80, 5,251,208, 4,169,146,208, 15,201, 36,176
180 DATA 68,201, 19,144, 7, 56,248,233, 18,216, 9,128
190 DATA 141, 11,220, 32,253,200,141, 10,220, 32,253,200
200 DATA 141, 9,220,169, 0,141, 8,220, 32,121, 0,240
210 DATA 13, 32,253,174, 32,158,183,224, 16,176, 22,142
220 DATA 167, 2,120,173, 20, 3, 73,161,141, 20, 3,173
230 DATA 21, 3, 73, 34,141, 21, 3, 88, 96, 76, 72,178
240 DATA 165,251, 72,165,252, 72,173,136, 2,133,252,169
250 DATA 0,133,251,160, 30,173, 11,220,201, 18,240, 17
260 DATA 201,128,144, 15, 41,127,201, 18,240, 9,248, 24
270 DATA 105, 18,216,208, 2,169, 0, 32,219,200,173, 10
280 DATA 220, 32,219,200,173, 9,220, 32,219,200,173, 8
290 DATA 220, 9, 48, 32,243,200,104,133,252,104,133,251
300 DATA 76, 49,234, 72, 41,240, 74, 74, 74, 74, 24,105
310 DATA 48, 32,243,200,104, 41, 15, 24,105, 48, 32,243
320 DATA 200,169, 58,145,251,173,167, 2,153, 0,216,200
330 DATA 96,200,177, 34, 56,233, 48,201, 6,176,134, 10
340 DATA 10, 10, 10,133,251,200,177, 34, 56,233, 48,201
350 DATA 10,176,238, 5,251, 96
360 IF S <> 32970 THEN PRINT "FEHLER IN DATAS !!" : END
370 PRINT "OK !"

```

Das Programm zur ständigen Anzeige der Uhrzeit wird folgendermaßen angewandt:

Wollen Sie die Uhrzeit eingeben und die Anzeige einschalten, so geben Sie folgenden Befehl ein:

```
SYS 51200, "HHMMSS",FARBE
```

Dabei bedeutet "HHMMSS" die Uhrzeit (Stunde, Minute, Sekunde) und FARBE den Farbkode der angezeigten Zeit (von 0 bis 15). Wollen Sie z.B. die Uhr auf 14 Uhr, 30 Minuten und 15 Sekunden setzen, und soll die Anzeige in gelber Farbe erscheinen, so würden Sie folgenden SYS-Befehl eingeben:

SYS 51200, "143015",7

Nun erscheint in der rechten oberen Ecke die aktuelle Uhrzeit mit Stunden, Minuten, Sekunden und Zehntelsekunden. Abschalten können Sie die Anzeige durch Eingabe von

SYS 51200

Dabei wird nur die Anzeige abgeschaltet; die Uhrzeit läuft intern weiter. Durch nochmalige Eingabe von

SYS 51200

erscheint wieder die aktuelle Zeit.

Und hier nun einige Hinweise zur Erstellung Ihrer eigenen Multitasking-Anwendungen:

Prinzipiell gibt es zwei Methoden zum Einschieben eines zweiten "Jobs":

Die erste Möglichkeit ist das Ausnutzen der System-Interruptroutine, die jede sechzigstel Sekunde aufgerufen wird. Diese Alternative wurde auch bei unserer Routine zur Anzeige der Uhrzeit benutzt. Dazu geht man folgendermaßen vor: Man ändert den Interruptvektor so, daß er auf die eigene Routine zeigt. Abgeschlossen werden muß die eigene Routine dann durch einen Sprung auf die ursprüngliche Interruptroutine, damit der Rechner weiterhin z.B. die Tastatur abfragt.

Bei der anderen Alternative löst die einzubindende Routine selbst einen Interrupt aus. Dies könnte man z.B. bei der Ausgabe zum Drucker machen. Hierbei könnten Sie z.B. die Busy-Leitung des Druckers als Interruptauslöser benutzen. Jedesmal wenn der Drucker bereit ist, ein Zeichen anzunehmen, löst er einen Interrupt aus. In der Interruptroutine wird nun ein Zeichen zum Drucker geschickt und anschließend im normalen Programm weitergemacht. Hat der Drucker das Zeichen verarbeitet, löst er wieder einen Interrupt aus, in dem dann das nächste Zeichen gesendet werden kann. Der Benutzer des Rechners merkt davon nichts.

Zur Programmierung eigener Anwendungen brauchen Sie natürlich Kenntnisse über das Betriebssystem des Commodore 64, die Sie in '64 intern' finden.

9.7 POKES UND DIE ZEROPAGE

Wie Sie sicherlich schon bemerkt haben, gibt es verschiedene Adressen, die bei der Programmierung sowohl in BASIC als auch in Maschinensprache von Nutzen sein können. Hier ein kleiner Ausschnitt von verschiedenen Adressen:

Adresse:	(mögliche) Anwendung:
-----	-----
0000-0001	Durch POKE in einer oder beider dieser Adressen, kann ein bestimmter Speicherbereich ab- und wieder eingeschaltet werden
0043-0044	Diese Adresse zeigt auf den Anfang des Benutzerspeichers, also auf den Start des BASIC-Programmes. Den Anfang können Sie mit dieser Zeile bekommen: PEEK(43)+256*PEEK(44). Auf der anderen Seite können Sie durch POKE auch den Anfang höher setzen und so den unteren Speicherbereich für die restlichen Sprites benutzen.
0045-0046	In dieser Adresse finden Sie den Start des Speicherplatzes, bei dem die numerischen Variablen beginnen. Diese Adresse liegt für gewöhnlich direkt hinter dem BASIC-Programmspeicher.
0047-0048	Hier steht die Anfangsadresse für den Array-Speicherplatz. Alle Felder (Arrays) werden in diesem Bereich abgelegt.
0049-0050	Diese Adresse deutet auf das Ende des Array-Speicherplatz.
0051-0052	Bei dieser Adresse beginnen die Textvariablen

(Strings) des BASIC-Programmes.

0055-0056 Zeiger auf das Ende des BASIC-RAMs. Durch versetzen dieser Adresse ist es möglich, einen bestimmten Speicherbereich im RAM des Commodore 64 vor Überschreiben zu schützen. So können Sie diesen geschützten Speicherbereich für Maschinenprogramme reservieren, und haben trotzdem den Bereich von \$C000 bis \$CFFF für andere Zwecke frei. Zum Beispiel: POKE 55,0: POKE 56,64. Diese POKES setzen das BASIC-RAM-Ende nach \$4000.

0115-0138 An dieser Stelle steht die CHRGET-Routine, die die Zeichen aus den einzelnen BASIC-Zeilen holt. Um eine BASIC-Erweiterung zu schreiben, muß diese Routine verändert werden.

0203 In dieser Adresse steht der Code der momentan gedrückten Taste. Ist der Wert dieser Adresse gleich 64, so bedeutet das, daß keine Taste gedrückt ist.

Wollen Sie sich intensiver mit dem 'Inneren' des Commodore 64 beschäftigen, so empfehlen wir Ihnen die Lektüre des DATA BECKERS BUCHES 64 intern. Dort erfahren Sie dann mehr über die Programmierung in Maschinensprache und den Aufbau des Commodore RAM und ROM. Experimentieren Sie doch einmal selber mit den verschiedenen Adressen des Commodore 64. Es steckt viel in Ihrem 64 - Sie müssen es nur aus ihm herausholen.

9.8 KONTROLLE VON TEXTEINGABE ÜBER DIE TASTATUR

Ein sehr oft auftretendes Problem in der Datenverarbeitung ist die Abfrage und die Kontrolle von Eingaben über die Tastatur. Aus diesem Grund wollen wir uns an dieser Stelle etwas intensiver mit diesem Problem befassen.

Die am häufigsten vorkommende Methode zur Eingabe von Information über Tastatur ist der INPUT-Befehl. Dieser Befehl ist zwar sehr einfach in der Anwendung, sein Nachteil ist jedoch die nicht ausreichende Möglichkeit der Kontrolle der Eingegebenen Zeichen. Wenn Sie diesen Befehl anwenden, sollten Sie dieses generell, auch bei Eingabe von Zahlen, mit einer Text-Eingabe (also \$) tun. Damit stellen Sie zumindest sicher, daß das Programm bei einer falschen Eingabe, also z.B. eines Textes bei einer Eingabe, die eine Zahl verlangte, nicht mit einem

?REDO FROM START

Fehler den Programmablauf stört. Wenden Sie dagegen eine Texteingabe an und wollen Sie eine Zahl eingeben, so müssen Sie selbstverständlich diesen Text zunächst mit der VAL-Funktion in eine Zahl umwandeln. Trotzdem ist diese Anwendung sicherer, als direkt eine Zahl einzulesen. Es gibt aber einige Besonderheiten, die Sie beachten müssen. So ergibt das Ergebnis der Funktion:

```
A$="123D4": PRINT VAL(A$)
```

den Wert 123. Dagegen ergibt

```
A$="123E2": PRINT VAL(A$)
```

den Wert 12300. Hier hat das Zeichen "E" die Bedeutung als EXPONENT, also 10^x . In unserem Beispiel: 123×10^2 . Wir sehen also, daß im "Normalfall" der Wert eines Textes nur bis zum nächsten Buchstaben ausgegeben wird, es sei denn, es handelt sich um ein "E". Aber auch hier gibt es eine Ausnahme. Die Funktion:

```
A$="123EE2": PRINT VAL(A$)
```

ergibt wiederum den Wert 123. Falls Sie einmal in die Verlegenheit kommen sollten, aus einem Text, der über Input eingegeben wurde eine Zahl zu bilden und Sie wollen alle Buchstaben ignorieren, so wird Ihnen folgende Routine recht nützlich sein:

(Wir gehen davon aus, daß in der Variable A\$ der Text gespeichert ist. Sie können das Programm natürlich dahingehend abändern, daß Sie auch andere Variable überprüfen können.)

```
50000 IF LEN(A$)=0 THEN RETURN: REM KEIN TEXT VORHANDEN
50005 B$="": REM TEXT LOESCHEN
50010 FOR I=1 TO LEN(A$): REM GANZEN TEXT TESTTEN
50020 TS$=MID$(A$,I,1): REM ZEICHEN AN POSITION X
50030 IF TS$<"0" OR TS$>"9" THEN 50050: REM EINE ZAHL ?
50040 B$=B$+TS$: REM NEUEN TEXT BILDEN
50050 NEXT I: REM NAECHSTES ZEICHEN
50060 A=VAL(B$): REM WERT BILDEN
50070 RETURN: REM ZURUECK INS HAUPTPROGRAMM
```

Wenn Sie aus Ihrem Hauptprogramm diese Routine mit dem übergebenen Wert A\$="123DD4" aufrufen (GOSUB 50000), so wird der Wert 1234 in der Variable A an das Hauptprogramm zurückgegeben. Die Variable A\$ bleibt allerdings unverändert. Sie müssen aber daran denken, daß nun die Exponentialangabe ohne Wirkung bleibt. Wollen Sie dieses kleine Problem beheben, so können Sie das Programm folgendermaßen abändern:

```
50025 IF TS$="E" THEN 50040
```

Nun wird auch bei dem Wert A\$="123E2" das korrekte Resultat 12300 geliefert.

Ein weiteres Problem das bei der Eingabe mit Hilfe von INPUT auftreten kann, ist die Unterscheidung von Klein- und

Großbuchstaben, sofern Sie den Commodore 64 durch `PRINT CHR$(14)` auf Groß-/Kleinschrift umgeschaltet haben. Diese Unterscheidungen sind oftmals sehr nützlich bei einer Suche von Begriffen innerhalb eines Textes oder irgend einer anderen Information, bei denen man nicht genau weiß, ob diese in Groß- oder Groß/Kleinschrift eingegeben wurden. So taucht beim Commodore 64 häufiger dieses Problem auf, da es eben die beiden Möglichkeiten Groß/Kleinschrift bzw. Groß/Grafik gibt.

Stellen Sie sich vor, Sie suchen nach innerhalb einer Adressendatei die Adresse mit dem Namen Meier. Nicht nur, daß dieser Name häufig vorkommt, es kann auch eine unterschiedliche Schreibweise vorliegen, also MEIER oder Meier. Um alle diese Namen zu finden, müssen wir wieder eine kleine Routine schreiben, die es uns ermöglicht, sowohl MEIER als auch Meier zu finden.

Wir haben in unseren ersten Routine zum Heraussuchen von Buchstaben in Zahlen schon ein ähnliches Problem. Dort haben wir den ganzen Text (neudeutsch `STRING`) auf Zahlen hin untersucht. Nun müssen wir den Text entweder auf Groß- oder Kleinbuchstaben untersuchen. Da sich beim Commodore normalerweise die kleinen Buchstaben im unteren Bereich der Zeichentabelle befinden, wollen wir auf Großbuchstaben untersuchen, da diese nach der Umschaltung auf Groß-/Graphik in Graphikzeichen verwandelt werden - und dieses sieht bei der normalen Textdarstellung nicht besonders schön aus.

Wie gerade erwähnt, befinden sich die Großbuchstaben im oberen Bereich des Zeichensatzes. Der Unterschied zwischen "a" und "A" beträgt genau 128. "a" ist codiert als `CHR$ 65` und "A" als `CHR$ 193`. Binär dargestellt ist 65 01000001 und 193 11000001. Wie Sie sehen unterscheiden sich beide Zahlen nur durch die erste "1". Wenn Sie sich ein wenig mit der booleschen Algebra auskennen werden Sie wissen wie man diese "1" in eine "0" umwandeln kann, sodaß aus der 193, also "A", eine 65 und damit "a" wird.

Aber auch wenn Sie sich nicht mit dieser Berechnungsweise

auskennen, werden Sie gleich verstehen wie diese Zahlenumwandlung vor sich geht.

0 1 0 0 0 0 0 1	65	a
1 1 0 0 0 0 0 1	193	A
0 1 1 1 1 1 1 1	127	

das Resultat soll lauten:

0 1 0 0 0 0 0 1	65	a
-----------------	----	---

Um zu diesem Resultat zu gelangen müssen Sie die Zahl 65 mit der Zahl 127 UNDIEREN (AND). Hierbei wird jede Zahl nur dann "1", wenn beide verglichenen Ziffern "1" sind. Wir sehen also, daß so die erste "1" einfach ausgeblendet wird. Hier nun das Programm, das aus A\$="Meier" A\$="meier" macht.

```

50000 IF LEN(A$)=0 THEN RETURN: REM KEIN TEXT VORHANDEN
50002 B$="": REM LOESCHEN DES ZWISCHENTEXTES
50005 FOR I=1 TO LEN(A$): REM GANZER TEXT
50010 T$=MID$(A$,I,1): REM JEDES ZEICHEN UEBERPRUEFEN
50020 T=ASC(T$): REM WELCHEN CODE HAT DAS ZEICHEN ?
50030 T=T AND 127: REM AUSBLENDEN DES ERSTEN BITS (1)
50040 T$=CHR$(T): ZEICHEN MIT NEUEM CODE BILDEN
50050 B$=B$+T$: REM NEUEN TEXT BILDEN
50060 NEXT I: REM NAECHSTES ZEICHEN
50070 RETURN: REM ZURUECK INS HAUPTPROGRAMM

```

Mit dieser kleinen Routine können Sie Ihre ganzen Eingaben leicht kontrollieren. Theoretisch wäre die selbe Umstellung auch von Groß/Kleinschrift auf Großschrift. Dazu müssen Sie dann den AND 127 durch ein OR 128 ersetzen. Dieses OR macht genau das umgekehrte was AND macht. Jedes BIT wird dann auf "1" gesetzt wenn eine Ziffer "1" ist. 128 wird dargestellt als 1000000.

9.9 - FORMATIERTES PROGRAMMLISTING -

Das nachfolgende kleine Programm wurde ursprünglich erstellt, um Ihnen das Abtippen von Programmen zu erleichtern. Alle Programme, die neu für diese 4. erweiterte Buchauflage geschrieben wurden, sind mittels dieser Routine zu Papier gebracht worden.

Sie haben mit diesem Programm ein Werkzeug zur Hand, daß es Ihnen ermöglicht, Ihre Programmlistings (von Basic Programmen) frei in der Druckbreite zu gestalten. Dazu ist es lediglich erforderlich, daß Sie, nachdem Sie diese Routine geladen haben, den Namen des auszudruckenden Programms und die gewünschte Zahl der Zeichen pro Zeile angeben. Das Listing wird dann erstellt, indem der Quelltext Byte für Byte von der Diskette geholt wird und in sein Druckformat übersetzt wird. Selbstverständlich werden auch Cursor-Steuerzeichen mit ausgedruckt. Dazu wird, wenn das Programm auf ein Anführungszeichen stößt, die Variable "M" auf "1" gesetzt. Danach werden dann alle ASC-Codes, die von der Diskette gelesen werden, auf ein Steuerformat hin überprüft. Handelt es sich um ein Steuerzeichen, so wird das entsprechende inverse Druckzeichen in das Listing eingefügt.

```
100 REM *****
110 REM FORMATIERTER AUSDRUCK
120 REM *****
130 DIMK$(75):FORI=1TO75:READK$(I):NEXTI
140 PRINTCHR$(147):INPUT "PROGRAMMNAME";
P$
150 INPUT"ZEICHEN JE ZEILE":J:IFJ=0ORJ>7
9THEN150
160 OPEN1.8.0,P$:OPEN4.4.0
170 GOSUB 480 : REM START
180 GOSUB 480 : REM FOLGE
190 IF F = 0 THEN CLOSE4:CLOSE1:END
200 GOSUB 480 : REM ZEILENNR
210 Z$=F$+" "
220 GET#1,A$:IFA$=""THENA$=CHR$(0)
230 A=ASC(A$):IFA=0THENGOSUB590:GOTO180
240 IFM=1THENGOSUB290:GOTO260
250 IFA>127ANDR<203THENZ$=Z$+K$(A-127)
260 IFA>31ANDR<128THENZ$=Z$+CHR$(A)
270 IFA=34THENM=M+1:IFM=2THENM=0
```

```

280 GOTO 220
290 IFA=5 THENZ$=Z$+" "
300 IFA=17 THENZ$=Z$+" "
310 IFA=18 THENZ$=Z$+" "
320 IFA=160 THENZ$=Z$+" "
330 IFA=145 THENZ$=Z$+" "
340 IFA=19 THENZ$=Z$+" "
350 IFA=20 THENZ$=Z$+" "
360 IFA=28 THENZ$=Z$+" "
370 IFA=29 THENZ$=Z$+" "
380 IFA=30 THENZ$=Z$+" "
390 IFA=31 THENZ$=Z$+" "
400 IFA=144 THENZ$=Z$+" "
410 IFA=146 THENZ$=Z$+" "
420 IFA=147 THENZ$=Z$+" "
430 IFA=156 THENZ$=Z$+" "
440 IFA=157 THENZ$=Z$+" "
450 IFA=158 THENZ$=Z$+" "
460 IFA=159 THENZ$=Z$+" "
470 RETURN
480 GET#1,A$: IFA$="" THEN A$=CHR$(0)
490 FL =ASC(A$)
500 GET#1,A$: IFA$="" THEN A$=CHR$(0)
510 FH =ASC(A$)
520 F= FH*256+FL : F$=STR$(F) : F$=RIGHT$(F$,LEN(F$)-1)
530 RETURN
540 DATA END, FOR, NEXT, DATA, INPUT#, INPUT, DIM, READ, LET, GOTO, RUN, IF, RESTORE, GOSUB
550 DATA RETURN, REM, STOP, ON, WAIT, LOAD, SAVE, VERIFY, DEF, POKE, PRINT#, PRINT, CONT
560 DATA LIST, CLR, CMD, SYS, OPEN, CLOSE, GET, NEW, TAB, TO, FN, SPC, THEN, NOT, STEP
570 DATA +, -, *, /, ^, AND, OR, >, =, <, SGN, INT, ABS, USR, FRE, POS, SQR, RND, LOG, EXP, COS, SIN
580 DATA TAN, ATN, PEEK, LEN, STR$, VAL, ASC, CHR$, LEFT$, RIGHT$, MID$
590 IF LEN(Z$)<J+1 THEN PRINT#4,Z$: RETURN
600 PRINT#4,LEFT$(Z$,J): Z$=RIGHT$(Z$,LEN(Z$)-J): GOTO 590

```

Erklärung zum Listing

-
- | | |
|---------|--|
| 100-120 | Programmkopf |
| 130 | Erstellen der Strings für die Basic-Schlüsselworte |
| 140-150 | Benutzereingabe der Parameter |
| 160 | Öffnen eines Kanals für das Einlesen des Programms |

	das ausgelistet werden soll. Öffnen des
	Druckerkanals
170-210	Aufbereitung der Folgeadresse und der Zeilennummer
220-260	Umwandeln des Programmbytes in ein Token oder den
	normalen Charakterwert
270	Prüfen auf Anführungszeichen
280	Schleifenende für nächstes Programmbyte
290-470	Texterzeugung für Cursor Steuerzeichen
480-530	Unterprogramm zur Berechnung einer 2 Byte Adresse
540-580	Stringwerte für Tokens
590-600	Druckausgabe einer aufbereiteten Zeile

Dieses Programm mag Ihnen als Grundgerüst für weitere Funktionen dienen, die hier noch nicht enthalten sind. So ist es vielleicht wünschenswert, außer der Anzahl von Zeichen je Zeile auch noch die Zeilenzahl je Seite mit zu berücksichtigen. Das ist relativ leicht zu realisieren, da hier nur ein Zähler in den beiden Druckausgabezeilen implementiert werden muß. Bisweilen werden auch die Zeichen für die Steuerzeichen der Cursorfarbe orange bis grau³ noch nicht erzeugt. Hierfür ist aber schon Platz in den Zeilen 460 bis 470 reserviert.

Zu Problemen kann es kommen, wenn ein String im Quelltext nicht durch ein zweites Anführungszeichen begrenzt wird. Das ist aber durch eine kleine Änderung zu erreichen. Zeile 180 muß wie folgt abgeändert werden :

```
180 GOSUB 480 : M = 0 : REM Folge
```

9.10 RETTEN VON VARIABLEN UND WARMSTART

Vielleicht kennen Sie auch unser Buch 64 intern? Dort haben wir bei der Map des Adressmanager angemerkt, daß das Programm zur Filterung der redundanten Kombinationen etliche Wochen 'am Stück' lief.

Möglicherweise haben auch Sie größere Aufgaben zu bewältigen, scheuen sich jedoch davor, das Programm unbeaufsichtigt laufen zu lassen.

Hier haben wir die Lösung für Sie:

Sie brauchen Ihr Programm nur um die hier vorgestellten Zeilen erweitern, und schon können Sie den Lauf jederzeit abbrechen und am nächsten Tage mit den aktuellen Variableninhalten fortsetzen.

Falls Sie in Ihrem Programm keinen festen Warmstartpunkt vorgesehen haben, ändern Sie die Zeile 7 in

```
7 STOP
```

Die zugehörigen Routinen sind erfreulich kurz. Die Zeilen 10 ... 140 dienen nur der Demonstration und werden für den 'Ernstfall' nicht benötigt.

Bitte glauben Sie nicht, daß die Zeilen 63050 ... 63070 redundant seien! Sie tun nur scheinbar dasselbe wie die drei Zeilen davor. Auf die Hintergründe einzugehen, würde an dieser Stelle zu weit führen.

Die Anwendung ist denkbar einfach:

Wenn Sie meinen, Ihr Programm sei lange genug gelaufen, drücken Sie einfach die STOP-Taste. Merken Sie sich die Zeilennr. der 'BREAK'-Meldung!

Tragen Sie unbedingt dafür Sorge, daß der Abbruch nicht mitten in einer Zeile erfolgt (wenn mehrere Befehle in einer Zeile stehen)!

Dann geben Sie ein: GOTO 63000

Das ist alles.

Falls Sie alle Stringvariablen als Literale im Programm selbst verankert haben wie in dem Beispiel, können Sie die Frage, ob die Strings auch gerettet werden sollen, mit 'N' beantworten. Diese Variablen sind nach dem Warmstart wieder vorhanden.

Haben Sie nicht mit 'N' geantwortet, dauert es eine Weile bis zur 'FERTIG'-Meldung, denn es wird der gesamte Bereich vom Programmende bis zur Speicherobergrenze gerettet.

Zum Warmstart laden Sie Ihr Programm und geben ein:

RUN 2

Jetzt kann es wieder eine Zeit dauern, bis die 'BREAK'-Meldung kommt (oder auch nicht, je nachdem, wie die Zeile 7 aussieht).

Das Programm selbst darf inzwischen nicht verändert worden sein, da sich sonst der Variablenbereich verschiebt!

Geben Sie jetzt (nach 'BREAK') ein:

GOTO Zeilenr.

Zeilenr. muß die sein, die als nächste derjenigen folgt, die Sie sich beim Abbruch gemerkt haben, und schon geht es weiter.

Einen Haken hat die Sache:

Beim Abbruch evtl. offene Dateien sind jetzt nicht geöffnet! Das müßten Sie nach dem 'BREAK' ggf. von Hand machen. Es empfiehlt sich in solchen Fällen, das Programm für den Warmstart speziell auszulegen, d.h. daß es nur an den Stellen

unterbrochen werden kann, wo alle Files geschlossen sind.

Der Kaltstart muß übrigens mit:

RUN 10 (oder eine andere Zeilennr.)
erfolgen.

```
1 goto 3
2 load"var1",8,1
3 poke48,159
4 poke47,z1:poke48,z2
5 poke49,z3:poke50,z4
6 poke51,z5:poke52,z6
7 goto 100
10 a=1:b=2:c=3
20 a$="aa":b$="bb":c$="cc"
30 d$(0)="dd":d$(1)="ee":d$(2)="ff"
40 fori=0to9:d(i)=i:next
50 goto63000
100 printa,b,c
110 printa$,b$,c$
120 printd$(0),d$(1),d$(2)
130 fori=0to9:printd(i),:next
140 end
63000 ca=827:f$="var1"
63010 input"auch strings j/n":re$
63020 z1=peek(47):z2=peek(48)
63030 z3=peek(49):z4=peek(50)
63040 z5=peek(51):z6=peek(52)
63050 z1=peek(47):z2=peek(48)
63060 z3=peek(49):z4=peek(50)
63070 z5=peek(51):z6=peek(52)
63080 open15,8,15,"s0:var1":close15
63090 rem filename nach kassettenpuffer
63100 fori=1tolen(f$):pokeca+i,asc(mid$(f$,i,1)):next
63110 if re$<>"n"then63150
63120 rem endadresse
63130 z5=peek(55):z6=peek(56)
63140 poke781,peek(49):poke782,peek(50):goto63160
63150 poke781,peek(55):poke782,peek(56)
63160 rem startadresse u. zeiger auf startadr.
63170 poke251,peek(45):poke252,peek(46):poke780,251
63180 rem geræteadr., zeiger auf filename u. laenge filename
63190 poke186,8:poke187,60:poke188,3:poke183,len(f$)
63200 sys65496:rem save
63210 print"ZUHFertig"
```

DAS BRINGT'S

Die professionelle Datenbank für Commodore 64 und Apple IIe/IIc



Jetzt
auch für
**APPLE
IIe und IIc**

Wer bisher neidisch auf die Leistungsmerkmale teurer Datenbanken für größere Computer geschaut hat, der kann jetzt mithalten, ohne sich einen neuen Computer kaufen zu müssen: mit SUPERBASE, der professionellen Datenbank für COMMODORE 64 und APPLE IIe/IIc. Hier einige Leistungsmerkmale dieser „Super“-Datenbank:

- maximale Datensatzlänge 1108 Zeichen, verteilt auf bis zu 4 Bildschirmseiten
 - bis zu 127 Felder pro Datensatz, wobei Textfelder bis zu 255 Zeichen lang sein können
 - insgesamt 15 Einzeldateien können zu einer SUPERBASE-Datenbank verknüpft werden
 - Speicherkapazität nur durch Diskette begrenzt
 - umfangreiche Auswertungsmöglichkeiten und komfortabler Report-Generator
 - Kalkulationsmöglichkeiten und Rechnen
 - Import- (Einlesen von externen Daten) und Export- (Ausgabe von SUPERBASE Dateien als sequentielle Datei) Funktionen ermöglichen Datenaustausch mit anderen Programmen
 - durch leistungsfähige Datenbanksprache auch als kompletter Anwendungsgenerator verwendbar
 - deutsche Bildschirmalgorithmen und umfangreiches deutsches Handbuch
- SUPERBASE ist lieferbar für COMMODORE 64 (DM 398,- unverb. Preisempfehlung) und ab ca. Mitte November für APPLE IIe/IIc (DM 498,- unverb. Preisempfehlung).

Das Expertensystem

Für COMMODORE 64, IBM PC und APPLE IIe/IIc Anwender hat die Zukunft bereits begonnen. Ab Ende November können wir Ihnen für diese Computer das Expertensystem XPER liefern. XPER ist eine Datenbank, die auf den Prinzipien der künstlichen Intelligenz beruht.

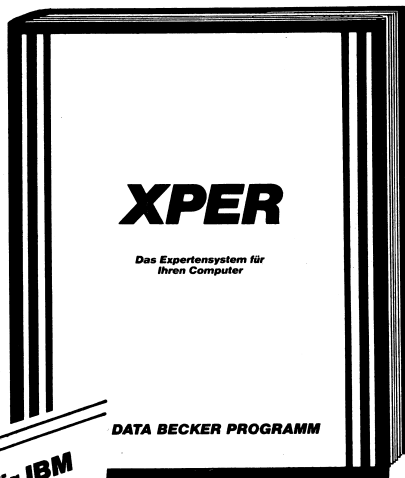
Das Prinzip eines Expertensystems wie XPER ist einfach und bestechend zugleich. Sie „füttern“ XPER mit Ihrem Wissen, z.B. als Arzt über Krankheiten und Symptome. XPER hilft Ihnen dann später bei der Diagnose, indem XPER Fragen stellt und auf der Basis Ihrer Antworten solange den Suchbaum abarbeitet, bis eine zufriedenstellende Lösung oder Eingrenzung möglicher Lösungen gefunden ist. Die Anwendungsmöglichkeiten eines Expertensystems sind praktisch unbegrenzt und bleiben der Phantasie des Benutzers überlassen. Die XPER-Versionen für die einzelnen Computer unterscheiden sich lediglich in den Datenmengen:

	Objekte	Kriterien	Eigenschaften
IBM PC	300	50	250
APPLE IIe/IIc	90	30	200
COMMODORE 64	100	40	200

Die COMMODORE 64 und APPLE-Versionen kosten jeweils DM 298,- und die IBM-Version DM 498,- (unverb. Preisempfehlung).

Jetzt
auch für
**APPLE
IIe und IIc**

für Ihren Computer



und für IBM

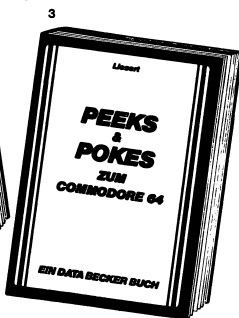
Maschinensprache

Kein Problem!



Der über 40.000mal verkaufte Bestseller, der Sie leicht verständlich und umfassend in die Maschinensprache einführt. Sie lernen Aufbau und Arbeitsweise des 6510 Mikroprozessors kennen, erfahren wie man Maschinenprogramme eingibt und startet und lernen den Umgang mit Monitor, Assembler und Disassembler. Assembler und Disassembler sind im Buch als Pro-

gramm ebenso enthalten, wie ein Einzelschrittsimulator, mit dem Sie Ihre Programme schrittweise ausführen können. Viele ausführlich beschriebene Beispielprogramme und Routinen. Mit diesem bewährten Buch sollte Ihnen der Einstieg in die Maschinensprache nicht schwerfallen. **DAS MASCHINENSPRACHE BUCH ZUM COMMODORE 64**, ca. 200 Seiten, DM 39,-



Sie haben den Einstieg in die Maschinensprache geschafft? Dann zeigt Ihnen der „neue Englisch“, wie Sie jetzt ein Profi werden. Von der Problemanalyse bis zum Maschinensprachealgorithmus werden Sie umfassend in die Grundlagen der professionellen

Maschinenspracheprogrammierung eingeführt... Dazu wieder viele Beispielprogramme, komplette Maschinenroutinen und wichtige Tips & Tricks zur Maschinenprogrammierung und zur Arbeit mit dem Betriebssystem. **DAS MASCHINENSPRACHE BUCH FÜR FORTGESCHRITTENE**, ca. 200 Seiten, DM 39,-

Mit den wichtigen PEEK und POKE Befehlen lassen sich auch von BASIC aus viele Dinge tun, für die sonst komplizierte Maschinenroutinen nötig wären. Dieses Buch erklärt leichtverständlich den Umgang mit PEEKs und POKES. Mit einer riesigen Anzahl wichtiger POKES und ihren Anwendungsmöglichkeiten. Dazu wird der Aufbau des C-64 prima erklärt: Betriebssystem, Interpreter, Zeropage, Pointer und Stacks, Charakter-Generator, Spritze-Register, Programmierung von Schnittstellen, Abschalten des Interrupts. Dazu eine erste Einführung in die Maschinensprache. Viele Beispielprogramme. **PEEKs & POKES FÜR DEN COMMODORE 64**, ca. 170 Seiten, DM 29,-



Compiler gehören zu den wichtigsten Arbeitsmitteln eines Programmierers überhaupt. In diesem Buch läßt sich ein erfolgreicher Compiler-Autor in die Karten gucken. Er erklärt nicht nur Grundlagen, Funktionsweise und richtigen Einsatz von Compilern, sondern zeigt am Beispiel einer eigenen Sprache auch, wie ein Compiler entwickelt wird. Dabei werden lexikalische Analyse, syntaktische Analyse, semantische Analyse und Codegenerierung ausführlich beschrieben. Viele nützliche Programme runden dieses auf den COMMODORE 64 zugeschnittene Buch ab, das zur Pflichtlektüre jedes ernsthaften Programmierers gehören sollte. **COMPIILER VERSTEHEN - ANWENDEN - ENTWICKELN**, ca. 200 Seiten, DM 49,-

Erscheinungstermin: Dezember '84

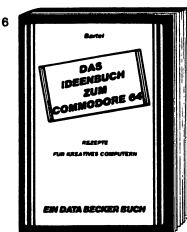
Eine umfassende, praxisorientierte Einführung in das wichtige Thema Dateiverwaltung und Datenbanken speziell für COMMODORE 64 Anwender. Dateiverwaltung, Datenbanken, Datenbanksprachen und Expertensysteme werden ebenso erklärt wie logische und physische Datenstrukturen sowie sequentieller und Direktzugriff. Wer wissen will, wie man mit Datensätzen größer als 255 Zeichen arbeitet, wie eine Hashing-Datei aufgebaut ist, wie der Direkt-

mehr wissen = besser programmieren



Erscheinungstermin: Dezember '84

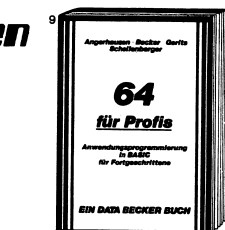
zugriff funktioniert oder wie man ein komplettes Dateiverwaltungsprogramm erstellt (im Buch als ausführlich beschriebenes Listing enthalten), der braucht dieses neue Superbuch. **ALLES ÜBER DATEI-VERWALTUNG UND DATENBANKEN**, ca. 200 Seiten, DM 39,-



Fast alles, was man mit dem 64er machen kann. Leichtverständlich und spannend geschrieben. Mit Programmen zum Nutzen und Lernen:

Gedichte vom Computer, Einladung zur Party, Werbefläche, Autokostenberechnung, Rezeptkarte, Gesundheitsarchiv, Handarbeitshilfen und noch mehr. Für jeden 64er Anwender unbedingt empfehlenswert **DAS IDEENBUCH ZUM COMMODORE 64**, 1984, über 200 Seiten, DM 29,-

Bedienerfreundlich und erfolgreich in BASIC programmieren ist keine Sache nur für Profis. Wie man's macht verraten die Software-Autoren aus dem Hause DATA BECKER: Menuesteuerung, Maskenaufbau,



Parameterisierung und Dokumentation sind die Stichworte. Dazu die leistungsfähige Datenverwaltung QUISAM mit lauffertigen Beispielprogrammen. **64 FÜR PROFIS**, 2. Auflage 1984, ca. 300 Seiten, DM 49,-

Graphik und

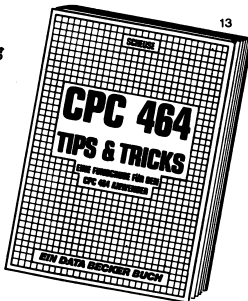


Der Bestseller zur Grafikprogrammierung des C64 vom Autor der berühmten Supergrafik. Für Einsteiger, Fortgeschrittene und Profis. Bringt alles über Sprites, High-Res-



Graphik und Multicolor bis hin zu 3-D und CAD. Unzählige Superprogramme und Routinen zum Abtippen. **DAS GRAFIKBUCH ZUM COMMODORE 64**, 1984, 295 S., DM 39,-

Jede Menge Tips & Tricks

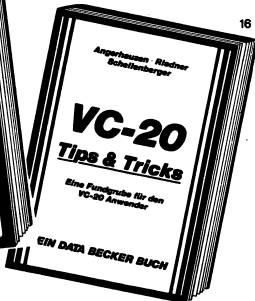


11 DER BESTSELLER – BAND 1

64 Tips & Tricks, das mit über 70.000 Exemplaren meistverkaufte DATA BECKER BUCH, ist eine hochinteressante Sammlung von Anregungen zur fortgeschrittenen Programmierung des COMMODORE 64, POKE's und andere nützliche Routinen, interessanten Programmen. Aus dem Inhalt: 3D-Graphik in BASIC – Farbige Balkengraphik – Definition eines eigenen Zeichensatzes – Tastaturbelegung – Simulation der Maus mit einem Joystick – BASIC für Fortgeschrittene – C-64 spricht deutsch – CP/M auf dem COMMODORE 64 – Druckeranschluß über den USER-Port – Datenübertragung von und zu anderen Rechnern – Synthesizer in Stereo – Retten einer nicht ordnungsgemäß geschlossenen Datei – Erzeugen einer BASIC-Zelle in BASIC – Kassettenpuffer als Datenspeicher – Multitasking auf dem COMMODORE 64-POKE's und die Zeropage – GOTO, GOSUB und RESTORE mit berechneten Zellennummern, INSTR und STRING-Funktion – Repeat-Funktion für alle Tasten. Alle Maschinenprogramme mit BASIC-Ladeprogrammen. 64 Tips & Tricks ist eine echte Fundgrube für jeden COMMODORE 64 Anwender. 64 TIPS & TRICKS, 1984, über 500 Seiten, DM 49,-

12 TIPP NOCH MEHR TIPS & TRICKS – BAND 2

Auch der zweite Band von 64 Tips & Tricks dürfte sehr schnell ein Bestseller werden. Das Buch enthält eine Fülle hochkarätiger Programme, Anregungen und Routinen: ein umfangreiches Kapitel über Softwareschutz – Befehlserweiterungen und wie man sie macht – Tips & Tricks zur Programmierung von Super-Spielen – Zeiger und deren Manipulation – mehr über Interrupt-Handling mit vielen Beispielen – erweiterte Hardware-Möglichkeiten – Betriebssystem ins RAM kopieren und dort manipulieren – sowie viele weitere Programme, Befehlserweiterungen und nützliche Routinen. Wer gerne programmiert und mehr wissen will über den COMMODORE 64, der braucht dieses neue Buch. 64 TIPS & TRICKS Band 2, ca. 250 Seiten, DM 39,-



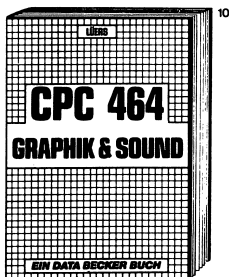
Viele Tips und Tricks rund um den CPC 464. Vom Hardwareaufbau, Betriebssystem, Basic-Tokens, Zeichnen mit dem Joystick, Anwendungen der Window-Technologie und sehr vielen interessanten Programmen wie einer umfangreichen Dateiverwaltung, Soundeditor, komfortablen Zeichengenerator bis zu kompletten Listings spannender Spiele bietet das Buch viele Anregungen und wichtige Hilfen. Diese riesige Fundgrube sollte jeder CPC 464-Besitzer haben! CPC 464 TIPS & TRICKS, 1984, über 250 Seiten, DM 39,-

Ein Buch mit Utilities, Tricks und vielen Programmen. Definitionen neuer Datenstrukturen: Liste, Stack, Schlange. Funktionsprogramme: Permutationen u. a. Spiele, Zahlenumwandlung. Grafik: CAD/CAM und 3-D-Grafik. Und der Hammer: TEXTOMAT als komfortabler Editor für PASCAL 64! PASCAL 64 TIPS & TRICKS, 1984, ca. 250 Seiten, DM 39,-

Um mehr als hundert Seiten wertvoller geworden ist das legendäre Buch zum VC 20. Neben den für jeden Benutzer wichtigen Grundlagen enthält das Buch eine Fülle erstklassiger Programmier-Tips: Programmieren der Funktionstasten, Autostart, Befehlserweiterungen und Anwenderprogramme. In diesem Wälzer wird jeder VC 20-Besitzer immer wieder etwas Neues finden! VC-20 TIPS & TRICKS, 3. erweiterte und überarbeitete Auflage, 1984, 324 Seiten, DM 49,-

ohne Abb.: Eine wahre Fundgrube für den TI-99 Anwender ist das Buch TI-99 Tips & Tricks. 300 Seiten, DM 49,-

Sound



nen zu Soundregistern, ADSR-Programmierung, Synchronisation und Ringmodulation. Zahlreiche Beispiele für Sound- und Songprogrammierung sowie wichtige Routinen runden den Inhalt ab. DAS MUSIKBUCH ZUM COMMODORE 64, über 200 Seiten, DM 39,-

Der Schneider CPC 464 besitzt nicht nur ein hervorragendes Preis-/Leistungsverhältnis, sondern auch außergewöhnliche Grafik- und Soundfähigkeiten. In diesem Buch werden die Möglichkeiten des CPC 464 umfassend und leicht verständlich dargestellt. Viele nützliche Beispielprogramme runden den Text ab. CPC 464 GRAPHIK & SOUND, ca. 200 S., DM 39,-

Erscheinungstermin: Dezember '84

Mit dem Musikbuch nutzen Sie die Klangmöglichkeiten des C64! Neben einer kurzen Einführung in die Computermusik finden Sie informatio-

Von wegen Spiel Das Superteam



DATAMAT

Deutschlands meistverkaufte Dateiverwaltung bietet einiges, was in dieser Preisklasse bisher unvorstellbar schien:

- menuegesteuertes Diskettenprogramm, dadurch extrem einfach zu bedienen – für jede Art von Daten – völlig frei gestaltbare Eingabemaske – 50 Felder pro Datensatz – 253 Zeichen pro Datensatz – bis zu 2000 Datensätze pro Datei je nach Umfang – Schnittstelle zu TEXTOMAT – läuft mit 1 oder 2 Floppies – völlig in Maschinensprache – extrem schnell – deutscher Zeichensatz auch auf COMMODORE-Druckern – fast jeder Drucker anschließbar – ausdrucken über RS 232 – duplizieren der Datendiskette – gute Benutzerführung – Hauptprogramm komplett im Speicher (kein Diskettenwechsel) – integrierte Minitextverarbeitung – deutsches Handbuch mit Übungslektionen.

Sie können:

- Jeden Datensatz in 2–3 Sekunden suchen – nach beliebigen Feldern selektieren – nach allen Feldern gleichzeitig sortieren – Listen in völlig freiem Format drucken – Etiketten drucken. Komplett nur DM 99,- (unverb. Preisempfehlung)

TEXTOMAT

Deutschlands meistverkaufte Textverarbeitung bietet Profileistung zum Hobbypreis:

TEXTOMAT in Stichworten:

- Diskettenprogramm – durchgehend menuegesteuert – deutscher Zeichensatz auch auf COMMODORE-Druckern – Rechenfunktionen für alle Grundrechenarten – 24.000 Zeichen pro Text im Speicher – beliebig lange Texte durch Verknüpfung – horizontales Scrolling für 80 Zeichen pro Zeile – läuft mit 1 oder 2 Floppies – frei programmierbare Steuerzeichen – Formularsteuerung für Randeinstellung u.s.w. – komplette Bausteinverarbeitung – Blockoperationen, Suchen und Ersetzen – Serienbriefschreibung mit DATAMAT – formatierte Ausgabe auf Bildschirm – an fast jeden Drucker anpaßbar – ausführ-

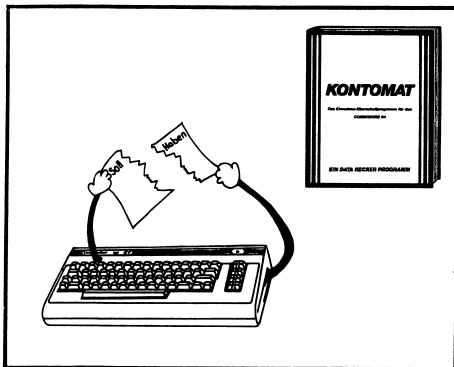
liches deutsches Handbuch mit Übungslektionen – komplett nur DM 99,- (unverb. Preisempfehlung).

Viele DATA BECKER Autoren nutzen TEXTOMAT wegen seiner Vielseitigkeit und leichten Bedienung sogar zum Erstellen ganzer Bücher.



KONTOMAT

KONTOMAT ist ein menuegesteuertes Einnahme-Erschußprogramm nach § 4(3) EStG mit Kassenbuch, Bankkontenüberwachung, automatischer Steuerbuchung, AFA Tabelleerstellung, Kontenblättern, Ermittlung der USt-Voranmeldungswerte und Monats- und Jahres



zeug...

abrechnung. Der neue KONTOMAT ist voll parametrisiert und läßt sich damit an Ihre Bedürfnisse anpassen. Für alle Gewerbetreibenden, die nicht laut HGB zur Buchführung verpflichtet sind. KONTOMAT ist für den gewerblichen Einsatz, aber auch als Lernprogramm oder zur Haushaltsbuchführung geeignet.

KONTOMAT in Stichworten:

Diskettenprogramm – maximal 120 Konten – Beträge mit bis zu 6 Vor- und 2 Nachkommastellen – 4 Mehrwert- und Vorsteuersätze – intervallmäßige Belegeingabe – 4 Buchungsarten (SOLL, HABEN, SOLL/HABEN und HABEN/SOLL) – Anzeige der Soll- und Habensumme bei mehrfachen Buchungssätzen – komfortable Belegeingabe mit Datum, Buchungstext, Steuerkennzeichen und Betrag – Druck des Journals während der Belegeingabe – Druck von umfangreichen Kontenblättern – Druck einer Summen- und Saldenliste mit Monats- und Jahresumsatzsummen – betriebswirtschaftliche Auswertung mit Druckausgabe – Ermittlung und Druckausgabe der Umsatzsteuerzahllast – Speicherung der Anlagegüter und automatische Abschreibung am Jahresende – übersichtliche AFA-Liste – arbeitet mit 1 oder 2 Laufwerken – umfangreiches deutsches Handbuch. **DM 148,-**

unverb. Preisempfehlung

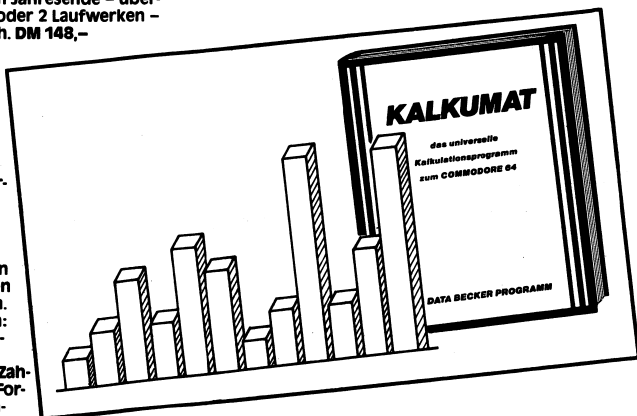
ZAHLUNGS- VERKEHR

Umfangreicher Zahlungsverkehr kann zur Plage werden. Das Software-Paket ZAHLUNGSVERKEHR übernimmt den größten Teil dieser Arbeit. Außer den notwendigen Fähigkeiten für das Ausfüllen und Auflisten von Überweisungen und Schecks ist der ZAHLUNGSVERKEHR in der Lage, Sammellisten, Einzugslisten etc. selbständig zusammenzustellen. **ZAHLUNGSVERKEHR in Stichworten:** Diskettenprogramm – max. 100 Zahlungsempfänger pro Diskette – drei definierbare Absenderbanken – 25 Zahlungsdateien – 14 frei definierbare Formulare – Kontrolldruck bei Belegeingabe möglich – Eingabe von Rechnungen-

daten oder eines Verwendungszwecks – Ausdruck einer Sammel-Überweisungsliste – Korrekturmöglichkeit der einzelnen Zahlungsdateien – arbeitet mit einer oder zwei Floppies – umfangreiches deutsches Handbuch. **DM 148,-** unverb. Preisempfehlung

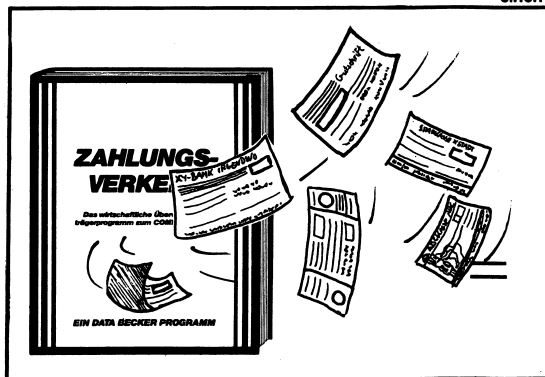
KALKUMAT

Das Software-Paket KALKUMAT setzt neue Standards für Kalkulations-Programme auf dem Commodore 64. Alle bewährten Leistungsmerkmale solcher Programme wurden übernommen, neue hinzugefügt und das Ganze mit einer ungewöhnlich komfortablen Bedienung versehen. Mit dem KALKUMAT können Sie Tabellen mit bis zu 255 Zeilen in 63 Spalten erstellen. Dabei kann jede Spaltenbreite frei gewählt werden. Werte werden über eine Edit-Zelle genauso bequem eingegeben wie in BASIC-Programmen, alle Optionen sind über Menues zu



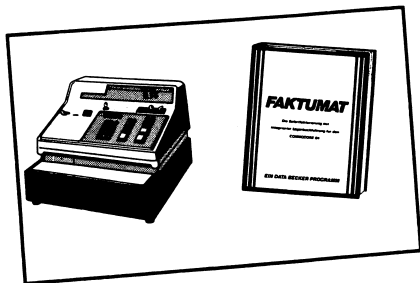
erreichen. Wenn Sie einmal unsicher sind, können Sie einen der vielen Hilfstexte aufrufen. Alle Texte werden in deutscher Sprache ausgegeben, außerdem kann zwischen dem amerikanischen und einem deutschen Zeichensatz (mit den Umlauten) gewählt werden. Einen besonderen Leckerbissen stellt das integrierte Graphik-Paket dar. Werte aus einem bearbeiteten Arbeitsblatt lassen sich auf vielfältige Weise graphisch darstellen: Kuchengraphik, Kurvenzüge, Minimum-Maximum-Graphik oder Säulendiagramme. Die so entstandenen Graphiken können Sie mit verschiedenen Schriftgrößen beschriften. Im Graphik-Programm können überdies 8 Fenster definiert werden. Der KALKUMAT ist ein Problemlöser für Aufgaben sowohl im geschäftlichen als auch im privaten Bereich. Jeder 64er-Besitzer wird sein individuelles Einsatzgebiet für den KALKUMAT finden.

KALKUMAT, auf Diskette und mit ausführlichem Handbuch DM 198,-
unverb. Preisempfehlung



Alle Programme auf Diskette

Programme aus gutem Hause.



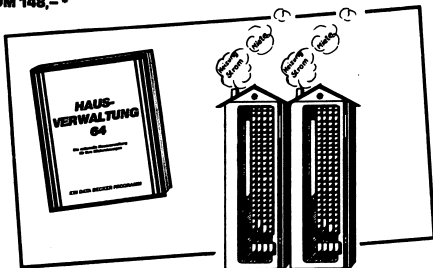
FAKTUMAT

Mit FAKTUMAT ist das Schreiben von Rechnungen kein Alptraum mehr. Eine Sofortfakturierung mit integrierter Lagerbuchführung. Individuelle Anpassung von Steuersätzen, Maßeinheiten und Firmendaten. Kunden- und Artikelstamm voll pfiegrbar. Schneller Zugriff auf Kunden- und Artikelnamen, über definierbaren, 6-stelligen Schlüssel. Automatische Fortschreibung von Artikel- und Kundendaten, individuell nutzbar. Alles in allem die Arbeits- und Zeitersparnis, die Sie sich schon immer gewünscht haben.

FAKTUMAT in Stichworten:

voll menuegesteuert – läuft mit einer oder zwei Floppies – Diskettenwechsel (eine Floppy) nur beim Wechsel vom Hauptmenue ins Unterprogramm und umgekehrt – mit Ausnahme des Ausschaltens der Floppy während der Verarbeitung werden alle Fehler abgefangen (z.B. Drucker nicht eingeschaltet – arbeitet mit 1525, 1526 MPS 801, EPSON Drucker und DATA BECKER Interface – voll parametrisiert: Firmenkopf, MWST. und Rabattsätze, Größe der Dateien beliebig wählbar – 5 Zeilen für Firmenkopf je 30 Zeichen (erste Zeile erscheint auf der Rechnung in Breitschrift – 4 Mehrwertsteuer-Sätze; während der Rechnungsschreibung können alle Artikel mit unterschiedlichem Mehrwertsteuer-Satz verrechnet werden – 10 Rabattsätze (Rabattsatz 1 vorbelegt mit 0%), bei der Rechnungsschreibung kann jedem Artikel ein Rabattsatz zugewiesen werden – maximal 1900 Artikel bei 50 Kunden oder 950 Kunden bei 100 Artikel (max. Artikel = [1000-Kunden]·2; max. Kunden = [2000-Artikel]/2) – manuelle Eingabe von Artikeln und/oder Kunde während der Rechnungsschreibung – d.h. es können mehr Artikel verrechnet werden als überhaupt in die Datei passen (bei Verzicht auf Lagerbuchführung) bzw. es können Rechnungen an Kunden geschrieben werden, die nicht erfasst wurden – integrierte Lagerbuchführung mit Ausgabe einer inventurartige Rechnungsbeträge und Datum werden in der Kundendatei festgehalten – Druck von: Rechnung (mit Abbuchen aus Lager), Rechnung (ohne Abbuchen aus Lager), Ulferschein – deutsches detailliertes Handbuch mit Übungs- und Anwendungsteil – deutsche Bedienerführung innerhalb des Programms (z.B. „Artikel nicht vorhanden“ anstelle „RECORD NOT PRESENT“).

DM 148,- *



HAUSVERWALTUNG

Jetzt können alle Hausbesitzer aufatmen: das Programm HAUSVERWALTUNG bietet Ihnen eine sehr komfortable Verwaltung der Mietwohnungen mit dem COMMODORE 64. Alles, was Sie dazu brauchen, ist ein COMMODORE 64, ein Diskettenlaufwerk 1541, ein anschlussfähiger Drucker und das oben genannte Programm HAUSVERWALTUNG. Die nachfolgenden und viele weitere leistungsfähige Features ermöglichen eine äußerst rationelle Verwaltung Ihrer Mietwohnungen.

* Bei den Preisen für Programme handelt es sich um unverb. Preisempfehlungen

HAUSVERWALTUNG in Stichworten:

Diskettenprogramm – Verwaltung von 50 Einheiten pro Objekt möglich – Stammdatenverwaltung für Häuser und Mieter – Verbuchen der Miete, Nebenkosten und Caragenmieten – Mietkontoaufstellung – Haus- und Mieteraufstellung – Mähtungen – Verbuchen der anfallenden Kosten – Kostengegenüberstellung – Jahresabrechnung mit automatischem Jahresübertrag – umfangreiches deutsches Handbuch.

DM 198,- *



Das neue Software-Paket FINANZGENIE macht Ihren 64er zum Hausverwalter. Ob feste Einnahmen und Ausgaben, Kredite und Sparposten oder variable Beträge, alles können Sie verwalten. Das FINANZGENIE liefert Ihnen die aufbereiteten Daten auf den Bildschirm oder schwarz auf weiß zum Abheften. Dabei arbeitet das Programm „terminorientiert“. Alle Daten werden anhand ihrer Fälligkeit gefunden. So ist es auch selbstverständlich, daß das FINANZGENIE über verschiedene nützliche Kalenderfunktionen verfügt. Feste Termine werden überwacht und auf einem persönlichen Terminplaner ausgedruckt. Das FINANZGENIE kümmert sich natürlich auch um Ihre Bankangelegenheit. Überweisungen werden vorbereitet und Kontostände automatisch aktualisiert. Aber auch Beratung können Sie vom Programm bekommen. Ein eigener Abschnitt dreht sich um das Thema Kredite und Sparen. Sie können beispielsweise Darlehensangebote berechnen und miteinander vergleichen oder ausrechnen lassen wieviel Zinsen Ihr Sparbuch pro Jahr bringen wird. Mit dem FINANZGENIE können Sie vieles machen, nur keine Fehler; die Bedienungsicherheit wurde auf ein ungewöhnliches Niveau gehoben. Es ist praktisch unmöglich sinnlose oder ungeeignete Eingaben zu machen. Jeder Versuch führt zu einer Fehlermeldung. Ihr 64er mit dem FINANZGENIE sind das Team für Ihre privaten Finanzen und Termine!

FINANZGENIE, DM 69,- *

DIE DISKETTE ZUM BUCH

Laden, Starten – Klari! Zu folgenden Büchern gibt's jetzt die Diskette zum Buch mit allen wichtigen Programmen:

- 64 INTERN
- 64 TIPS & TRICKS
- 64 FÜR PROFS
- DRUCKER BUCH
- FLOPPY BUCH
- MASCHINENSPRACHEBUCH
- MASCHINENSPRACHE FÜR FORTGESCHRITTENE
- GRAPHIKBUCH
- VC-20 TIPS & TRICKS
- TRAININGSBUCH DATAMAT
- COMPIER VERSTEHEN ANWENDEN ENTWICKELN
- APPLE II TIPS & TRICKS
- PASCAL 64 TIPS & TRICKS
- Preis jeweils DM 39,- *

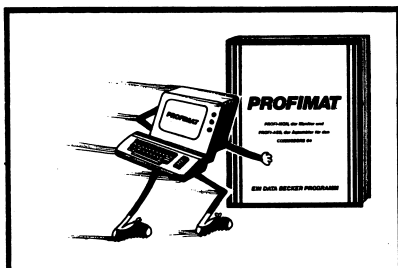


Diskette zum SCHULBUCH ZUM COMMODORE 64 mit 14 weiteren Programmen und 70-seitigem Handbuch DM 49,- *

Alle Programme auf Diskette

Der C64 spricht nicht nur Basic...

PROFIMAT

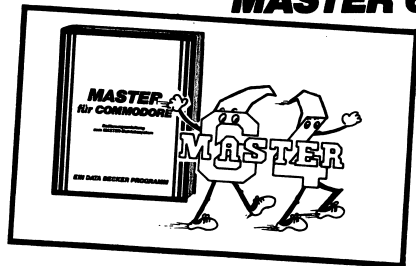


Wer sich tiefer in die inneren des Computers begeben will, kommt ohne besonderes Werkzeug nicht aus. Einerseits muß der volle Einblick in alle Speicherbereiche möglich sein, andererseits soll der Umgang mit Maschinenprogrammen so komfortabel wie möglich gestaltet sein. PROFIMAT hat Lösungen für beide Probleme: Der Maschinensprache-Monitor PROFI-MON bietet alle Hilfsmittel zum Umgang mit Maschinenprogrammen; PROFI-ASS ist ein Macro-Assembler, der das Schreiben von Maschinenprogrammen fast so einfach macht wie das Programmieren in BASIC.

PROFIMAT in Stichworten:

Registerinhalte und Flags anzeigen – Speicherinhalte anzeigen – Maschinenprogramme laden, ausführen und speichern – Speicherbereiche durchsuchen, vergleichen, füllen und verschieben – echter Einzelschrittmodus – Setzen von Unterbrechungspunkten – schneller Trace-Modus – Rückkehr zu BASIC – formatfreie Eingabe – Verkettung beliebig vieler Quellprogramme – erzeugter Objektcode kann in Speicher oder auf Diskette gehen – formatiertes Assemblerlisting – ladbare Symboltabellen – redefinierbare Symbole – Operatoren – Unterstützung der Floatkommaarithmetik – bedingte Assemblierung – Assembler-Schleifen – MACROS mit beliebigen Parametern.
DM 99,- *

MASTER 64



MASTER 64 ist ein professionelles Programmentwicklungssystem für den C-64, das es Ihnen ermöglicht, die Programmentwicklungszeit auf einen Bruchteil der sonst üblichen Zeit zu reduzieren. MASTER 64 bietet einen Programmkomfort, den Sie nutzen sollten.

MASTER 64 in Stichworten:

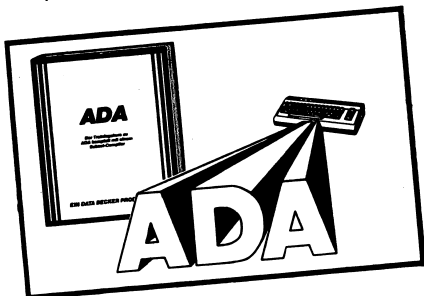
70 zusätzliche Befehle – Bildschirmmaskengenerator – definieren von Bildschirmzonen – Eingabe aus Zonen – formatierte Ausgabe – Abspeicherung von Bildschirminhalten – Arbeiten mit mehreren Bildschirmmasken – ISAM Dateiverwaltung, in der Datensätze über einen Zugriffsschlüssel angesprochen werden können – Datensätze bis zu 254 Zeichen – Schlüsselänge bis zu 30 Zeichen – Dateigröße nur von Diskettenkapazität abhängig – Zugriff über Schlüssel und Auswahlmasken – Bildschirm- und Druckmaskengenerator – Erstellung beliebiger Formulare und Ausgabemasken – BASIC-Erweiterungen – Toolkitfunktionen – Mehrfachgenaue Arithmetik (Rechnen mit 22 Stellen Genauigkeit).
DM 199,- *

ADA

Diese Programmiersprache der Zukunft, die das Pentagon in Auftrag gegeben hat, wird jetzt durch DATA BECKER auch dem C-64 Anwender zugänglich gemacht durch den TRAININGSKURS zu ADA, der eine sehr gute Einführung in diese Supersprache bietet. Der dazu gelieferte Compiler liefert ein umfangreiches Subset der Sprache.

ADA in Stichworten:

blockstrukturierte Programme – modularer Aufbau der Programme – ermöglicht die Behandlung von Ausnahmezuständen – Fehlerüberprüfung beim Übersetzen und zur Laufzeit – ermöglicht das einfache Einbinden von Maschinenprogrammen – sehr leichtes Arbeiten mit Programmibibliotheken – Programmdiskette enthält Editor, Übersetzer, Assembler und Disassembler – umfangreiches deutsches Handbuch.
DM 198,- *



STRUKTO 64

STRUKTO 64 ist eine fantastische neue Programmiersprache für strukturiertes Programmieren mit dem C-64 und für alle Programmierer geeignet, die den C-64 als Allround-Computer einsetzen und auf einfache Weise anspruchsvolle Programme erstellen wollen.

STRUKTO 64 in Stichworten:

Interpretersprache, die die Vorzüge von BASIC und PASCAL vereint – strukturiertes Programmieren – übersichtliche Programme – leichte Erlernbarkeit – einfache Bedienung – eingebautes Toolkit erleichtert das Eingeben und Verbessern von Programmen – leichteres Arbeiten mit der Floppy – Sprite-Editor ermöglicht das Einlesen der Sprite-Formen direkt vom Bildschirm – Graphikbedienung wird mit gut durchdachten Befehlen unterstützt – Abspielen von Musik ist unabhängig vom Programmablauf möglich – ca. 80 neue Befehle – lieferbar als Diskettenprogramm – ausführliches deutsches Handbuch.
DM 99,- *



Alle Programme auf Diskette

* Bei den Preisen für Programme handelt es sich um unverb. Preisempfehlungen



BASIC 64

Der Compiler BASIC 64 bietet die Möglichkeit, BASIC-Programme entweder in Maschinensprache oder in einen sogenannten Speedcode zu übersetzen. Beide Varianten sorgen dafür, daß Ihre Programme 4 bis 14mal schneller laufen. Bearbeiten Sie mit BASIC 64 alle Programme, die Ihnen schon immer zu langsam waren. Mit dem kompakten Speedcode können Sie den Speicherplatzbedarf Ihres Programms um 25% verringern, während der Speicherplatzaufwendigere Maschinencode zusätzlichen Geschwindigkeitszuwachs bringt. BASIC 64 kann jedes Programm verarbeiten, das im COMMODORE 64 BASIC geschrieben wurde (Ausnahme: einzelne POKE-Befehle) und unterstützt teilweise auch die bekannten Befehlserweiterungen. Außerdem können Sie mit BASIC 64 den Speicherplatz für Daten um 24 K erweitern. Nebenbei erledigt BASIC 64 einige Arbeiten für Sie: Umformung mathematischer Ausdrücke, möglichst ökonomische Speicherplatzausnutzung und Integer Arithmetik. Durch eine völlig veränderte Stringbehandlung schrumpft die gefürchtete Garbage Collection auf wenige Sekunden. Alle Optionen werden per Menue aufgerufen und Eingaben auf Ihre Korrektheit überprüft. Mit BASIC 64 haben Sie ein Hilfsmittel in der Hand, das Ihre BASIC-Programme schneller macht, als Sie es bisher für möglich gehalten haben. DM 99,- (unverb. Preisempfehlung)

PASCAL 64

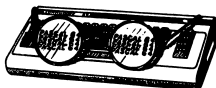
PASCAL 64 ist ein PASCAL Compiler für den COMMODORE 64 mit folgenden Eigenschaften: sehr umfangreicher Befehlsvorrat - erlaubt Interruptprogrammierung und

So macht Lernen richtig Spaß



Alle Programme auf Diskette

bietet Schnittstellen zu Monitor und Assembler - erzeugt sehr schnelle Programme in reinem Maschinencode - unterstützt relative Dateiverwaltung, Graphik und Sound - bietet die Datentypen REAL, INTEGER, CHAR und BOOLEAN sowie Aufzähltypen und Pointer, die zu Datenstrukturen RECORD, SET, ARRAY und PACKED ARRAY kombiniert werden können - erlaubt vorzeitigen Abschluß von Prozeduren mit EXIT, uneingeschränkte Rekursionen und komfortable Verarbeitung von Teilfeldern (Strings) - ist ein ausgereiftes, deutsches Produkt und wird mit ausführlichem Handbuch geliefert. DM 99,- (unverbind. Preisempfehlung)



DISKOMAT



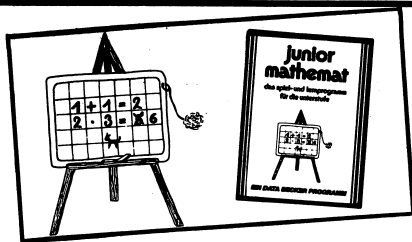
Der Umgang mit Diskettenlaufwerken ist für viele noch immer mit Geheimnissen belastet. Andere stören sich an den wenig komfortablen Diskettenbefehlen des BASIC V2. DISKOMAT bringt Abhilfe: alle Diskettenbefehle des BASIC 4.0 stehen zur Verfügung. Außerdem können mit dem Programm SUPERTWIN zwei 1541-Laufwerke wie ein Doppellaufwerk verwaltet werden. Für Benutzer, die sich die Fähigkeiten der Floppy 1541 ganz erschließen wollen, steht der DISK-MONITOR bereit: er macht es endlich möglich, den direkten Zugriff auf einzelne Blocks einfach und bequem vorzunehmen.

DISKOMAT in Stichworten:

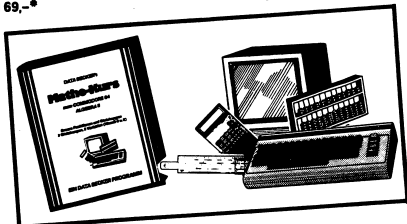
Diskettenprogramm - DISK BASIC unterstützt Diskettenbefehle des BASIC 4.0 (CONCAT, HEADER, APPEND, RENAME, OPEN, COLLECT, DSAVE, SCRATCH, DCLOSE, BACKUP, LOAD, DIRECTORY, RECORD, COPY, CATALOG, DS & DSS) - SUPER TWIN behandelt 2 Laufwerke 1541 wie ein Doppellaufwerk - DISK-MONITOR ermöglicht direkte Analyse und Manipulation von Disketten (direktes Lesen und Schreiben einzelner Blöcke, ändern von Blöcken mittels Bildschirm-Editor, Anzeige des Diskettenstatus, direktes Absenden von Disketten-Befehlen) - ausführliches deutsches Handbuch beschreibt jeden einzelnen der 3 Programmteile. DM 99,- unverb. Preisempfehlung

Brush up your English!

Können Sie sich einen geduldigeren Lehrer vorstellen als Ihren Computer? Wohl kaum. Das neue Lernprogramm **BRUSH UP YOUR ENGLISH** hat immer Zeit für Sie, hilft Ihnen und weiß, was Sie schon gelernt haben. Voraussetzung sind Englischkenntnisse gleich weicher Qualität. Das Programm stellt Ihnen verschiedene Fragen, die Sie beantworten können. Dabei schöpft **BRUSH UP...** aus einem Wortschatz von 1500 Vokabeln, mit dem über 700 Aufgaben formuliert werden können. Wenn Sie einmal nicht weiter wissen, wählen Sie die **HELP**-Funktion und Sie bekommen einen Hinweis auf die Lösung. Reicht das nicht aus, erscheint die Antwort. Außerdem besteht die Möglichkeit, eine Frage zu überspringen oder ganz aus der aktuellen Lektion zu streichen. Das alles findet in einem simulierten Kurs statt. Sie bestimmen den Schwierigkeitsgrad und das Tempo, der Computer bewertet Ihre Leistung, überwacht Ihr Tagespensum und macht Sie auf Lücken aufmerksam. Mit dem neuen Lernprogramm **BRUSH UP YOUR ENGLISH** können Sie Ihre verschollenen Sprachkenntnisse wieder richtig aufbrushen! Der komplette Kurs besteht aus drei Teilen, die sich durch die verschiedenen Wortfelder unterscheiden. Jede Diskette kann für sich allein benutzt werden. **BRUSH UP YOUR ENGLISH**, Teil 1, 2 und 3 jeweils DM 49,- unverb. Preisempfehlung

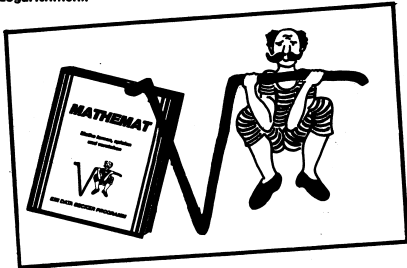


Spielerisch lernen ist beim neuen Lernprogramm JUNIOR-MATHEMATIK kein Schlagwort. Ohne verblissenen Ernst oder überzogene Anforderungen können Schüler der Unterstufe (bis zur 4. Klasse) Rechnen üben. Eingebettet sind die Aufgaben in ein kleines Videospiel, das als Belohnung und Motivation fungiert. Die Übungen sind bei jedem Schwierigkeitsgrad den offiziellen Lehrplänen angepaßt. Auch die Notengebung richtet sich nach diesem Standard. Ihr Kind kann mit dem JUNIOR-MATHEMATIK ganz selbstständig lernen und spielen. Sie haben jedoch die Möglichkeit, den Leistungsstand zu überprüfen. Alle Aufgaben werden anschaulich dargestellt und können in mehreren Anläufen gelöst werden. Klappt es beim ersten Mal nicht, gibt das Programm behutsame Hilfen und bewertet – je nach Aufgabentyp – eine Antwort erst nach dem dritten Fehlversuch als Minuspunkt. Diese Methode hilft Kindern, auch Mißerfolge zu bewältigen. Zudem sind die einzelnen Trainingseinheiten so begrenzt, daß das Kind auch wirklich in der vorgesehenen Zeit konzentriert bleiben kann. Versuchen Sie auch einmal Aufgaben aus der Mengenlehre zu lösen! Vielleicht müssen Sie dann Nachhilfe nehmen. Der JUNIOR-MATHEMATIK ist das richtige Programm für Eltern von Grundschulkindern. Mit dem leicht verständlichen Handbuch können Erwachsene und Kinder sofort anfangen. JUNIOR-MATHEMATIK, DM 69,-*



Eine neue Serie leicht verständlicher Lernprogramme, entwickelt von erfahrenen Pädagogen und Programmierern, zu den zentralen Themen der Mathematik weiterführender Schulen. Jeder Baustein besitzt einen Einführungsteil, der den Themenbereich verdeutlicht und die benötigten Begriffe, Fachausdrücke und Symbole erklärt. Die Graphikmöglichkeiten des C-64 werden voll ausgeschöpft, um auf anschauliche Weise Verständnis für mathematische Zusammenhänge zu wecken.

Regeln und Gesetze werden schrittweise entwickelt. Bei Bedarf erhält man zusätzliche Erläuterungen durch die Fragetaste. Übungsaufgaben liefert der Computer in unbegrenzter Anzahl. Jede Aufgabe wird auf Wunsch detailliert vorgerechnet. Man kann auch Tests absolvieren und vom Computer benoten lassen. Der Programmteil „Hilfe bei Hausaufgaben“ ermöglicht dem Benutzer seinerseits Aufgaben zu stellen, die der Computer löst. Jeder Kursteil auf Diskette für C-64 nur DM 49,-. Jetzt lieferbar: ALGEBRA I (Einführung in die Gleichungslehre), ALGEBRA II (Lineare Gleichungssysteme von zwei Gleichungen mit zwei Variablen), ALGEBRA III (Quadratische Gleichungen), ALGEBRA IV (Potenzen, Exponentialfunktionen, Logarithmen).



Ob Schule, Studium oder Beruf: endlich können Sie Logarithmentafeln und Formelsammlungen in der Schublade lassen. Dieses Hilfsmittel und vieles mehr ersetzt das Software-Paket MATHEMATIK. Bei einfachen mathematischen Berechnungen wird Ihr Computer mit diesem Programm zum Taschenrechner mit allen Grundrechenarten und Potenzen, Wurzelziehen, Prozentrechnen und Logarithmen. Die arithmetischen Unzulänglichkeiten des C-64 sind ausgeglichen und die Rechengenauigkeit beträgt 10 Nachkommastellen extern. Fast alle algebraischen Aufgaben können mit dem MATHEMATIK berechnet werden. Auch in der Differential- und Integralrechnung ist MATHEMATIK zu Hause. Ein Kapitel für sich ist die integrierte Programm zur graphischen Darstellung. Jeder Funktionsgraph kann auf dem Bildschirm gezeichnet werden; je nach Monitor auf Wunsch auch mehrfarbig. Zum gleichen Thema gehören die Fähigkeiten des MATHEMATIK auf dem Gebiet der Geometrie. Alle erdenklichen Flächen und Körper hat der MATHEMATIK im Griff. Auch vor der Vektorgeometrie muß der MATHEMATIK nicht kapitulieren, alle Varianten sind implementiert. Zusätzlich bietet das Software-Paket ein komplettes Mathematik-Lexikon mit den wichtigsten Begriffen, Formeln und Lösungsansätzen. Mit dem MATHEMATIK läßt sich jedes mathematische Problem lösen. Den Anwendungen sind keine Grenzen gesetzt! MATHEMATIK, DM 99,-*

SPANNUNG, SPIEL und SPORT

Sie suchen möglichst gute Spiele für möglichst wenig Geld? Bitte sehr! SUPER 4 bietet für sage und schreibe nur 49 Mark vier absolute Topspiele: STAR CRASH – ein faszinierendes Weltraumabenteuer. SPUK – ein tolles Kletter- und Leiterspiel mit 29 verschiedenen Bildern.

PANCO – ein Actionspiel mit einem kleinen Mexikaner. CROWN – ein Spielautomat der besten Sorte mit Risikotaste und goldener Serie. Alle Spiele sind bisher in Deutschland unveröffentlicht und wurden von uns aus über 100 Spielen für sie ausgesucht.



Die SUPER 4 Diskette mit den vier Superspielen gibt's jetzt bei Ihrem Händler für nur DM 49,-*



Heute schon die Bundesliga-Tabelle von morgen kennen, das geht mit UNI-TAB. Alle Recherieren, die man ohne dieses Programm nie machen würde, lassen sich in Sekundenbruchteilen durchführen. Wer will, kann mit simulierten Spielergebnissen den Weltmeister '86 vorausberechnen. Aber nicht nur Fußball-Ligen können tabellarisch erfasst werden, fast alle Sportarten sind UNI-TAB-fähig. Gag am Rande: für viele Sportarten stehen die bekannten Piktogramme zur Verfügung.

UNI-TAB ist Stichworten: Menuesteuerung über die Funktionstasten mit leicht verständlichen Auswahlmöglichkeiten – Bedienerfreundlich (Mannschaften werden über Kennzahlen gesteuert) – Ligen mit 3 bis 20 Mannschaften können verwaltet werden (3 bis 38 Spielstage möglich) – unsinnige Ligen (z.B. 13 Mannschaften sollen 5 Spielstage absolvieren) sind ausgeschlossen – favorisierte Mannschaft kann während des Programmablaufs durch reverse Darstellung gekennzeichnet werden – Tabelle kann geändert werden

(wichtig bei Spielanullierungen) – drei verschiedene Tabellensarten können abgespeichert und später eingelesen werden (die aktuelle Tabelle unabhängig von der Vollständigkeit eines Spieltages), der komplette Spieltag (Vollständigkeit und Nummer des Spieltages werden automatisch errechnet), die simulierte Tabelle (der Anwender kann so selbst Schicksal spielen und seinen Tip später mit dem tatsächlichen Geschehen vergleichen) – zwei verschiedene Arten der Saisonübersicht (die statistische Übersicht zeigt an, welche Tabellenplatz das jeweilige Team bei welchem Punkte- und Torverhältnis an den einzelnen Spieltagen einnahm; die graphische Übersicht zeigt die Leistungskurve jeder Mannschaft) – alle Tabellen und Graphiken sind als Hardcopy auf einem Drucker darstellbar – bei Fehlbedienung (z.B. gewünschte Druckausgabe bei nicht eingeschaltetem Drucker) erscheinen leicht verständliche deutsche Fehlermeldungen. DM 69,-*

Alle Programme auf Diskette

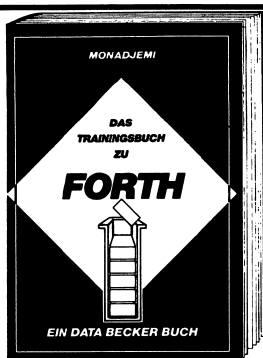
* Bei den Preisen für Programme handelt es sich um unverb. Preisempfehlungen

Ercheinungstermin Dezember '84



DAS TRAININGSBUCH ZU
TURBO PASCAL, ca. 220 Seiten,
DM 39,-

Ercheinungstermin Dezember '84



DAS TRAININGSBUCH ZU FORTH,
ca. 250 Seiten, DM 39,-

Ercheinungstermin Dezember '84



DAS TRAININGSBUCH ZU LOGO,
ca. 200 Seiten, DM 39,-



DAS TRAININGSBUCH ZU
WORDSTAR/MAILMERGE,
ca. 270 Seiten, DM 39,-



DAS TRAININGSBUCH ZU PASCAL,
250 Seiten, DM 39,-



DAS TRAININGSBUCH ZU
MULTIPLAN, 250 Seiten, DM 49,-

Optimales Software-Training

Software richtig nutzen ist nicht schwer – wenn man weiß wie es geht. Leider kommen immer noch die wenigsten Anwender der hochgelobten Standardprogramme mit den mitgelieferten Handbüchern der Hersteller alleine zurecht. Dies liegt nicht nur an den Handbüchern, sondern auch daran, daß den Software-Anwendern oft auch das notwendige Grundwissen fehlt.

DATA BECKER hat deshalb die überaus beliebte und erfolgreiche Reihe der Software-Trainingsbücher geschaffen. Diese Bücher führen nicht nur leicht verständlich in das jeweilige Programm ein, sondern vermitteln auch das notwendige Rüstzeug, um die Programme sinnvoll und erfolgreich einzusetzen. Auch der fortgeschrittene Anwender findet in den Trainingsbüchern viele wichtige Tips & Tricks. Die Autoren dieser Buchreihe haben nicht nur ausgiebige Erfahrungen mit dem richtigen Einsatz der Programme, sondern kennen in der Regel die Sorgen und Wünsche ihrer Leser auch auf Grund intensiver Seminarerfahrung als Referenten.

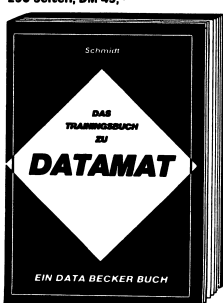
DAS TRAININGSBUCH ZU TEXTOMAT,
200 Seiten, DM 39,-



DAS TRAININGSBUCH ZU SIMON'S
BASIC, 380 Seiten, DM 49,-



DAS TRAININGSBUCH ZU dBASE,
250 Seiten, DM 49,-



DAS TRAININGSBUCH ZU
DATAMAT, 200 Seiten, DM 39,-

DAS STEHT DRIN:

64 Tips & Tricks Bd. 1, mit weit über 70000 verkauften Exemplaren ein Bestseller aus dem Hause DATA BECKER, ist eine echte Fundgrube für jeden COMMODORE 64 Anwender. Mit POKES und anderen nützlichen Routinen, interessanten Programmen sowie wichtigen Programmiertips & -tricks.

Aus dem Inhalt:

- Definition eines eigenen Zeichensatzes
 - Tastaturbelegung und ihre Änderung
 - Dateneingabe mit Komfort
 - Simulation der Maus mit einem Joystick
 - BASIC für Fortgeschrittene
 - CP/M auf dem COMMODORE 64
 - Druckeranschluß über den USER-Port
 - Datenübertragung von und zu anderen Rechnern
 - Expansionport
 - Synthesizer in Stereo
 - Retten einer nicht ordnungsgemäß geschlossenen Datei
 - Erzeugen einer BASIC-Zeile in BASIC
 - Kassettenpuffer als Datenspeicher
 - Sortieren von Stringfeldern
 - Multitasking auf dem COMMODORE 64
 - POKE's und die Zeropage
 - Repeat-Funktion für alle Tasten
- und vieles mehr...

UND GESCHRIEBEN HAT DIESES BUCH:

Das bewährte DATA-BECKER-Autorenteam mit Michael Angerhausen, Lothar Englisch, Klaus Gerits und Frank Thrun. Alle sind nicht nur begeisterte Programmierer, die ihren 64er in- und auswendig kennen, sondern auch bekannte Autoren vieler weiterer Bücher.

ISBN 3-89011-001-0