

**Miedel/Kotulla**

# **Das große CPC-Arbeitsbuch**

Eine aktuelle und praktische Anleitung,  
das Optimum aus den Schneider-Computern  
herauszuholen – für CPC 464, 664 und 6128



***Franzis'***

**Miedel/Kotulla**  
**Das große CPC-Arbeitsbuch**

In der Reihe  
**Franzlis Computer-Praxis**  
sind erschienen:

Benda, Mikrocomputer-Technik praxisnah  
Busch, Basic für Aufsteiger  
Busch, Basic für Einsteiger  
Busch, Der sichere Einstieg in Pascal  
Esders, Das Buch zum Apple II  
Esders, Assembler-Programme zum Apple II  
Feichtinger, Mit Computern steuern  
Haugg, Software-Engineering und ihre Qualitätssicherung  
Janson, Die beiden Datenbanksysteme dBase II und III  
Klein/Klein, Z-80-Applikationsbuch  
Klein, Mikrocomputer selbstgebaut und programmiert  
Klein, Was ist Pascal  
Klein, Die Prozessoren 68000 und 68008  
Link, Messen, Steuern und Regeln mit Basic  
Merker, Hardware-Erweiterungen für den ZX 81  
Miedel/Kotulla: Das große CPC-Arbeitsbuch  
Piotrowski, IEC-Bus  
Plate, Anwenderhandbuch CP/M-68K  
Plate, Betriebssystem CP/M  
Plate/Lechner, Hard- und Software für den Epson HX 20  
Plate/Wittstock, Pascal: Einführung, Programmentwicklung, Strukturen  
Pütz, Das große C64-Arbeitsbuch  
Röckrath, Microsoft-Basic: Konzepte, Algorithmen, Datenstrukturen  
Ruhland, DOS 3.3 – Das Diskettenbetriebssystem des Apple II  
Stenzel, Maschinensprache? – Kein Problem!  
Troitzsch, Mikrocomputer-Schaltungstechnik  
Wunderlich: Erfolgreicher mit CBM arbeiten  
Wunderlich: Erfolgreicher mit dem VC 64 arbeiten  
Zech, Die Programmiersprache Forth

Lothar Miedel / Martin Kotulla

# Das große CPC-Arbeitsbuch

Eine aktuelle und praktische Anleitung, das Optimale aus den  
Schneider-Computern herauszuholen – für CPC 464, 664 und 6128

Mit 30 Abbildungen

---

***Franzis'***



CIP-Kurztitelaufnahme der Deutschen Bibliothek

**Miedel, Lothar:**

Das grosse CPC-Arbeitsbuch: e. aktuelle u.  
prakt. Anleitung, d. Optimale aus d. Schneider-  
Computern herauszuholen – für CPC 464, 664 u. 6128 /  
Lothar Miedel; Martin Kotulla. – München:  
Franzis, 1986.

(Franzis-Computer-Praxis)

ISBN 3-7723-8421-8

NE: Kotulla, Martin:

© 1986 Franzis-Verlag GmbH, München

Sämtliche Rechte – insbesondere das Übersetzungsrecht – an Text und Bildern vorbehalten. Fotomechanische Vervielfältigungen nur mit Genehmigung des Verlages. Jeder Nachdruck, auch auszugsweise, und jegliche Wiedergabe der Bilder sind verboten.

Druck: Buch- und Offsetdruckerei Sommer, Feuchtwangen

ISBN 3-7723-8421-8

## Vorwort

In diesem Buch finden Sie eine Vielzahl nützlicher Informationen, Tips, Kniffe, Tricks und Programme für alle CPC's von Schneider. Sie haben richtig gelesen, für alle drei Schneider-Computer der Typenreihe CPC. Also sowohl für den CPC 464, den CPC 664 und natürlich auch für den CPC 6128.

Soweit es möglich war, haben wir alle Programme dieses Buches so geschrieben, daß sie auf allen drei CPC-Typen laufen und zwar sowohl mit, als auch ohne Diskettenstation. Bei Programmen aber, die beispielsweise direkt auf die Diskettenstation zugreifen, war es natürlich nicht möglich, diese als "Kassettenversion" zu schreiben.

In allen Kapiteln führen wir langsam und detailliert an die von uns erarbeiteten Lösungen heran. Überfliegen Sie deshalb dieses Buch nicht, sondern "erarbeiten" Sie sich das in diesem Buch enthaltene "Wissen". Voraussetzung zum Verständnis dieses Buches ist, daß Sie sich bereits einigermaßen mit dem Handbuch zum CPC (und natürlich auch mit Ihrem Computer selbst) vertraut gemacht haben. Sie wollten mit diesem Buch ja bestimmt auch kein zweites Handbuch, sondern weiterführende Literatur erwerben.

Damit Sie aber den Faden nicht verlieren und vor allem auch deshalb, damit eine gemeinsame Basis vorhanden ist, sollten Sie sich intensiv mit den Grundlagenartikeln beschäftigen, denn erst durch Studium und Verständnis dieser wird auch das Verstehen der Anwendungs-Programmbeispiele gewährleistet sein.

Wir führen Sie langsam, aber sicher, in die "Tiefen" Ihres CPC's. Daß wir in diesem Buch aber nicht alle Möglichkeiten Ihres Computers, nicht alle Sie interessierenden Themen "anpacken" können, werden Sie sicher verstehen. Dadurch aber, daß wir einen sehr weiten Bogen über die gesamte Thematik spannen, die verschiedensten Anwendungen und Lösungen aufzeigen, sollte es Ihnen später nicht schwerfallen, auch Lösungen für Ihre Probleme zu finden.

Viele Fragen, die von CPC-Besitzern an uns herangetragen wurden, wurden in den Artikeln berücksichtigt und werden "so ganz nebenbei" beantwortet.

Durch die vielen Programmbeispiele wollen wir Ihnen aufzeigen, wie Sie Ihren CPC besser nutzen, programmieren und verstehen können.

Wir sind der Meinung, daß man durch eigenes Handeln mehr lernt als durch "graue Theorie" und haben deshalb - gerade auch bei

vielen Grundlagen-Themen, bei denen "Erkenntnisse" erarbeitet werden sollen - auch immer wieder kleine Programme "parat", die das Verständnis erleichtern.

Auch die Anwendungs-Programme stellen in vielen Fällen immer nur ein "Grundgerüst" dar und sollen Sie animieren, diese zu Ihren eigenen, auf Ihre Belange zugeschnittenen, Programme auszubauen. Wir haben deshalb auf effektvollen "Schnick-Schnack" verzichtet, andererseits aber so programmiert, daß die Darstellungen auf dem Bildschirm doch sehr übersichtlich sind. Soweit es vertretbar war, haben wir versucht, die Listings selbst so übersichtlich zu gestalten, damit Ihnen die Analysen dieser Programme leichter fallen.

Zu fast allen Maschinenprogrammen finden Sie auch die zugehörigen kommentierten Assemblerlistings, denn dieses Buch soll Ihnen auch nach Jahren noch als Arbeitsbuch dienen. Wir haben es bei uns selbst erlebt, daß wir - als wir mit der "Computerei" anfangen - eigentlich nur in der Programmiersprache Basic arbeiten wollten, aber irgendwann, reichte Basic nicht mehr aus und dann wären wir froh gewesen, wenn wir solche Hilfen, wie die in diesem Buche enthaltenen kommentierten Assemblerlistings, gehabt hätten. Lassen Sie sich deshalb, auch wenn Sie sich bisher mit Maschinensprache überhaupt nicht beschäftigt haben (und bisher vielleicht auch gar nicht vorhaben, dies jemals zu tun), nicht durch die kommentierten Assembler-Listings verschrecken, denn im Laufe der Zeit werden Sie vieles fast automatisch verstehen. Außerdem können Sie die Programme auch nutzen, ohne daß Sie jedes Detail verstehen.

Da wir wissen, daß es beim Abtippen von Programmen sehr leicht zu "Übernahmefehlern" kommen kann, haben wir lange überlegt, ob wir einen sogenannten Checksummer in diesem Buch veröffentlichen sollen. Wir haben uns dagegen entschieden, denn dieser ist nur dann von Nutzen, wenn Sie alle Programme genau so schreiben, wie die Originale. Jede geringste Abweichung - und sei es nur ein zusätzliches Leerzeichen (das in den meisten Fällen unkritisch wäre) - würde zu nervtötenden Fehlermitteilungen führen und Sie immer wieder verunsichern. Die Programmeingabe würde, unserer Meinung nach, dadurch sogar erschwert werden. Deshalb also ohne "Checksummer".

Bei den Hex- oder Basicladern, also bei den Programmen, die Maschinencode als Hexadezimalzahlen in Datazeilen enthalten, haben wir meist nur eine kleine Prüfroutine, die grobe Fehler erkennt, eingebaut. In einigen Fällen aber auch Prüfroutinen, die sehr "stark" sind und auch Vertauschungen von Statements in einer Zeile erkennen.

Wer trotzdem noch Probleme hat oder sich die mühevollen Arbeit des Abtippens ersparen will, kann alle größeren Programme (bei solchen mit nur wenigen Programmzeilen haben wir es unterlassen) dieses Buches vom Franzis-Software-Service auf Diskette beziehen. Da diese Programme die Originale sind, können Sie sich jederzeit davon überzeugen, daß dann, wenn ein Programm nicht läuft, meist eben doch der Fehler beim Abschreiben lag.

Bei sehr vielen Programmen sind wir sogar so vorgegangen, daß wir diese nach den Prüfungen auf Fehlerfreiheit in ASCII-Files umgewandelt und dann in den Buch-Text eingelesen haben. Anschließend wurden diese Programme aus dem Text heraus wieder isoliert und nochmals geprüft! Obwohl gerade diese mehrfachen Prüfungen sehr, sehr viel Zeit kosteten, können Sie sicher sein, daß von unserer Seite wirklich sehr viel getan wurde, damit die Programme dieses Buches (was den Abdruck betrifft) fehlerfrei sind. Trotzdem könnte es aber vorkommen, daß wir Fehler "einprogrammiert" oder übersehen haben, diese könnten sich aber nicht so auswirken, daß ein Programm nicht läuft, denn diese Programme laufen!

Wenn Sie also Probleme mit einem Programm haben, dann liegt es wahrscheinlich nicht an dem veröffentlichten Programm, sondern muß andere Ursachen haben. In den meisten Fällen wird es sich um Abtippfehler handeln.

Aber auch andere Gründe kann es geben, daß Programme nicht das tun, was sie tun sollten. Dies kann verschiedenste Ursachen haben. Um Ihnen dies zu verdeutlichen, wollen wir hier gleich etwas konkreter werden. Eine angeschlossene Speichererweiterung, oder angeschlossene "Sideway-ROMs", die genau den Bereich belegen oder benutzen, in dem die Maschinenprogramme dieses Buches ablaufen sollen, können genauso zu Problemen führen, wie beispielsweise ein angeschlossener Drucker, der eine Hardcopy ausgeben soll und dies aber technisch gar nicht kann.

Die Programme arbeiten also immer auf den "Grundsystemen". Sobald weitere Peripherie angeschlossen wird, hängt es von dieser ab, ob es zu Problemen kommt. Da wir nicht alle peripheren Geräte und Zusätze, die es bis jetzt gibt und noch in der Zukunft geben wird, kennen können, konnten Eigenarten und Restriktionen, die durch deren Anschluß erfolgen, nicht berücksichtigt werden.

Alle Programme wurden sehr ausgiebig und sorgfältig getestet. Trotzdem könnte es vorkommen, daß irgendwann einmal kleine Fehler auftreten (Nobody is perfect). Bitte teilen Sie uns diese mit, damit sie bei einer Neuauflage des Buches berücksichtigt werden können.

Wir wünschen Ihnen viel Spaß und Erfolg bei Arbeiten mit Ihrem Schneider CPC.

München/Nürnberg

Martin Kotulla  
Lothar Miedel



## Diskette zum Buch

Zu diesem Buch

gibt es eine **Sammeldiskette** von den abgedruckten Programmen.

Sie können diese Diskette bei unserem Franzis-Software-Service, Postfach 37 01 20, 8000 München 37, direkt bestellen, oder verwenden Sie bitte die beiliegende Bestellkarte.

Bestell-Nr.: 3-8178-0232-3

Preis: 49,- DM

Liefermöglichkeit und Preisänderung vorbehalten.

## Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen und Verfahren werden ohne Rücksicht auf die Patentslage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden\*).

Alle Schaltungen und technischen Angaben in diesem Buch wurden vom Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. Der Verlag sieht sich deshalb gezwungen, darauf hinzuweisen, daß er weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen kann. Für die Mitteilung eventueller Fehler sind Autor und Verlag jederzeit dankbar.

---

\* Bei gewerblicher Nutzung ist vorher die Genehmigung des möglichen Lizenzinhabers einzuholen.



# Inhaltsverzeichnis

1.	Einführung.....	17
TEIL 1 (GRUNDLAGEN)		
2.	Allgemeine Grundlagen.....	21
2.1	Einige Vorbemerkungen.....	21
2.2	Die Speichertypen.....	22
2.3	Der RAM-Speicher beim CPC.....	23
2.4	Die mitgelieferte Intelligenz.....	24
2.5	Verschiedene Zahlensysteme.....	25
2.6	Das Sedezimalsystem.....	25
2.7	Umrechnungen mit dem Schneider.....	27
3.	Hardware, die "anfassbare Ware".....	31
3.1	Der Prozessor Z80A.....	32
3.2	Der Soundgenerator (AY-3-8912).....	35
3.3	Der Schnittstellenbaustein (PIO 8255).....	36
3.4	Der VDU-Controller (8845).....	37
3.5	Das Gate-Array.....	39
3.6	Das Firmware-ROM.....	43
3.7	Zusätzliche Bausteine für die Diskettenstation.....	44
3.7.1	Der UPD 765.....	44
3.7.2	Das Floppy-ROM.....	44
4.	Wichtige Details.....	46
4.1	Der RAM-Speicherbereich.....	46
4.2	Die Variablen und Konstanten.....	47
4.3	MON - Ein modular aufgebautes Hilfsprogramm.....	50
4.3.1	Programmbeschreibung des Hexladers MON1.HEX.....	50
4.3.2	Listing: MON1.HEX.....	52
4.3.3	Listing: Basicteil von MON1.....	56
4.3.4	Assemblerlisting: MON1.ED1.....	59
4.3.5	Listing: HC.HEX eine Hardcopy-Routine.....	65
4.3.6	Ein starkes Hardcopy-Programm.....	67
4.3.7	Assemblerlisting: HC.ED1.....	67
4.3.8	Falls Sie einen speziellen Drucker haben.....	69
4.3.9	Umgehung des MERGE-Fehlers beim 464.....	72
4.3.10	Die Bedienung des Monitor-Programmes.....	73
4.4	Erste Analysen.....	75
4.5	Die Token.....	77
4.5.1	Eine Zeile läßt sich nicht listen.....	77

5.	Einblick in den RAM-Speicher.....	82
5.1	Die Programmablage.....	82
5.2	Wir sehen uns im Speicher um.....	88
5.2.1	Listing: KILLREM.HEX.....	89
5.2.2	Assemblerlisting: KILLREM.EDI.....	90
5.3	Der Tastaturpuffer der CPCs.....	95
5.3.1	Wir manipulieren den Tastaturpuffer.....	95
5.3.2	Listing: TCODE.....	96
5.3.3	Programmbeschreibung TCODE.....	97
5.3.4	Listing: Datagenerator für alle CPCs.....	98
5.3.5	Programmbeschreibung DGEN.BAS.....	100
5.4	Kassetteninhaltsverzeichnis.....	102
5.4.1	Programmbeschreibung: Kassetteninhaltsverzeichnis....	104
5.4.2	Listing: Kassetteninhalt.....	106
5.4.3	Druckprogramm zum Kassetten-Inhaltsverzeichnis.....	108
6.	Der Bildschirm und sein Speicher.....	109
6.1	Listing: SCREEN.BAS.....	110
6.2	Listing: Offset.....	112
6.3	Noch mehr Details über den Bildschirmspeicher.....	112
6.4	Listing: PIXZEIL.BAS.....	114
6.5	Tabelle der Pixelzeilenadressen.....	116
6.6	Listing: PIXCODE12.BAS.....	120
6.7	Codierung der Zeichen im MODE 0.....	123
6.7.1	Listing: MODE0COD.BAS.....	124
6.7.2	Listing: AUFL0ES.BAS.....	125
6.7.3	Listing: MODE1COD.....	126
6.8	Sprites auf den CPCs.....	128
6.8.1	Listing: SPRITE.....	129
6.8.2	Bedienung des Programmes SPRITE.....	131
6.8.3	Listing: SPDEMO.BAS.....	133
6.8.4	Assemblerlisting: SPMOVE.EDI.....	134
6.8.5	Übersicht: Codierung einzelner Pixel im MODE 1 ...	135
6.9	Umkopieren von Bildschirmzeilen.....	135
6.9.1	Assemblerlisting: LINECOPY.EDI.....	136
6.9.2	Listing: LINECOPY.HEX.....	137
6.9.3	Listing: LINECOPY.BAS.....	137
6.10	Die Hardware-Nummern der Farben.....	139
6.10.1	Farbwerte sortiert nach Normalwerten.....	140
6.10.2	Farbwerte sortiert nach POKE-Werten.....	140
6.11	Ein eigener Zeichensatz.....	141
6.11.1	Listing: Zeicheneditor.....	142
6.11.2	Listing: DEUTAST.....	144
6.11.3	Listing: Lottovorschlag.....	145
6.11.4	Listing: TEBENE.BAS.....	146
6.11.5	Listing: Invertierung von PEN und PAPER.....	147
6.11.6	Listing: Farbwahl.....	148
6.11.7	Listing: Schnellkurs Farbprogrammierung.....	148
6.11.8	Die Steuerzeichen der CPCs.....	153
6.11.9	Listing: MINIMAL.BAS.....	155
7.	Die Firmware-Pakete (Packs).....	159
7.1	Übersicht der Firmware-Packs.....	159
7.2	Die Aufgaben der einzelnen Pakete.....	160
7.3	Die Unterteilung der Einträge.....	161
7.4	Weitere Indirections und Speicherbereiche.....	163
7.5	Die RAM-Bereiche.....	163

8.	Die Sideways-ROMs.....	166
8.1	Die ersten fünf Bytes eines Erweiterungs-ROMs.....	168
8.2	Assemblerlisting: CPC-ART.EDI.....	169
8.2.1	Listing: CPC-ART.HEX.....	170
8.3	Machen Sie aus Ihrem CPC 464 einen CPC 664.....	171
8.4	Wie macht man aus einem 464 einen 664?.....	172
8.5	Wo liegen externe ROMs?.....	172
8.6	Wir "blenden" ein und aus.....	173
9.	Rund um die Diskette.....	176
9.1	Listing: MONERW.BAS.....	176
9.2	Listing: SECREAD.HEX.....	177
9.3	Die Diskette.....	179
9.4	Das Formatieren.....	183
9.4.1	Listing: FORMAT.HEX.....	184
9.4.2	Assemblerlisting: FORMAT.EDI.....	185
9.5	Ein absichtlicher Sektor-Fehler.....	189
9.5.1	Listing: WRONGSEC.HEX.....	189
9.5.2	Assemblerlisting: WRONGSEC.EDI.....	189
9.5.3	Assemblerlisting: SNRCHECK.EDI.....	191
9.5.4	Listing: SNRCHECK.BAS.....	193
9.6	Der direkte Diskettenzugriff.....	194
9.6.1	Assemblerlisting: SECREAD.EDI.....	194
9.6.2	Listing: SECREAD.BAS.....	195
9.6.3	Listing: DINFO.BAS.....	196
9.6.4	Listing: PCOPY.EDI.....	198
9.6.5	Listing: PCOPY.BAS.....	199
9.6.6	Listing: PCOPY.HEX.....	199
9.6.7	Listing: FRETTER.....	201
9.6.8	Listing: SECTOR.HEX.....	203
10.	Hilfen zur Maschinensprache-Programmierung.....	204
10.1	Was macht ein Disassembler?.....	205
10.1.1	Listing: Z80CODE.....	206
10.1.2	Listing: READER.....	210
10.1.3	Listing: FIRMJUMP.....	211
10.1.4	Listing: FJLESE.....	211
10.1.5	Listing: SAGEN.....	212
10.1.6	Listing: SALESE.....	213
10.1.7	Listing: ANADIS.....	214
10.1.8	Programmbeschreibung: ANADIS.....	218
10.2	Aus dem Disassembler wird ein Reassembler.....	220
10.2.1	Listing: REASS.....	220
10.3	Die Garbage-Collect-Routine.....	222
10.3.1	CPC-Uhr.....	222
10.4	Der CALL-Befehl.....	226
10.4.1	Assemblerlisting: CALL-DEMO.EDI.....	227
10.4.2	Listing: CALLEDemo.BAS.....	227
10.5	REM-Routinen.....	228
10.5.1	Assemblerlisting: BASMPG1.....	228
10.5.2	Listing: BASMPGB.....	228
11.	DEC\$ auf dem CPC 464.....	230
11.1	Listing: DEC\$.....	231
11.2	Listing: Befehle.....	232
11.3	Listing: \$-Finder.....	232

<b>TEIL 2 (ANWENDUNGEN).....</b>		<b>233</b>
<b>12.</b>	<b>Das Floppy-RAM läßt sich verschieben!.....</b>	<b>233</b>
12.1	Assemblerlisting: FMOVE.ASM.....	234
12.2	Listing: FLMOVE.BAS.....	235
<b>13</b>	<b>DIRSYS listet auch Systemfiles.....</b>	<b>238</b>
13.1	Assemblerlisting: DIRSYS.....	240
13.2	Listing: DIRSYS.BAS.....	243
<b>14</b>	<b>Echte Unterlängen für den Drucker GP-500.....</b>	<b>245</b>
14.1	Listing: SEIKO.BAS.....	245
14.2	Listing: TRANSF.BAS.....	251
<b>15</b>	<b>Zwei Zeichensätze für den Seikosha-Drucker.....</b>	<b>254</b>
15.1	Assemblerlisting: IZGP500.....	254
15.2	Listing: IZGP500.BAS.....	257
15.3	Listing: ZEISADEM.BAS.....	259
<b>16</b>	<b>Auch Maschinenprogramme lassen sich unterbrechen!.....</b>	<b>262</b>
16.1	Assemblerlisting: MCBREAK.....	262
16.2	Listing: MCBREAK.BAS.....	264
<b>17</b>	<b>DEFAULTS setzt den Computer zurück.....</b>	<b>269</b>
17.1	Assemblerlisting: DEFAULTS.....	269
17.2	Listing: DEFAULTS.BAS.....	271
<b>18</b>	<b>RSX-Befehle intern.....</b>	<b>275</b>
18.1	Assemblerlisting: GETADDR.....	275
18.2	Listing: GETADDR.BAS.....	277
18.3	Assemblerlisting: RSXLIST.....	282
18.4	Listing: RSXLIST.BAS.....	285
<b>19</b>	<b>Es gibt nicht nur einen Bildschirmspeicher!.....</b>	<b>292</b>
19.1	Assemblerlisting: SCREENUTIL.....	293
19.2	Listing: SCREENU.BAS.....	297
<b>20</b>	<b>Blitzschnell eingeblendet.....</b>	<b>302</b>
20.1	Assemblerlisting: HIDESHOW.....	302
20.2	Listing: HIDESHOW.BAS.....	304
<b>21</b>	<b>Der Schneider versteht auch fremde Diskettenformate....</b>	<b>308</b>
21.1	Listing: DPBSHOW.BAS.....	314
21.2	Assemblerlisting: DPBSHOW.COM.....	315
21.3	Assemblerlisting: SREAD.ASM.....	321
21.4	Listing: SECREAD.BON.....	322
21.5	Listing: FORMAT.TAL.....	326
21.6	Listing: FORMAT.BON.....	326
21.7	Listing: OSMOVE.ASM.....	327
21.8	Listing: OSMOVE.BAS.....	328
21.9	Listing: FORMAT.OSB.....	328
21.10	Listing: FORMAT.OS2.....	329
21.11	Assemblerlisting: ACCESS.ASM.....	330
21.12	Listing: ACCESS.BAS.....	333
21.13	Assemblerlisting: SF.COM.....	341
21.14	Turbo-Pascal-Listing: SETFORMATPAPN.....	345

22	Funktioniert Ihr Computer richtig?.....	350
22.1	Assemblerlisting: MEMORY CHECK.....	350
22.2	Listing: MEMCHECK.BAS.....	354
22.3	Assemblerlisting: MEMORY CHECK 6128.....	357
22.4	Listing: MCHE6128.BAS.....	362
23	Trickreiches rund um die Tastatur.....	364
23.1	Assemblerlisting: KEYBOARDKIT.....	367
23.2	Listing: KBKIT.BAS.....	372
23.3	Listing: KEYPROG.BAS.....	378
24	Dateien kopiert und verbunden.....	380
24.1	Assemblerlisting: COPYFILE.....	380
24.2	Listing: COPYFILE.BAS.....	382
24.3	Assemblerlisting: CONCAT.....	385
24.4	Listing: CONCAT.BAS.....	387
25	Listschutz auf CPC-664 und 6128.....	389
26	Der Screen-Editor.....	392
26.1	Listing: SCREENED.....	393
27	SYNOPSIS - Der Durchblick beim CPC.....	394
27.1	Listing: CODEGEN.BAS.....	395
27.2	Assemblerlisting: Hilfsroutinen für SYNOPSIS 6128....	400
27.3	Assemblerlisting: COPYROM.ASM.....	403
27.4	Listing: COPYROM.BAS.....	406
27.5	Listing: SYNOPSIS.BAS.....	407
28	Sparen Sie Farbbänder!.....	423
28.1	Listing: FBSAVE.BAS.....	423
29	VARLIST - Variablen gelistet.....	425
29.1	Listing: VARLIST.BAS.....	426
30	Benutzertip MAXAM - mal anders.....	430
30.1	MAXCALL.BAS.....	431
	TEIL 3 (Anhang).....	432
31	Anhang 1 - Firmware-Einsprünge.....	432
32	Anhang 2 - Die System-Adressen.....	435
33	Anhang 3 - Die Betriebssystem-Erweiterungsplatine.....	444
34	Anhang 4 - PINOUTs und Anschlußbelegungen.....	446



# 1. Einführung

## Die CPC's von Schneider

Eigentlich ist die obige Überschrift sachlich nicht ganz korrekt, denn wie Ihnen sicher bekannt ist, wurden diese Computer von der englischen Firma Amstrad entwickelt und werden in Deutschland durch die Firma Schneider vertrieben. Trotzdem wird in diesem Buch aber auch immer wieder von den "Schneidern" zu lesen sein.

Als im Herbst 1984 der erste Schneider CPC 464 auf den Markt kam, wurde dieser als ein sogenannter Kompaktcomputer angeboten. CPC ist auch nichts anderes als die Abkürzung der englischen Bezeichnung: Compact Personal Computer. Als kompakt wurde er deshalb bezeichnet, weil er - im Gegensatz zu vielen anderen Computern - alles enthielt, was für einen Rechnerbetrieb erforderlich ist. Ein Sichtgerät (Monitor) mit integriertem Netzteil, die Rechnerkonsole mit integrierter Tastatur und der eingebaute Massenspeicher (das Kassettenlaufwerk), stellen ja wirklich einen kompletten Computer dar. Kommt allerdings noch erforderliche Peripherie hinzu, dann geht auch die CPC-Kompaktheit schnell wieder verloren.

## Anfängliche Skepsis

Als der CPC 464 vorgestellt bzw. auf den Markt gebracht wurde, waren viele skeptisch, wie denn ein neuer Rechner (mit dem doch schon antiquierten Z-80-Prozessor) gegen die etablierten Vertreter der Home- und Personalcomputer antreten und Marktanteile gewinnen sollte. Doch diese Zweifel und diese Skepsis wichen bald einer großen Begeisterung für diesen Newcomer. Denn es sprach sich sehr schnell herum, daß ihm ein gutes Basic vom englischen Softwarehaus Locomotive verpaßt worden war.

Aber auch das Gesamtkonzept des CPC fand nicht nur anerkennende Worte in der Fachpresse, sondern auch bei Computerinteressenten und vor allem bei den Besitzern. So begann ein anfangs eigentlich gar nicht so sehr erwarteter "Aufstieg". Die Wahl zum Homecomputer des Jahres 1985 war fast nur noch ein "Nebenbei".

## Nicht nur Freudentränen

Nach der Markteinführung des 464 gab es aber dann doch Probleme, die nicht unbedingt zur Freude der Besitzer beitrugen. Die Diskettenstation und der Drucker ließen auf sich warten, Software war auch nur in beschränktem Umfang erhältlich und noch weitere



Kleinigkeiten trübten den - ansonsten doch sehr positiven - Gesamteindruck. Dies ist aber keine große Ausnahme in einem Markt, der sich mit täglichen Neuigkeiten fast überschlägt, bei dem Computer kometenhaft aufsteigen und in einigen Fällen mit noch höherer Geschwindigkeit wieder "abstürzen". Vieles wird angekündigt und wieder zurückgezogen, es wird versprochen und wieder etwas später hatte man sich eben "versprochen". Computer werden vorgestellt und sind erst Jahre danach lieferbar.

Sie sehen: Eine turbulente Szene stellt der Computermarkt also schon dar. Deshalb wurden viele CPC 464-Besitzer auch etwas geschockt, als nach etwa einem halben Jahr bereits der Nachfolger angepriesen wurde, der CPC 664. Dieser hatte - im Gegensatz zu seinem bereits zu diesem Zeitpunkt sehr erfolgreichen Vorgänger - anstelle des eingebauten Kassettenrecorders eine Diskettenstation und erfüllte den Anspruch "kompakt" wieder mehr als der 464 mit nachgerüsteter Floppy. Es war zwar ein Verbindungskabel zwischen Monitor und Konsole zusätzlich erforderlich, denn zum Betrieb der Diskettenstation wurden neben der erforderlichen Spannungsversorgung von 5 Volt auch noch 12 Volt benötigt, aber durch die eingebaute Diskettenstation entfiel andererseits das Disketteninterface und auch noch das Netzkabel für die Diskettenstation.

Da bei den "alten" Monitoren aber kein 12-Volt-Ausgang vorgesehen war, bedeutete dies auch einen neuen Monitor. Der Unterschied ist aber lediglich eine kleine Zusatzplatine mit dem 12-Volt-Netzteil und eine entsprechende Ausgangsbuchse am Monitor.

Das "Gesicht" des 664 wurde ebenfalls neu "gestylt", denn anstelle des Kassettenlaufwerkes mußte die Diskettenstation in das Gehäuse "verpackt" werden. Dabei fiel auch die Tastatur der Verschönerung zum Opfer. Mit anderen Worten der 664 unterschied sich optisch zwar stark vom 464, aber es war immer noch deutlich zu sehen, daß es ein Schneider-CPC war.

### **Die Kompatibilität blieb auf der Strecke**

Schlimm und auch ärgerlich für die Besitzer der 464-Ausführung aber war die sich nach und nach erhaltende Erkenntnis, daß trotz der Versprechungen von größtmöglicher Kompatibilität, die Firmware des "Neuen" durch Locomotive Software total "umgekrempelt" worden war.

Zwar hatte man das neue Basic erweitert, das heißt es waren neue Befehle in den Basic-Befehlssatz aufgenommen worden, aber dies war nicht der alleinige Grund, für derartig große Änderungen im eingebauten Firmware-ROM. In der Firmware des CPC 464 waren nämlich ein paar mehr oder weniger stark bemerkbare Fehler enthalten und diese mußten ebenfalls beseitigt werden.

Die Folge war, daß manche für den CPC 464 geschriebene Software nicht mehr auf dem 664 lief und auch umgekehrt.

Einen ähnlichen "Verbesserungseffekt" hatten Besitzer eines 464 schon einmal vermeiden können. Nach Anschaffung einer Disketten-

station liefen plötzlich manche Kassettenprogramme nicht mehr, denn der Anschluß des Laufwerkes "kostete" 1284 Bytes des oberen RAM-Bereiches. Kein Wunder also, daß 464-Besitzer, die mit ihrem bisherigen Schneider eigentlich hochzufrieden waren, verunsichert wurden und sich fragten, ob sie sich nun den 664 zulegen sollten oder nicht.

Um der ganzen Verunsicherung dann noch die "Krone aufzusetzen", kam etwa zwei Monate nach der Vorstellung des CPC 664 ein weiterer Schock, der selbst informierte Insider überraschte: Der CPC 6128 wurde vorgestellt! Zwar rechnete man damit, daß über kurz oder lang eine Reaktion auf den von Commodore angekündigten C128 erfolgen würde, aber kaum jemand ahnte, daß dies so schnell geschehen würde.

Die Fachpresse war noch nicht einmal mit der "Abhandlung" des 664 fertig, schon kam der nächste, ganz zu schweigen von der Vorstellung des vierten Schneiders, von JOYCE. Letzterer (oder sollte man schreiben letztere?) wird in diesem Buch allerdings nicht behandelt, da er nicht zur CPC-Familie gehört.

Nun aber war die Verwirrung komplett, denn auch der CPC 6128 war zu seinem Vorgänger nicht voll kompatibel! Zwar waren die Unterschiede (abgesehen von der zweiten 64 K-Bank) nicht ganz so groß wie zwischen dem 664 und dem 464, aber sie waren da. Und deshalb muß bei den in diesem Buch enthaltenen Programmen eben ab zu eine Anpassung an den jeweiligen CPC-Typ erfolgen.

Zwischenzeitlich wird bereits wieder von einem neuen Schneider gemunkelt, der aber ebenfalls nicht zur CPC-Familie gehören soll. Während der Arbeiten an diesem Buch erreichte uns auch noch die Nachricht, daß Amstrad die Vertriebsrechte für Sinclair übernommen hat, vielleicht ist deswegen demnächst auch ein "CPC-ZX" zu erwarten? Wahrscheinlicher aber ist, daß man beide Namen nicht vermischen will, sondern daß diese getrennt nebeneinander leben dürfen.

Neben Amstrad und Schneider waren aber auch andere Firmen aufgewacht und aktiv geworden, die vermeintliche oder auch echte Schwächen der CPCs insofern ausnutzten, daß sie Zusatzperipherie entwickelten und anboten.

Diskettenlaufwerke höherer Speicherkapazität oder günstigeren Preises, Speichererweiterungen um unter CP/M einen größeren TPA (Transient Programm Area) zur Verfügung zu stellen, Erweiterungs-ROMs mit den verschiedensten Hilfsprogrammen (Utilities), Zusatzplatinen für unterschiedlichste Zwecke, RS-232-Schnittstellen usw. sollen die CPCs zu noch mehr Leistung befähigen.

Auch Harddisks werden an die CPCs adaptiert, ja sogar eine MS-DOS-Platine kann den "Schneidern" verpasst werden.

Alle diese Vorinformationen sollten Sie nur etwas in die Hintergründe einführen, weshalb es nicht immer ganz einfach, ja manchmal sogar unmöglich war und ist, Programme auf allen drei CPC-Vertretern und allen Zusätzen ohne Tricks lauffähig zu bekommen.

Wir haben uns aber darum bemüht, die Unterschiede aufzuzeigen und in einer Tabelle alle von uns bereits gefundenen unterschiedlichen RAM-Adressen zusammengefaßt. Dadurch sind Sie dann evtl. in der Lage, Programme die speziell für einen CPC-Typ geschrieben wurden, auch für einen anderen CPC-Typ selbst anzupassen.

Diese Tabelle wurde zu ca. 80 % durch einen - in diesem Buch ebenfalls mit abgedruckten - "analysierenden" Disassembler erarbeitet.

Mit dessen Hilfe kann sich jeder CPC-Besitzer ein eigenes kommentiertes ROM-Listing erstellen. Bei den "Adressen-Namen" haben wir uns sehr stark an das im Huslik-Verlag erschienene Buch "CPC Inside out" und an das von Schneider erhältliche Firmware-Handbuch angelehnt, damit die ohnehin schon ausreichende Verwirrung nicht noch größer wird.

Ein weiterer Grund für die Veröffentlichung dieses Disassemblers ist die Tatsache, daß es bis dato weder für den CPC 664 noch für den CPC 6128 komplette, kommentierte ROM-Listing gibt, welche zu den jeweiligen Adressen und Kommentaren auch die Mnemonic enthalten. Der Grund dafür, daß es ein derartiges Buch im Augenblick nicht gibt, liegt daran, daß der Chef von Amstrad (Alan Shugar) anscheinend etwas dagegen hat. Eigentlich schade, denn für jeden, der sich intensiver mit seinem CPC beschäftigen will, sind derartige Hilfen unentbehrlich. Für ein dauerndes "Parallelesen" von Adressangaben mit Kommentaren und Disassemblerlisting ist schon eine gehörige Portion an Selbstbeherrschung nötig, damit man nicht verzweifelt. Zwar sind in der Zwischenzeit schon kommentierte ROM-Listings als Bücher erschienen, aber diese bauen auf dem 464 auf und bringen nicht alle Unterschiede der CPCs.

Durch das in diesem Buch enthaltene Disassembler-Programm kann sich jeder Interessent sein eigenes "Intern" erstellen. Wir möchten aber vorab gleich darauf hinweisen, daß es mit viel Arbeit verbunden ist, denn denken kann der Disassembler nicht. Alles was er wissen muß, müssen Sie ihm erst mitteilen. Wer kein eigenes ROM-Listing erstellen will, kann aber diesen Disassembler ebenfalls benutzen, er braucht ihn dann nur nicht "intelligent" zu machen. In diesem Zusammenhang ist auch das in diesem Buch enthaltene Programm SYNOPSIS zu erwähnen, das es den CPC 6128-Besitzern erlaubt, die Inhalte der drei verschiedenen Firmware-ROMs parallel anzusehen.

Bitte haben Sie aber Verständnis dafür, daß wir bei der enormen Informationsfülle, die wir Ihnen bieten, manchmal nicht immer auf die kleinsten Details eingehen konnten. Dafür erhalten Sie aber Programme, mit denen Sie sich alles selbst ansehen und vor allem erarbeiten können.

So nach diesen einleitenden Worten wollen wir nun näher auf die CPC's selbst eingehen und Ihnen erst einmal etwas Grundlagenwissen vermitteln.

## 2. Allgemeine Grundlagen

### 2.1 Einige Vorbemerkungen

Wir wollen Ihnen nun nicht Grundkenntnisse der Basic-Programmierung beibringen, geschweige denn Ihnen mitteilen, wo sich der Joystickanschluß befindet oder Ähnliches, nein, dies steht alles im Handbuch zu Ihrem System. Vielmehr soll nun eine Basis aufgebaut bzw. geschaffen werden, damit Sie etwas tiefer in Ihren Computer "einsteigen" können. Deshalb müssen wir etwas bei "Adam und Eva" anfangen, um Sie mit Ihrem CPC noch vertrauter zu machen.

Auch Fortgeschrittene sollten diesen Teil nicht einfach überfliegen, denn wer weiß, vielleicht ist doch das eine oder andere Detail dabei?

Bitte haben Sie Verständnis dafür, daß gerade in diesen Grundlagenkapiteln manchmal - thematisch - "kreuz und quer" gesprungen wird, aber viele Dinge, die Sie in Ihrem CPC-Handbuch noch nicht so genau nachgelesen haben, sollen zumindest nochmals kurz aufgefrischt werden, damit Sie mit diesem Buch auch wirklich "arbeiten" können.

#### Vieles ist durch den Prozessor vorgegeben

Wie Sie wissen, ist bei den CPC's (als "Herz" des Systems) ein Prozessor vom Typ Z80A eingesetzt. Entwickelt wurde dieser Mikroprozessor von der Firma ZILOG und ist zu dem von der Firma INTEL entwickelten 8080 aufwärts-kompatibel. Das bedeutet, daß für den 8080 entwickelte Programme auch mit dem Z80 laufen können.

Der Z80-Prozessor ist ein sogenannter 8-Bit-Prozessor. Dies wiederum bedeutet, daß er einen 8-Bit-breiten Datenbus besitzt. Er hat also acht Datenleitungsanschlüsse.

In technischen Unterlagen werden die Anschlüsse dieses Busses meist mit D0 bis D7 gekennzeichnet. Im Regelfalle also nicht mit D1 bis D8!

Jedes Bit, also jeder dieser Anschlüsse (Leitungen), kann zwei Zustände signalisieren bzw. signalisiert bekommen, nämlich Signal vorhanden oder nicht vorhanden. Elektrisch betrachtet, also Spannung anstehend oder keine Spannung.

Bei 8 Bits sind  $2 \text{ hoch } 8 = 256$  Kombinationsmöglichkeiten gegeben, etwas "mitzuteilen". Es ist also möglich, insgesamt 256 Varianten von Ein- und Aus-Zuständen zu signalisieren.

Rechnen Sie es ruhig einmal auf Ihrem CPC im Direktmodus nach:

Print 2 hoch 8

Anstelle des Wortes "hoch" müssen Sie allerdings den Hochpfeil, der sich links neben der Taste CLR befindet, eingeben. Also:

PRINT 2^8 (Enter-Taste)

Im Buch und auch in den Listings erscheint für dieses Zeichen immer nur der obere Teil dieses Pfeiles, also ein kleines "Dach".

(Ausnahmsweise haben wir diesmal darauf hingewiesen daß Sie, damit Ihr Schneider die Aufgabe ausführt, die ENTER-Taste drücken müssen. Diesen Hinweis werden wir ab sofort unterlassen, denn dies gehört zum Grundwissen!)

Als Antwort erhalten Sie den Wert 256. Damit haben Sie dann die Bestätigung dafür, daß mit 8 Bit 256 Werte darstellbar sind.

Bei der "Computerei" werden acht Bit auch ein Byte genannt. Manchmal wird auch von Wort oder Datenwort gesprochen. Da es aber auch z.B. 16- oder 32-Bit "breite" Datenwörter gibt und wir versuchen wollen möglichst "einstimmig" zu sprechen, merken Sie sich bitte, daß acht Bit ein Byte sind. Anders ausgedrückt, ein Byte hat acht Schalter, die (jeder für sich) ein- oder ausgeschaltet sein können.

## 2.2 Die Speichertypen

Ihr CPC hat - je nach Typ - entweder 64 oder 128 KByte RAM-Speicher. RAM ist die Abkürzung von Random Access Memory und heißt soviel wie Speicher mit wahlfreiem Zugriff. Es handelt sich also um einen Schreib-/Lesespeicher, aus dem sowohl "ausgelesen" als auch "eingeschrieben" werden kann.

Im Gegensatz hierzu gibt es beispielsweise auch ROMs. Diese Abkürzung von Read Only Memory ist die Bezeichnung für reine Nur-Lese-Speicher. Aus diesen kann zwar ausgelesen, aber es kann nichts mehr hineinschreiben werden. Das "Einschreiben" geschah bei der Herstellung des Chips und ist nachträglich nicht mehr zu ändern.

Sehr nahe "verwandt" mit den ROMs sind EPROMs (ERASEABLE PROGRAM-ABLE Read Only Memory), die zwischen den beiden vorgenannten Typen einzuordnen sind.

EPROMs können durch entsprechende Einrichtungen (Eprommer) beschrieben und mittels ultravioletem Licht auch wieder gelöscht werden. Sie besitzen deshalb nach der Programmierung die Eigenschaften von ROMs und können diese auch ersetzen.

Es gibt zwar auch noch elektrisch löschbare EPROMs (EEPROM), aber diese sind derzeit für die CPCs noch nicht aktuell.

## 2.3 Der RAM-Speicher beim CPC

Was bedeutet es nun, daß Ihr CPC entweder 64 oder 128 KByte RAM besitzt?

Um dies zu erfahren, muß zunächst erst geklärt werden, was der Buchstabe "K" vor dem Wort Byte bedeutet. Dieses "K" (in Großschreibweise) wird zwar als "Kilo" gesprochen, bedeutet aber im Gegensatz zum Vorsatz "k" (Kleinschreibweise) aus dem sonstigen täglichen Leben nicht eine Multiplikation mit dem Wert eintausend, sondern mit 1024 (eintausendundvierundzwanzig). Warum dies so ist, werden Sie bald selbst erkennen.

Zunächst aber noch einmal zur Frage zurück, welche Speichergröße mit wahlfreiem Zugriff Ihr CPC besitzt.

Nachdem Sie nun wissen, was "K" bedeutet, brauchen Sie es nur noch auszurechnen. 64 KByte sind 64 mal 1024 Byte also 65536 Byte und 128 KByte sind 128 mal 1024 Byte also 131072 Byte.

Diese Anzahl von Speicherplätzen kann bei Ihrem CPC frei bearbeitet werden. Damit Sie es aber nicht verkehrt verstehen, dies bedeutet lediglich, daß in diese sowohl geschrieben als auch, daß aus diesen gelesen werden kann. Es ist aber keinesfalls so, daß Ihnen diese Anzahl an RAM-Speicherplätzen auch uneingeschränkt für Ihre Programme zur Verfügung steht.

Einen nicht unerheblichen Teil benötigt nämlich der Computer selbst, um seine Aufgaben erledigen zu können. Der größte Teil wird dabei durch den Bildspeicher "weggenommen", nämlich insgesamt 16 KByte.

Der CPC braucht aber noch mehr Speicherplatz für sich, damit er sich z.B. auch etwas merken kann. Darauf gehen wir später ebenfalls ausführlicher ein.

Damit Sie nun aber zunächst einmal ganz ins "Schleudern" kommen, setzen wir Ihnen noch weitere Einzelheiten zum "Verdauen" vor.

Der im CPC eingebaute Z-80-Prozessor hat einen 16 Bit breiten Adressbus. Die Kennzeichnung dieser Anschlüsse erfolgt meist mit den Bezeichnungen A0 bis A15.

Wenn Sie nun nachrechnen, dann werden Sie feststellen, daß damit insgesamt 65536 Bytes adressiert werden können, denn  $2^{16} = 64 \text{ K}$ . Ein Speicherbereich von 128 KByte, wie er beim CPC 6128 vorhanden ist, wäre deshalb aber Unsinn, wenn.., ja wenn die Entwickler von Computern nicht auch hier Lösungen parat hätten, um selbst mit einem etwas zu kleinen Prozessor Computersysteme zu entwickeln, die mehr können, als der jeweilige Prozessor eigentlich erlaubt.

Hinzu kommt nun noch, daß Ihr CPC ja nicht nur RAMs sondern auch noch ROMs besitzt. Letztere werden benötigt, damit der Computer auch etwas "tut".

Er braucht ja ein sogenanntes "Grundprogramm", welches es ihm überhaupt erst einmal ermöglicht, als Computer zu arbeiten. Sei

es nun ein eingebautes komplettes Betriebssystem oder nur ein sogenannter "Uriader" (Booter), der ein Betriebssystem einlädt und den Computer in einer bestimmten Weise konfiguriert.

Sie sehen, daß man, um den CPC zu verstehen, schon etwas mehr über ihn wissen muß.

## 2.4 Die mitgelieferte Intelligenz

Jeder CPC besitzt in seinem Inneren ein sogenanntes Firmware-ROM, in welchem sich sowohl das Betriebssystem als auch der Basic-Interpreter befinden. Dieses ROM hat eine Kapazität von 32 KByte. Durch die von Amstrad gewählte "Technik" wird dieses ROM in einen unteren und einen oberen Teil von je 16 KByte aufgeteilt. Diese Aufteilung ist aber nur eine elektronische Auswahl und ist nicht mechanisch "aufgeteilt" zu betrachten.

Der untere Teil enthält das Betriebssystem, welches im Bereich von &0000 bis &3FFF dem RAM-Speicher parallel liegt.

Der obere Teil enthält den Basic-Interpreter und liegt im Bereich von &C000 bis &FFFF und damit ebenfalls RAM-Speichern parallel. Die RAM-Speicherstellen dieses Adressbereiches werden normalerweise als Bildschirm-RAM verwendet.

Bei diesen Zahlenangaben weist das vorangestellte "&"-Zeichen darauf hin, daß es sich um eine hexadezimale Zahl handelt. Anstelle der Schreibweise

&C000

sind in Veröffentlichungen, also in Zeitschriften, Datenblättern Büchern usw., manchmal auch folgende Darstellungen zu finden:

C000h, HC0000 oder auch \$C000.

Alle diese Schreibweisen meinen das Gleiche, nämlich: Es handelt sich bei der angegebenen Zahl um eine Hexadezimalzahl. Korrekterweise müßte es eigentlich Sedezimalzahl heißen, aber es hat sich zwischenzeitlich so eingebürgert, immer nur von Hexadezimal-, oder noch kürzer, von Hexzahlen zu sprechen.

Bitte gewöhnen Sie sich

- von Anfang an -

an die hexadezimale Darstellungs-, Schreib- und Denkweise.

Sie werden es sicher in sehr kurzer Zeit erlernt und dann sehr große Vorteile dadurch haben, denn vieles wird dann übersichtlicher und auch schneller verständlich. Von fast gleich großer Wichtigkeit ist auch binäres Rechnen!

"Dezimaless Denken" in der Computertechnik ist nicht nur unübersichtlich, sondern wird auch in der Praxis, das heißt in Fachbeiträgen und Artikeln, vor allem wenn es um Maschinensprache geht,



kaum verwandt. Wenn Sie dieses Buch durcharbeiten, werden Sie sehr schnell bemerken, welche Vorteile Sie durch den Einsatz von Hexadezimalzahlen haben. Was läßt sich denn leichter merken:

&C000 oder 49152 ?

Damit Sie aber auch dies nun schneller überblicken, wollen wir auch hierüber eine kleine Einweisung geben.

## 2.5 Verschiedene Zahlensysteme

Wie Sie bestimmt wissen, hat sich unser "normales" Zahlensystem (mit der Basis 10) vor allem deshalb bewährt, weil wir zufällig zehn Finger besitzen.

Ein Abzählen von kleinen Mengen ist dadurch leicht möglich. Hätten wir eine andere Anzahl von Fingern, dann hätten wir höchstwahrscheinlich auch ein anderes Zahlensystem, vermutlich ebenfalls basierend auf der Fingeranzahl, sofern die Anzahl der Finger nicht zu hoch wäre.

Mit der gleichen Selbstverständlichkeit, mit der wir heute bei Berechnungen mit der Basis Zehn arbeiten, würden wir genauso in einem anderen Zahlensystem denken, rechnen usw., wenn wir es gelernt hätten.

Sie glauben es noch nicht?

Nun dann denken Sie bitte daran, daß nicht immer und überall im "Zehnersystem" gearbeitet wurde und wird. Zur Erinnerung nur ein paar Stichworte: Dutzend, Gros, Schock, Paar, Seidel, Pfund, Binar, Oktal, Dual, Uhrzeit, Datum usw.

Mit einem Teil dieser Zahlen-, Maß- oder Einheitensysteme gehen Sie genauso perfekt um, wie mit unserer Zehnerzählung, denken Sie nur an das Datum und die Uhrzeit.

Es gibt also keinen Grund, weshalb Sie sich nicht auch - innerhalb kürzester Zeit - an die Sedezimal- (Hexadezimal-) Zahlen gewöhnt haben sollten.

## 2.6 Das Sedezimalsystem

Im Gegensatz zum Dezimalsystem, bei dem immer von Null bis neun (= 10 Werte) gezählt, bevor die nächsthöhere Stelle (Wertigkeit) um eins erhöht wird, besitzt das hexadezimale Zahlensystem 16 Werte.

Da aber die Darstellung mit unseren normalen Dezimalziffern hierzu nicht mehr ausreicht, hat man zusätzliche "Ziffern" einführen müssen.

Hier eine kleine Gegenüberstellung:

Dezimal	Hexadezimal	Dezimal	Hexadezimal
0	0	8	8
1	1	9	9
2	2	10	A
3	3	11	B
4	4	12	C
5	5	13	D
6	6	14	E
7	7	15	F

Wie Sie sehen, geht die einziffrige Darstellung eines Zahlenwertes - über den Wert 9 hinaus - vom Buchstaben A bis zum Buchstaben F. Diese sind allerdings beim Hexadezimalsystem keine Buchstaben sondern Ziffern.

Doch wie wird dann weiter gezählt? Ganz genauso wie beim Dezimalsystem.

Die Dezimalzahl 16 ist also &10, die Zahl 17 entspricht &11, die Zahl 18 = &12 usw.

Wenn Sie nun eine Hexadezimalzahl tatsächlich einmal selbst in eine Dezimalzahl umrechnen wollen, ist dies prinzipiell eigentlich ganz einfach. Nur wer noch nicht gelernt hat, mit anderen Zahlensystemen Berechnungen bzw. Umrechnungen durchzuführen, wird dabei so seine kleinen Schwierigkeiten haben. Deshalb werden wir an ein paar Beispielen das Prinzip erläutern.

Nehmen wir zunächst einmal die Zahl &80. Hier brauchen Sie lediglich die erste Hexziffer mit 16 zu multiplizieren, schon haben Sie das Ergebnis. Schwieriger wird es aber schon, wenn wir beispielsweise die Zahlen &FE oder gar &C03F haben. Aber auch das ist ganz einfach, wenn man etwas übt. Die einzelnen Stellenwerte der Hexzahl müssen nämlich nur richtig miteinander "addiert" werden.

#### Beispiele:

$$\&FE = F \cdot 16 + E = 15 \cdot 16 + 14 = 254$$

$$\&C03F = C \cdot 16^3 + 0 \cdot 16^2 + 3 \cdot 16^1 + F \cdot 16^0 = 49215$$

$$\&FEAF = F \cdot 4096 + E \cdot 256 + A \cdot 16 + F = ?$$

So nun können Sie etwas knabbeln, bevor Sie weiterlesen. Die Darstellung beim Wert &C03F haben wir aber so gewählt, daß Sie die Lösung für jede Hexzahl finden müßten.

Wenn nicht, dann helfen wir Ihnen hier natürlich sofort weiter. Die Umrechnung einer Hexzahl in ihr Dezimaläquivalent erfolgt nach folgender Rechenregel (Algorithmus):

$\&WXYZ =$       W-mal 16 hoch (Anzahl der Hexziffern-1)  
                  plus X-mal 16 hoch (Anzahl der Hexziffern-2)  
                  plus Y-mal 16 hoch (Anzahl der Hexziffern-3)  
                  plus Z-mal 16 hoch (Anzahl der Hexziffern-4)

Nochmals anders dargestellt:

Dezimalwert =  $W \cdot 16^{(n-1)} + X \cdot 16^{(n-2)} + Y \cdot 16^{(n-2)}$  usw.  
                  (n            =Anzahl der Stellen)

Wer es noch anders betrachten will, kann auch so vorgehen, daß er - rechts beginnend - die einzelnen Stellen der Hexzahl mit folgenden Werten multipliziert:

1. Ziffer mit 1
2. Ziffer mit 16
3. Ziffer mit 256
4. Ziffer mit 4096

Wir hoffen, daß Ihnen eine dieser Methoden zusagt, und eine davon für Sie auch verständlich ist. Ansonsten müßten wir exakt mathematisch vorgehen, und dann wäre die Gefahr, daß ein Teil unserer Leser dieses Buch nun vorzeitig welegt, zu groß.

Selbst wenn Sie die ganze "Hexzahlengeschichte" nun noch nicht verstanden haben, verzweifeln Sie nicht, die Stadt Rom ist auch nicht an einem Tag erbaut worden. Versuchen Sie trotzdem, sich immer wieder an die "Hexadezimale Denkweise" zu gewöhnen, denn diese vereinfacht wirklich sehr vieles.

Auch der umgekehrte Rechnungsweg, nämlich die Umrechnung einer Dezimalzahl in eine Hexzahl ist selbstverständlich durchführbar. Bevor Sie sich nun aber quälen, gehen Sie doch einen einfacheren Weg, als "zu Fuß" rechnen. Sie brauchen diese ganzen Umrechnungen nämlich nicht selbst durchzuführen, wozu haben Sie denn einen CPC? Dieser wird die entsprechenden Umrechnungen ohne Mühe für Sie meistern.

## 2.7 Umrechnungen mit dem Schneider

Falls Sie dies nun anzweifeln, dann wissen Sie vielleicht schon etwas mehr über Ihren Schneider-Computer. Denn tatsächlich zeigt sich bei derartigen Umwandlungen bereits der erste kleine "bug", also ein Fehler im Betriebssystem der CPC's, und zwar im eingebauten Firmware-ROM (Basic-Interpreterteil).

Diesen Fehler haben alle CPC's. Er ist nicht weiter tragisch, nur sollte man wissen, daß er vorhanden ist, denn nur dann kann man sich verschiedene - durch diesen Fehler hervorgerufene - Effekte erklären und auch entsprechende "Gegenmaßnahmen" in einem Programm vorsehen.

Die Umrechnung einer Hexadezimalzahl in ihr dezimales Equivalent kann bei Ihrem CPC durch z.B. folgende Weise geschehen:

```
PRINT &Hexzahl
```

Also z.B.: 

```
PRINT &FF
```

Ihr CPC wird korrekterweise mit der Ausgabe der Zahl 255 antworten.

Das gleiche gilt beispielsweise auch für die nachfolgende kleine Aufgabe:

```
PRINT &7F2E
```

Auch diese Mathematikaufgabe wird Ihr CPC problemlos meistern.

Sobald Sie aber Werte größer als &7FFF umgerechnet haben wollen, dann werden Ihnen plötzlich, anstelle von positiven, negative (!) Zahlen als Ergebnis genannt. Ab der Zahl &8000 erhalten Sie - als Ausgabewerte - Zahlen mit negativen Vorzeichen.

Der Befehl `ABS(&8000)` hilft dann zwar bei dieser Zahl weiter und gibt den korrekten Wert mit 32768 aus, aber dann läuft alles wieder rückwärts.

Bitte geben Sie in Ihren Computer doch einmal folgendes ein:

```
? &7FFF, ABS(&7FFF)
? &8000, ABS(&8000)
? &8001, ABS(&8001)
```

(Anstelle des Befehlswortes `PRINT` können Sie wie in obigen Beispiel gezeigt also auch das Fragezeichen benutzen!)

Die Funktion `ABS` bewirkt eine vorzeichenlose Ausgabe, aber der Fehler oberhalb von &8000 wird dadurch nicht behoben.

Sie brauchen nun aber nicht lange herumzurätseln weshalb es so ist, akzeptieren Sie es einfach, der CPC verhält sich eben so.

Addieren Sie in diesem Falle einfach den Wert  $2^{16}$  hinzu. Sinnvollerweise sollte dies gleich bei der Eingabe geschehen. Denken Sie also bei Hexzahlen größer &7FFF immer daran, die Korrektur durchzuführen.

Beispiele:

```
PRINT &C03F + 2^16
```

```
PRINT &A033 + 2^16
```

```
PRINT &FEFF + 2^16
```

Und nun gleich zu einem unangenehmen Effekt dieses Fehlers. Die CPC's erlauben es, `FOR-NEXT`-Schleifen auch mit Hexzahlen zu programmieren.

Damit Sie gleich wissen was damit gemeint ist, hier ein kurzes Beispiel für eine FOR-NEXT-Schleife:

```
FOR i = 1 TO 255: PRINT i: NEXT i
```

Dieses kleine "Programm" macht nichts anderes, als die Zahlen von 1 bis 255 auf dem Bildschirm auszugeben. Schleifen können aber auch hexadezimal programmiert werden und sind eigentlich denen mit Dezimalzahlen gleichwertig. Auch obige Schleife kann hexadezimal programmiert werden und ist der obenstehenden gleichwertig, das heißt, sie bewirkt genau die gleichen Ausgaben. Obige Schleife also hexadezimal programmiert:

```
FOR i = 1 TO &FF: PRINT i: NEXT i
```

Beide Schleifen geben also auf dem Bildschirm die Zahlen von 1 bis 255 aus.

Auch wenn das Ziel höher gesetzt wird klappt dies meist ohne irgendwelchen Probleme. Nur wenn Sie die schon genannte Hexadezimalzahl &7FFF in der Schleife überschreiten, dann zeigt sich der Fehler. Damit Sie nicht allzu lange auf die Folter gespannt werden, geben Sie in Ihren CPC doch folgendes ein:

```
FOR i = &7FFE TO &8020: PRINT i: NEXT
```

Diese Eingabe können Sie sowohl im Direktmodus als auch in einem Programm versuchen. Das Ergebnis ist in jedem Falle das gleiche.

Nach sehr, sehr kurzer Zeit wird Ihr CPC ausgegeben:

READY

aber keine Spur einer Zahl, geschweige denn von 255, wie eigentlich zu erwarten gewesen wäre. Ihr CPC hat nämlich den Endwert als negative Zahl betrachtet und da dieser Wert niedriger ist als der Anfangswert, hat er sich so verhalten, als hätte er seine Aufgabe vor dem eigentlichen Beginn schon erledigt.

#### Fazit:

Eine derartig programmierte Schleife wird vom CPC nicht abgearbeitet. Abhilfe kann bei einem solchen Problem entweder die Angabe mit Dezimalwerten oder aber, wie schon bei der Berechnung aufgezeigt, die zusätzliche Addition von  $2^{16}$ , schaffen. Diese Addition kann natürlich auch von einer

IF kleiner Null Abfrage (If Wert < 0 then ....)

abhängig gemacht werden. In vielen verschiedenen Programmen dieses Buches können Sie nachsehen, was damit gemeint ist, falls Sie es nicht schon wissen.

Sie kennen nun diesen Effekt, aber auch ein schon erfahrener Programmierer, der seine gesamten Erfahrungen mit einem anderen Computer gesammelt hat, wird zunächst, wenn er sein Programm mit

einer derartigen Schleife versehen hat, erst einmal verblüfft sein, wie schnell diese auf dem CPC "abgearbeitet" ist. (Falls er es überhaupt sofort bemerkt.)

Da diese "negative Arbeitsweise" sowohl im Direktmodus, als auch in einem laufenden Programm auftritt, wird derjenige der - aufgrund seiner Programmiererfahrung - Programme aus dem linken Handgelenk schüttelt, zunächst einmal verwundert über sein eigenes - und vielleicht schon so oft auf anderen Computern einwandfrei ablaufendes - Basic-Programm nachdenken.

Anzumerken ist, daß dieser Effekt nur dann auftritt, wenn unterhalb von &8000 "gestartet" wird.

Folgende Schleife wird also wieder korrekt abgearbeitet:

```
FOR i = &8002 TO &9FFF: .....
```

Damit haben Sie nun schon eine Eigenart Ihres CPC kennengelernt. Aber nun wollen wir uns doch weiter erst einmal ganz allgemein mit dem CPC beschäftigen.

Sie wissen bereits, daß der "Schneider" RAM- und ROM-Speicher besitzt. Aber natürlich sind auch andere Bauteile nötig; diese sollen im nächsten Abschnitt behandelt werden.

### 3. Hardware, die „anfaßbare Ware“

Als Hardware werden bei Computersystemen alle Bauteile bezeichnet. Betrachten Sie deshalb der Einfachheit halber alles was anfaßbar ist, bzw. eine räumliche Ausdehnung hat, als Hardware und alles was Sie nicht anfassen können als Software (z.B. Programme). Bei einem Firmware-ROM ist zwar das Speichermedium als solches anfaßbar, die enthaltenen Programme aber nicht. Das ROM selbst ist also Hardware, der Inhalt Software.

Wir wollen Sie einerseits zwar nicht allzusehr in die Hardware-Details einweisen, aber zum allgemeinen Verständnis ein paar davon kurz vorstellen und auch einige Kurzhinweise geben, bzw. die Anschlußbelegungen und Arbeitsweisen aufzeigen. Nur wie schon bemerkt sehr tief gehen wir nicht, denn dies würde den Rahmen dieses Buches sprengen. Andererseits sind diese Informationen vielleicht für diejenigen, die daran interessiert sind, die extern ansteckbare Betriebssystem-Erweiterungsplatine für den CPC 464 nachzubauen und Probleme damit haben, evtl. der Rettungsanker um doch damit zurecht zu kommen.

Wer sich weniger für die Hardware interessiert sollte diesen Abschnitt zwar nicht auslassen, braucht ihn aber nur zu überfliegen.

Neben dem Prozessor Z80 sind an hochintegrierten Bausteinen, die in Ihrem CPC enthalten sind, vor allem die nachfolgenden erwähnenswert:

Der Sound-Generator	AY -3-8912
Der Parallel-Input-/Output-Baustein (PIO)	MSL8255AP-5
Der VDU-Controller	HD6845SP
Das GATE-ARRAY	HS83130,
	HS83170 oder 20RA043
Das Firmware-ROM	TMM 23256

Betrachten wir als nächstes einmal die Hauptfunktionen und Anschlußbelegungen der bisher genannten integrierten Schaltkreise.



### 3.1 Der Prozessor Z80A

Dieser Baustein ist das Herz des Systemes. Bei den Anschlußbezeichnungen machen wir der Einfachheit halber keine Unterscheidung zwischen negiertem und normalen Signal-Bezeichnungen, falls aber erforderlich geben wir die Erklärung im jeweiligen Text.

Mit Ausnahme der Versorgungsspannungsanschlüsse, kann man sich den Rest in drei Busse eingeteilt denken.

In einen 16 Bit breiten Adressbus (unidirektional),  
einen 8 Bit breiten Datenbus (bidirektional)  
und einen Kontrollbus.

Die Adressleitungen A0 bis A15 (Pin 1 bis 5 und 30 bis 40) stellen den Adressbus dar. Sie dienen dazu, eine Speicherstelle "anzuwählen". Der Adressbereich umfasst 65536 Speicherplätze ( $2^{16}$ ). Da im CPC aber mehr Adressen angesprochen werden müssen, haben sich die Entwickler des CPCs einiges einfallen lassen. Es wird mit sogenanntem "banking" gearbeitet. Das bedeutet, daß 16-KByte-Blöcke ein- und ausgeblendet werden.

Die Datenleitungen D0 bis D7 (Pin 7 bis 10 und 12 bis 15) stellen den Datenbus dar. Auf diesen bidirektionalen Leitungen läuft der gesamte Datentransfer ab.

Die Leitung an Pin 6 des Prozessors ist der Taktsignaleingang. Der Takt kann zwischen 0 Hertz und der vom Hersteller genannten Maximalfrequenz liegen. Beim CPC wird mit 4 MHz getaktet.

Die Versorgungsspannung von 5 Volt liegt an Pin 11 (VCC) an. Die Stromaufnahme des Prozessors liegt unter 200 mA.

Am Pin 16 befindet sich der INT- (Interrupt Request)-Eingang. Ein Low-Signal an diesem Eingang bewirkt, wenn diese Art der Unterbrechung vom Programm freigegeben wurde, je nach gewähltem Interruptmodus, unterschiedliche Auswirkungen. Bei diesem Eingangssignal muß es sich um ein statisches Signal handeln, das bis zur Erkennung der Interruptanforderung durch den Prozessor, anstehen muß.

Der NMI (Non Maskable Interrupt) an Pin 17 ist ein weiterer Eingang für eine Prozessorunterbrechung. Durch ein High - Low Signal, also durch die Flanke eines Signalwechsels von High nach Low, kann der Prozessor in seinem laufenden Programm unterbrochen werden. CPC-intern wird dieser Anschluß nicht genutzt, steht aber am herausgeführten Erweiterungsanschluß zur Verfügung. Bei einer externen Unterbrechung wird der Programmzähler (PC = Programm Counter) mit den Speicherinhalten von &66 und &67 geladen und an dieser Adresse das Programm fortgesetzt. Wie der Name schon sagt, ist diese Unterbrechungsmöglichkeit nicht "maskierbar", sondern wird immer ausgeführt. Im Gegensatz zum INT-Eingang führt hier also ein Signalwechsel zur Unterbrechung.

Der Halt-Ausgang an Pin 18 zeigt durch ein Low-Signal an, daß der Prozessor den Maschinensprachebefehl "Halt" ausgeführt hat. Der Prozessor führt nur noch NOPs (No Operations) aus, um den Refresh

der dynamischen RAMs durchzuführen, aber ansonsten "schläft" er und nur ein Interrupt kann ihn wieder "wecken" und zu weiteren Aktivitäten bewegen.

Durch ein Low am Ausgang Memory Request (MREQ) signalisiert der Z80, daß ein Lese- oder Schreibzugriff auf eine Speicherstelle erfolgt und die Adressbussdaten gültig sind. Dieser Anschluß liegt an Pin 19.

Der Ausgang IORQ (Input Output Request) an Pin 20 hingegen zeigt durch Low-Signal an, daß eine Portadresse gültig ist, auf die ein Zugriff erfolgt.

Das Ausgangssignal RD (Read) ist ebenfalls "lowaktiv" und wird dann low, wenn der Prozessor Daten aus einer Port- oder Speicheradresse lesen will. Die logische Verknüpfung mit IORQ und MREQ macht die Unterscheidung, ob der Lesevorgang für einen Port oder für den Speicher gilt, möglich. Pin 21 ist der zuständige Signalanschluß.

Das Gegenstück hierzu liegt an Pin 22, der Anschluß WR (Write). Es gilt das gleiche wie beim Anschluß RD, nur daß sich alles auf Schreiboperationen bezieht.

Die beiden Anschlüsse BUSAK (Busaknowledge) am Pin 23 und BUSRQ (Bus-Request) am Pin 25 müssen miteinander betrachtet werden. Das Ausgangssignal BUSAK ist in Korrespondenz zum BUSREQ.

Ein Low-Signal an BUSAK zeigt einem evtl. angeschlossenen zweiten Prozessor oder einem DMA-Controller (Direkt Memory Access = direkter Speicherzugriff) an, daß sich alle Bus- und Steuerausgänge im hochohmigen Zustand befinden, denn nur dann ist ein korrekter Zugriff auf den Speicher und eine schnelle Datenübertragung unter Umgehung des Prozessors möglich.

Die Ansteuerung geschieht über den Eingang BUSREQ, der dann, wenn dort ein Low-Signal anliegt, nach der Abarbeitung des laufenden Befehles, die schon genannten Anschlüsse hochohmig werden läßt. Elektronisch betrachtet wirkt dies, als wären diese Anschlüsse nicht vorhanden.

Diese beiden Signalanschlüsse haben zwar im CPC intern nichts zu erledigen, sind aber ebenfalls auf den Erweiterungsanschluß herangeführt.

Der Anschluß WAIT oder auch READY, an Pin 24, soll es einem externen, langsamen "Gerät" ermöglichen, den Prozessor zu bremsen, falls dessen Geschwindigkeit mit der Prozessorgeschwindigkeit nicht mithalten kann. Es handelt sich also um einen Eingang.

Der RESET-Eingang an Pin 26 bewirkt bei einem Low-Signal, daß der Programmzähler (PC) mit dem Wert &0000 geladen wird. Dabei wird der Interruptmodus 0 eingeschaltet und alle Interrupts gesperrt. Sobald der Eingang wieder High-Signal bekommt, beginnt der Prozessor mit der Abarbeitung des Programmes. Die Startadresse steht im Programmzähler. Dies bedeutet, daß der Prozessor einen Neustart durchführt.

Der Steuer-Ausgang M1 (Machine Cycle One) an Pin 27 zeigt durch Low-Signal an, daß der Prozessor damit beschäftigt ist, einen

Operationscode vom Datenbus zu lesen.

Da es sich bei den in den CPCs eingesetzten RAM-Bausteinen um dynamische Typen (4164-2) handelt, ist eine immer wiederkehrende Auffrischung ihres "Wissens" zwingend erforderlich. Dies geschieht durch den Prozessoranschluß RFSH (Refresh) an Pin 28. Der Prozessor benötigt ja nur zu bestimmten Zeiten den Adress- und Datenbus und in der Zwischenzeit wird durch diesen Anschluß deklariert, daß auf den unteren 8 Adressleitungen eine gültige Refresh-Adresse liegt.

Der Pin 29 ist der Anschluß GND, also der Minus-, Null-, oder Masse-Anschluß der Versorgungsspannung.

Damit sind alle Anschlüsse des Prozessors abgehandelt. Die PINDOUTS der Bausteine finden Sie im Anhang.

### 3.2 Der Soundgenerator (AY-3-8912)

Dieser von General Instruments stammende Schaltkreis ist ein programmierbarer Tongenerator. Ursprünglich eigentlich für Telespiele entwickelt, darf er also auch in den CPCs zeigen, was in ihm steckt. Für guten und vollen Sound sollte allerdings an den CPC ein Stereoverstärker angeschlossen werden.

Der Soundgenerator besitzt einen bidirektionalen Parallelport mit 8 Bit und unter anderem drei voneinander unabhängige Analogausgänge. Jeder dieser Ausgänge kann unabhängig von den anderen in Frequenz und Lautstärke programmiert werden.

Die Programmierung dieses Klangerzeugers muß über den 8255-P10 erfolgen. Leider ist das nicht ganz einfach, da die Registerauswahl nur über zwei Anschlüsse erfolgen kann.

Diese Anschlüsse haben im Schaltbild folgende Bezeichnungen: BD1R und BC1. Der Anschluß BC2, ein weiterer Steueranschluß, kann nicht genutzt werden, da dieser auf +5 Volt gelegt ist. Mit den beiden zur Verfügung stehenden Anschlüssen können folgenden Wirkungen erzielt werden:

BC1	BD1R	
0	0	keine Funktion
0	1	schreiben
1	0	lesen
1	1	Adresse "latchen"

Dieser Baustein dient aber nicht nur zur reinen Tonerzeugung, sondern übernimmt in Verbindung mit dem Schnittstellenbaustein (PIO) und einem Decoder (74LS145) auch die Abfrage der Tastatur und der Joysticks.

Fünfzehn der insgesamt sechzehn Register dieses Geräuschproduzenten können benutzt werden und sorgen für ein gutes Klangbild. Die Lautstärken können ebenso programmiert werden, wie beispielsweise Rauschen oder auch Musik.

Da wir in diesem Buch nicht näher auf die Tonerzeugung eingehen, bzw. auch keine Programme für Musik geschrieben haben, verzichten wir an dieser Stelle auf eine weitere Beschreibung dieses Bausteines, da eine detaillierte Beschreibung seiner Funktionen, Eigenschaften und Möglichkeiten ein eigenes Buch füllen würden.

### 3.3 Der Schnittstellenbaustein (PIO 8255)

Dieses programmierbare Parallelschnittstellen-Interface besitzt drei 8-Bit-Ports, die für verschiedenste Funktionen zuständig sind.

Dies betrifft beispielsweise die Tastaturabfrage, das Ein-/Aus-schalten des Kassettenrekorder-Motors, die Festlegung welche Einschaltmeldung sich auf Ihrem Bildschirm zeigt (durch Brücken programmierbar), das Lesen und Schreiben in Verbindung mit der Kasette usw.

Wenn man sich das Datenblatt dieses Bausteines näher betrachtet, kann man mehrere "Blöcke" zur Verbindung mit anderen Bauteilen erkennen.

Einen bidirektionalen Datenbus,  
einen Kontrollbus, und  
vier Ein-/Ausgabeports.

Die Pins 27 bis 34 (D7 bis D0) sind Dateneingangsleitungen zum Systembus. Hierüber erfolgt der Datenaustausch mit dem gesamten System.

Der Kontrollbus besteht aus den Eingängen

RD (neg)	=READ (Pin 5)
WR (neg)	=WRITE (Pin 36)
CS (neg)	=CHIP SELECT (Pin 6)
A0 und A1	=(Pin 9,8) Adresseingänge, interne Registerauswahl
und	
RESET (neg)	=(Pin 35)

Die Ein-/Ausgabeleitungen sind in drei Ports aufgeteilt.

PA0 bis PA7	=I/O-Leitungen des Ports A (Pin 4 bis 1 und 40 bis 37).
PB0 bis PB7	=I/O-Leitungen des Ports B (Pin 18 bis 25)
PC0 bis PC7	=I/O-Leitungen der Ports C (Pin 14 bis 17, Pin 13 bis 10)

Der Port C kann dabei in zwei 4-Bit-Gruppen aufgeteilt werden, wovon beim CPC auch Gebrauch gemacht wird.

Auch bei diesem Baustein wollen wir nicht weiter in die Tiefe gehen. Wer sich näher damit beschäftigen will, sollte sich entweder die technischen Unterlagen oder beispielsweise das Buch "Z 80 Anwendungen" (SYBEX-Verlag) besorgen. In letzterem ist der Benutzung des 8255 (mit dem Z80) ein ganzes Kapitel gewidmet.

### 3.4 Der VDU-Controller (6845)

Dieser auch als CRTC (Cathode Ray Tube Controller) bezeichnete Baustein ist der "Manager" der Bilderzeugung und produziert alle für den CPC erforderlichen Bildsignale. Es ist also der Steuerbaustein für die Bildsignale. Allerdings wird das Video-Signal nicht durch ihn, sondern vom Gate-Array erzeugt.

Da dieser Baustein in weiten Grenzen programmierbar ist, stellt er für Spezialisten ein ideales Betätigungsfeld für eigene Bildschirmgestaltungen dar. Er besitzt einen Light-Pen-Eingang, erlaubt einen programmierbaren Cursor und bietet auch sonst noch Möglichkeiten, auf die aber hier nicht näher eingegangen werden soll.

#### Die Anschlüsse des CRTC

VCC (Pin 20) ist der Versorgungsspannungsanschluß (+5 Volt).

GND (Pin 1) ist der Ground-Anschluß, auch Minus oder Null genannt.

RESET (Pin 2) ist der Resetanschluß, der dafür sorgt, daß der CRTC in einen definierten Zustand gebracht werden kann. Über diesen Anschluß wird auch die "eingefrorene" Adresse (siehe Anschluß L PEN) wieder "aufgetaut".

D0 bis D7 (Pin 26 bis 33) sind die Datenleitungsanschlüsse.

EN (Pin 23) = Enable-Anschluß, an dem das invertierte, über verschiedene Gatter aus den Signalen RD, WR und IORQ des Z80 gewonnene Signal, ansteht.

L PEN = LIGHT PEN (Pin 3) der Anschluß für das Triggersignal eines Lightpens. Durch einen Signalwechsel von Low nach High (Strobe-Impuls) kann eine, gerade ausgelesene, Bildschirm-Adresse "eingefroren" werden, um die Position eines angeschlossenen Lichtgriffels zu ermitteln.

DISPEN = DISPLAY ENABLE (Pin 18) Signal-Ausgang um dem Gate Array mitteilen zu können, daß Daten aus dem Bildschirmspeicher ausgelesen werden müssen.

HSYNC = HORIZONTAL SYNCHRONISATION (Pin 39) Über diesen Anschluß wird am Ende einer Rasterzeile ein Signal an das Gate Array gegeben, um die Zeilensynchronisation zu gewährleisten.

VSYSN = VERTICAL SYNCHRONISATION (Pin 40) wie HSYNC, aber vertikal. Dieses Signal wird auch dem Anschluß B0 des PIO-Bausteines zugeführt.

CCLK = CRTC CLOCK (Pin 21) Das NMWE-Signal des Gate-Array synchronisiert über diesen Eingang die Arbeit des CRTC.

MA0 bis MA10 = MEMORY-ADRESSES (Pins 4,5,6,7,8,9,10,11,15,16,17)  
Über einen Teil dieser Anschlüsse werden die 4 zu 1 Datenmulti-  
plexer (74 LS 153) angesteuert, die ihrerseits die Adressen des  
Bildschirmspeichers ansprechen. Nicht angeschlossen sind die Pins  
14 und 15 (MA10 und MA11) wobei Pin 15 zu einem sogenannten Pad  
führt. Vermutlich ist dies nur ein Meßpunkt für die Fertigungs-  
kontrolle.

RA0 bis RA4 = RASTER-ADRESSES (Pins 38,37,36,35,34) üblicherweise  
wählen diese Leitungen die Rasterzellen eines Charakter-Genera-  
tor-ROMs an. (Charaktergenerator = Zeichengenerator)  
Da der CPC kein Charakter-Generator-ROM besitzt, wird über diese  
Leitungen in Verbindung mit den schon genannten Multiplexern der  
Bildschirmspeicher adressiert. Die Ansteuerung wird nur durch die  
Anschlüsse RA0 bis RA2 bewirkt. Die restlichen RA-Pins sind nicht  
angeschlossen.

R/W = READ/WRITE-Anschluß (Pin 22), durch das dort anliegende  
Signal wird bestimmt, ob Daten vom oder zum CRTC übertragen wer-  
den. Dieser Anschluß wird über die Adressenauswahl (Adressleitung  
A9 des Z80-Busses) gesteuert!

RS = REGISTER SELECT (Pin 24) Dieser Eingang wird vom Adressbus  
(Adressleitung A8) angesteuert und wählt aus, ob Daten- oder Ad-  
reßregister des CRTC angesprochen werden.

### 3.5 Das Gate-Array

Haben wir beim Prozessor gesagt, daß dieser das Herz des CPC darstellt, dann müssen wir diesem Baustein die Bezeichnung "Gehirn" zugestehen, denn dieser integrierte Schaltkreis, der nicht handelsüblich ist, sondern speziell für die CPC's entwickelt wurde, trägt sehr viel zur Koordination im CPC bei.

Obwohl in den CPC's unterschiedliche Typen eingesetzt sind (auch mit unterschiedlicher Anschluß-Belegung) haben alle die gleiche Aufgabe, nämlich die Steuerung und Koordination des Gesamtsystems. (Es gibt Typen mit und ohne Kühlblech. Derzeit sind uns drei verschiedene Typen bekannt.)

Da es sich um ein sogenanntes Customer-IC handelt, dessen eigentlicher Entwickler Amstrad (oder Locomotive Software Ltd.) ist, gibt es leider nur sehr wenige Informationen hierüber. Amstrad will damit vermutlich verhindern, daß zuviel "Know-how" bekannt wird.

Obwohl also viele Einzelheiten noch unbekannt sind, die Aufgaben des Gate-Arrays sind doch ziemlich bekannt.

#### Der Aufgabenumfang:

- Erzeugung aller benötigten Taktfrequenzen
- Steuerung der Zugriffe auf die RAM-Speicherstellen,
- Erzeugung der Betriebssignale für die RAMs
- Zu- und Abschalten der ROMs
- Erzeugung des Interruptimpulses
- Erzeugung aller Video-Signale
- Speicherung der INKs
- Erzeugung aller RGB-Informationen für einen Farbmonitor
- Steuerung des Bildschirmmodus

#### Anschlußbelegung und Steuersignale des Gate Array

Ja, nun haben wir in der Zwischenüberschrift fast etwas zuviel versprochen, denn wie wir schon mitteilten, sind uns bereits drei verschiedene Gate-Arrays bekannt und diese haben auch noch verschiedene Anschlußbelegungen.

Wir nehmen einfach den Typ 40010, der im CPC 6128 eingesetzt ist und nennen in der Klammer hinter der Pin-Bezeichnung die Anschlußnummer der anderen Gate-Array-Belegung. Der Buchstabe "N" vor den Anschlußbezeichnungen bedeutet, daß das Signal negativ aktiv ist!

#### Pin 14 (1) NCPU

Dieser Anschluß hat mehrere Funktionen. Das Ausgangssignal dieses Anschlusses bestimmt mit, ob der Prozessor oder der CRTC das RAM ansprechen kann, bei einem Low werden über (die bereits beim CRTC genannten) Multiplexer die Prozessor-Adressleitungen zu den RAMs durchgeschaltet. Weiterhin steht an diesem Pin ein phasenverschobenes 1 MHz-Signal an, welches für den Soundchip als Taktsignal dient.



Pin 22 (2) READY

Dieses Ausgangssignal dient in Verbindung mit dem Signal RAMRD von Pin 13 dazu, ein 8-Bit-D-Latch (74LS373) derart anzusteuern, daß Informationen auf dem Prozessor-Datenbus während der WAIT-Zyklen erhalten bleiben. Durch ein High des READY-Signales hält das 8-Bit-D-Latch die Daten auf dem Ausgang. Es verhält sich also wie ein Speicher. Wird das READY-Signal low, dann arbeitet das Latch als einfacher Puffer.

Pin 16 (3) NCAS

Dieser Ausgang steuert die RAM-Bausteine an. CAS = Column Address Select, also Spalten-Adressen-Auswahl.

Pin 23 (4) N244EN

Die Namensbezeichnung dieses Anschlusses zeigt klar, daß das Gate-Array ein "customer design" ist. Dieser Name rührt daher, daß über diese Leitung ein Daten-Buffer mit der Typenbezeichnung 74LS244 "enabled" wird (enable = eingeschaltet).

Pin 33 (5) NMWE

Durch die Überlappung von RAM, ROM und auch den Ports in verschiedenen Adressbereichen, muß im CPC gewährleistet werden, daß Schreibbefehle auch immer an den richtigen "Empfänger" gehen. Damit der Prozessor aber immer in's RAM schreibt und nicht einen sinnlosen Versuch macht, in's ROM zu schreiben, wird über diesen Anschluß ein Signal ausgegeben, welches prinzipiell das RAM auswählt. (MWE = Memory Write Enable). Diese korrekte "Ansprache" wird in Zusammenarbeit mit weiteren Adressleitungen des Gate-Array bewirkt.

Pin 31 (6) NCASADD

Das Signal an diesem Ausgang steuert 4 zu 1-Datenselektoren/Multiplexer an, welche wiederum in Verbindung mit anderen Signalen die Column Adresse der RAMs ansteuern.

Pin 34 (7) NRAS

Dieses Ausgangssignal ist ebenfalls am zeitlichen Ablauf der RAM-Ansteuerung beteiligt. (RAS = Row Address Select, also die "Reihe" der angesprochenen Adresse.)

Pin 24 (8) CK16

Dies ist der Grundtakt-Eingang, vom - Takt bestimmenden - Frequenzgenerator (Taktgeber). Die Taktfrequenz beträgt 16 MHz. Aus diesem Signal werden alle im CPC benötigten Taktfrequenzen abgeleitet bzw. erzeugt. Das frequenzbestimmende Element des Taktgebers ist ein Quarz. Die Schaltung zur Erzeugung der Taktfrequenz ist in den CPCs unterschiedlich.

Pin 6,25 (33,9) VDD1 und VDD2

Dies sind Versorgungsspannungs- bzw. Betriebsspannungsanschlüsse (+5 Volt) dieses Bausteines.

Pin 15,26,36 (35,22,38) VSS1 bis 3

Dies sind die negativen Versorgungsspannungs-Anschlüsse.

Pin 32 (10) NINTERRUPT

Wie der Name schon sagt, ist dies der Ausgang für das Interrupt-Signal.

Pin 5 (11) NSYNC

Ausgang für das Synchronisationssignal eines Monitors.

Pin 27 (12) NROMEN

Auch hier deutet der Name schon die Funktion an. Mit dem Signal auf dieser Leitung werden die ROMs ein- und ausgeschaltet. Dieses Signal bestimmt prinzipiell, ob ROMs eingeschaltet sind und wird üblicherweise an den Anschluß-Pin 20 von ROMs oder EPROMs geführt. Da die ROMs und EPROMs aber auch über ihren Pin 22 noch korrekt angesteuert werden müssen, wird durch das Signal ROMEN nur eine "Ansprechbedingung" erfüllt. Ein weiterer ganz wichtiger Punkt ist die Tatsache, daß das Firmware-ROM getaktet werden muß, um ausgelesen werden zu können.  
(Siehe Hinweis bei der Beschreibung des TMM 23256!)

Pin 29 (13) NRAMRD

Dieses Signal bewirkt, daß (auch in Abhängigkeit vom Bus-Signal "RAM DIS") das schon erwähnte 8-Bit-D-Latch entweder als Buffer oder als "Speicher" arbeitet.

Pin 11 (14) HSYNC

Eingang für das Horizontalsynchron-Signal, das vom CRTC kommt.

Pin 13 (15) VSYNC

Eingang für das Vertikalsynchron-Signal, ebenfalls vom CRTC.

Pin 18 (16) NIORQ

Bei einem erkannten Interrupt werden die Prozessorsignale IORQ und M1 gleichzeitig auf Low gesetzt. Dieser Eingang dient mit dazu, dies zu erkennen.

Pin 20 (17) NM1

Dient zum Erkennen eines Interruptes.

Pin 17 (18) NMREQ

Dient zur Generierung der Signale NRAMRD und NROMEN.

Pin 21 (19) NRD

Dient ebenfalls zur Generierung der Signale NRAMRD und NROMEN.

Pin 28 (20) A 15

Pin 30 (21) A 14

Diese beiden Anschlüsse ermöglichen das Bankswitching von RAM und ROM.

Pin 9 (23) DISPEN

Wird zur Erzeugung des Video-Signales benötigt.

Pin 35, 37, 38, 39, 40, 1, 2, 3 (24 bis 31) D0 bis D7

Die Datenleitungsanschlüsse.

Pin 8 (32) B

Das Blau-Signal für einen RGB-Monitor.

Pin 10 (34) G

Das Grün-Signal.

Pin 12 (36) R

Das Rot-Signal

Pin 7 (37) RESET

Der Reset-Eingang, wie üblich zum Rücksetzen.

Pin 19 (39) NPhi

Ausgang der 4 MHz für den Prozessortakt.

Pin 4 (40) CCLK

Ausgang 1 MHz-Takt für den CRTC.

### 3.6 Das Firmware-ROM (TMM 23256)

Bei diesem Baustein handelt es sich um ein 256 KBit-ROM der Firma Toshiba Mos Memory Products, genauer gesagt um ein N-Channel ROM mit 32 K Word \* 8 Bit.

Normalerweise wäre es nicht in diese Aufstellung der wichtigsten Bausteine des CPC aufgenommen worden, aber aus einem ganz besonderen Grund geschah dies doch.

Dieses ROM hat nämlich integrierte Adresslatches und dies zu wissen ist enorm wichtig, wenn man dieses ROM zum Beispiel mittels eines Eprommers auslesen will.

Sinn eines derartigen Auslesevorganges könnte beispielsweise sein, Änderungen am Betriebssystem durchzuführen um dann ein EPROM zu schießen und dieses an die Stelle des ROMs treten zu lassen.

EPROMs des Types 27256 sind pincompatibel zu diesem ROM. Dies bedeutet, alle Anschlüsse stimmen überein und das Original-ROM kann direkt gegen ein EPROM getauscht werden. Lediglich der Pin 1, der bei einem EPROM auf VCC gelegt werden soll, ist beim ROM "N.C" (Not connectet) hat also keine Verbindung zum Chip selbst.

Bevor sich jemand aber ein EPROM schießen kann, hat er erst einmal eine kleine Barriere zu überwinden, wenn das ROM in einem separaten Eprommer ausgelesen werden soll. Dieses Auslesen des ROMs wird nämlich mit den üblichen preisgünstigen "Schießgeräten" nicht immer direkt klappen.

Dies geht nur, wenn durch den "Ausleser" am Anschluß-Pin 20 ein Taktsignal angelegt wird. Durch eine negative Flanke, an diesem Anschluß, wird die an den Adressleitungen anstehende Adresse zu den internen Adreßdekodern durchgeschaltet und erst jetzt, kann der Inhalt der Adresse ausgelesen werden, wenn am Pin 20 und 22 negatives Potential ansteht. Das ist der ganze Dreh beim Auslesen.

Damit haben wir Ihnen die wichtigsten Bausteine Ihres CPC vorgestellt und auch deren Funktionsweise grob angesprochen. Wenn in den folgenden Kapiteln also diese IC's genannt werden, dann wissen Sie ganz grob um was es sich handelt.

### 3.7 Zusätzliche Bausteine für die Diskettenstation

Falls Sie einen CPC mit Diskettenstation besitzen, kommen noch ein paar IC's hinzu. Auch diese wollen wir Ihnen nun nennen und kurz vorstellen.

Der UPD 765 AC ist der eigentliche Floppy-Disk-Controller. Der FDC 9216 bzw. 9229 (Floppyinterface 464) ist der Datenseparator.

Sehr interessant - für eigene Programme - ist vor allem das Diskettenbetriebssystem-ROM.

#### 3.7.1 Der UPD 765

Ist ein Floppy-Disk-Controller (FDC) der es normalerweise ermöglichen würde, bis zu vier (!) IBM-kompatible, doppelseitige (!) und mit doppelter Aufzeichnungsdichte arbeitende, Laufwerke zu steuern.

Beim CPC aber wird dieser Controller in Verbindung mit den Laufwerken mit "gebremstem Schaum" betrieben. Es ist nur die Steuerung von zwei einseitig arbeitenden Laufwerken vorgesehen.

Da in diesem Buch keine Artikel über Hardware-Änderungen an den Diskettenstationen enthalten sind, wollen wir es deshalb bei dieser Anmerkung belassen und auch die Funktionsweise des Datenseparators nicht näher beschreiben.

#### 3.7.2 Das Floppy-ROM

In diesem 16 KB-ROM stecken viele, auch für den Nur-Anwender interessante Einsprungsadressen, auf die bei einigen Programmen dieses Buches "zurückgegriffen" wird.

In diesem ROM belegt das Disketten-Betriebssystem AMSDOS (Amstrad Disc Operating System) circa 8 KByte. Im restlichen Teil dieses Nur-Lese-Speichers befindet sich ein Teil der Programmiersprache DR LOGO, welche bei Diskettensystemen mitgeliefert wird.

Glücklicherweise stimmen - im Gegensatz zu den Firmware-ROMs - bei allen drei CPCs diese Disketten-ROMs überein.

Wenn Sie aber eine "fremde" Diskettenstation mit eigenem Controllerbaustein haben, dann kann das dort eingebaute Disketten-ROM keinesfalls mit dem CPC-Disk-ROM übereinstimmen, denn Amstrad hat darauf das Copyright und hat dieses bisher noch nicht freigegeben.

Dies bedeutet, daß Programme, die direkt mit den dort implementierten Programmroutinen arbeiten sollen, diese nicht direkt anspringen dürfen, denn sonst können diese Programme nur auf dem System arbeiten, für das sie geschrieben sind.

Wenn aber bei "fremden" Diskettencontroller-ROMs bestimmte Bedingungen eingehalten werden, ist es nicht schwierig Programme so zu schreiben, daß sie mit allen Controllern laufen. Wir haben uns darum bemüht, alle Diskettenroutinen so zu schreiben, daß Sie bei

- meist nur selten erforderlichen - kleinen Änderungen auch zum Beispiel mit den weit verbreiteten Diskettenstationen der Firma Vortex eingesetzt werden können.

Mit Kenntnissen über den Inhalt dieser Firmware können Sie einiges anfangen. Dies besprechen wir dann aber immer getrennt in den entsprechenden Kapiteln und bei den Programmen.

Damit haben Sie nun einen groben Überblick über die Hardwareausstattung Ihres CPC, wer sich noch näher mit der Hardware beschäftigen will, sollte sich von der Firma Schneider oder von seinem Computershop den Schaltplan seines CPCs besorgen.

Wir fahren nun mit weiteren Grundlageninformationen fort.

Als nächstes beschäftigen wir uns etwas intensiver mit dem RAM-Speicher und seinen vielen Informationen, die er uns geben kann.

## 4. Wichtige Details

### 4.1 Der RAM-Speicherbereich

Während das vorhergehende Kapitel vor allem dazu diente, Ihnen einen - nur groben - Überblick über Ihren CPC zu verschaffen, geht es nun schon weiter in die Feinheiten.

Wie bereits vermerkt, stehen bei allen CPC's als direkt ansprechbare, bzw. vom Prozessor adressierbare RAM-Speicherkapazität insgesamt 64 KByte zur Verfügung.

Subtrahiert man den für den Bildspeicher benötigten Speicherplatz von dieser Anzahl und vergleicht dann das Ergebnis mit der Ausgabemeldung des CPC's nach Eingabe von `PRINT FRE(0)`, oder auch mit der von `PRINT HIMEM`, so kann man feststellen, daß diese Werte nicht übereinstimmen.

Der Grund ist Ihnen schnell genannt. Der CPC braucht auch noch Speicherstellen, in denen er Werte zwischenspeichern kann, um sie sich zu merken oder zu verarbeiten. Aber auch sogenannte Sprungleisten, über die Betriebssystem-Routinen in einem der ROMs angesprungen werden können, nehmen Platz weg. Last not least braucht auch noch die Diskettenstation RAM-Speicher. Letzteres bemerken CPC 464-Besitzer besonders schmerzlich, weil evtl. Kassettenprogramme in Verbindung mit der Diskettenstation nicht mehr ablauf-fähig sind.

#### Von Basic aus betrachtet

Der Basic-Beginn liegt nicht etwa - wie vielleicht anzunehmen wäre - bei der Speicherstelle Null, sondern bei &170 (dezimal 368). Dies heißt, daß ein Basic-Programm, welches Sie eintippen oder einladen auch dort beginnt. Das erste Zeichen eines Basic-Programmes steht also in dieser ersten Speicherstelle. Auf die genaue "Ablage" gehen wir später noch ein.

Die ersten 367 Bytes des CPC-RAM-Speichers stehen Ihnen also normalerweise nicht zur Verfügung.

Fragen Sie nach dem Einschalten Ihren Computer doch einmal, wieviele RAM-Speicherstellen er Ihnen für ein Programm "zuteilt".

Dies geschieht durch

```
PRINT FRE(0)
```

und durch Betätigen der Enter- bzw. Return-Taste.

464-Besitzer die keine Diskettenstation angeschlossen, oder diese nicht eingeschaltet haben, bekommen als Antwort die Zahl 43533 mitgeteilt.

In allen anderen Fällen meldet der CPC 42249 freie Bytes. Daran ist zu sehen, daß durch die Diskettenstation  $43533 - 42249 = 1284$  Bytes weggenommen werden.

CPC 6128-Besitzer erhalten also ebenfalls nur die Angabe der 42249 Bytes, denn die zweite Speicherbank wird durch den Basic-Interpreter nicht "bedient". Hierzu dient das auf der Diskette mitgelieferte Programm "BANKMAN", in diesem Buch zeigen wir aber auch wie es ohne dieses Programm, das unsinnigerweise auch noch geschützt ist, geht!

Machen wir eine weitere Probe. Fragen Sie den CPC doch einmal nach seiner höchsten - von Basic aus benutzbaren - RAM-Speicheradresse. Also nach der Adresse, die durch ein Basic-Programm noch genutzt werden kann.

Dies geschieht durch:

PRINT HIMEM oder PRINT HEX\$(HIMEM)

Diskettenbesitzer bekommen folgende Mitteilungen:

42619 (dezimal) oder A67B (hexadezimal)

Nun subtrahieren wir von diesen Zahlen einfach die "unten fehlenden" Bytes.

Also  $42619 - 367 = 42252$  zur Verfügung stehende Speicherstellen. Die Differenz von 3 Bytes ist deswegen vorhanden, weil Sie den Computer etwas gefragt haben.

Wir haben nun schon ein paar wichtige Punkte festgestellt bzw. bestätigt bekommen:

Der nutzbare Basic-Speicherbereich liegt zwischen &170 und &A67B. (Bei dieser Angabe müssen die Besitzer eines "nackten" CPC 464, also bei nicht angeschlossenem, oder bei nicht eingeschaltetem Diskettenlaufwerk, 1284 Bytes hinzuzählen.)

Das heißt, nur in diesem Bereich können Basic-Programme oder vom Programm verwendete Variablen liegen.

Der Bereich bis einschließlich &16F wird vom Betriebssystem weggenommen, denn dort wird vom Betriebssystem "gearbeitet".

#### 4.2 Die Variablen und Konstanten

Wie Sie dem Handbuch entnehmen können, kennt Ihr CPC verschiedene Arten von Variablen, nämlich Real- (Gleitkomma-), Integer- (Ganzzahl) und String- (Zeichen-) Variablen.



Zusätzlich kennt er diese auch noch in dimensionierter Form, bzw. jede der Variablentypen kann in "Feldform" dimensioniert werden. Bei dimensionierten Feldern spricht man auch von Arrays.

Erhält eine Variable einen "Inhalt" zugewiesen, dann wurde aus dieser eigentlich eine Konstante, denn nun ist bis zur nächsten Änderung der "Wert" ja gleich. Allgemein kann folgendes gesagt werden:

Wird bei einem Programmlauf der Inhalt einer "Variablen" nicht geändert, so ist es eine Konstante, ändert er sich, so ist es eine Variable.

Beispiele:

```
a=5: v= 12: y=&de: t$="Text"
(a,v,y und t$ sind Konstanten)
```

```
for i = 1 to 15:next i
(i ist eine Variable, in diesem Falle die Laufvariable)
```

```
w(3,2)=81
(w ist eine Konstante mit der Feldzuordnung 3,2)
```

Beispiel für ein Matrixfeld:

	1	2	3
1	17	19	08
2	11	02	90
3	<u>300</u>	<u>81</u>	78
4	14	12	43

(In diesem Matrixfeld sehen Sie die Konstante w unterstrichen.)

Der Einfachheit halber betrachten wir nun die Variablen und die Konstanten gleich, d.h. alles was nun über Variablen geschrieben ist, gilt gleichermaßen auch für Konstanten.

Variablen haben einen Namen und einen Wert. Beides muß sich der CPC merken. Deshalb benötigen sie auch dann Speicherplatz, wenn sie im Direktmodus eingegeben und Inhalte zugewiesen werden.

Holen Sie sich doch einmal hierüber die Bestätigung. Setzen Sie den CPC falls Sie ihm schon gearbeitet haben, durch gleichzeitiges Betätigen der Tasten CTRL, SHIFT und ESC zurück und geben Sie dann bitte folgendes ein:

```
a=5: PRINT FRE(0)
```

Als Antwort bekommen Sie nun neun Bytes weniger mitgeteilt als vorher. Denn diese neun Bytes braucht der CPC, um sich sowohl den Variablennamen, als auch den Wert zu merken.

Versuchen Sie es ruhig mit eigenen Eingaben, damit Sie mit Ihrem CPC vertrauter werden, denn durch praktisches Arbeiten lernt man am meisten.

Der Befehl PRINT a, oder bei anderen Namensbezeichnungen natürlich der zugehörige Name, zeigt, daß sich der CPC beides gemerkt hat, denn sonst könnte er Ihnen auf Ihre Frage ja nicht die korrekte Antwort geben.

Sorgen Sie nun einmal dafür, daß der CPC diesen Wert wieder vergißt. Dies kann beispielsweise durch den Befehl CLEAR geschehen. Wenn Sie nun nach der Anzahl der freien Bytes fragen, bekommen Sie als Antwort wieder den Wert, den Sie auf Befragen auch nach dem Einschalten erhielten.

Wie sie gesehen haben, wurden für die Variable "a" neun Bytes benötigt. Dies traf für die Zahl "fünf" zu, bitte überzeugen Sie sich davon, daß bei einem anderen Wert (z.B. 675371) ebenfalls nur neun Bytes "verbraucht" werden.

Damit haben Sie nun festgestellt, daß Gleitkommavariablen jeweils neun Bytes verbrauchen. Oder etwa nicht? Nun, wenn Sie anderer Meinung sind, dann haben Sie als Variablennamen mehr als ein Zeichen benutzt! Oder Sie haben evtl. keine Gleitkommazahl definiert.

Betrachten wir einmal Integerzahlen. Diese haben als Kennzeichnung an der letzten Stelle des Variablennamens ein Prozentzeichen. Im Gegensatz zu Gleitkommazahlen (auch Realzahlen genannt), handelt es sich hierbei um Zahlen ohne Nachkommastellen. Also um sogenannte ganze Zahlen. Zu beachten ist, daß diese Zahlen nur im Bereich zwischen -32767 und + 32768 liegen dürfen, sonst akzeptiert sie Ihr CPC nicht.

Unser kleines Spiel mit der Wertzuweisung zeigt, daß derartige Variablen mindestens sechs Speicherplätze belegen.

Beispiel:

```
a% = 5: PRINT FRE(0)
```

Das Ergebnis sehen Sie ja auf dem Bildschirm. Wenn Sie mehrere Zuweisungen durchführen, dann werden Sie auch hier feststellen, daß jede, wenn die Variablennamen gleich lang sind, die gleiche Anzahl an Speicherplätzen "verbraucht."

Als nächstes ist natürlich von Interesse, wo denn der CPC diese Werte ablegt. Um dies zu erfahren, benutzen wir eine in den CPC eingebaute Möglichkeit. Wir fragen ihn nämlich selbst. Der Printbefehl in Verbindung mit dem Klammersymbol (die Taste rechts neben dem Buchstaben P) und dem Variablennamen gibt nämlich darüber Auskunft, wo denn die Variablen abgelegt wurden.

Testen Sie dies einmal, indem Sie durch:

```
PRINT @ Variablenname
```

nachfragen.

So, damit haben Sie nun genügend "zu Fuß" erledigt. Deshalb soll Ihnen nun ein Programm weiterhelfen.

#### 4.3 MON - Ein modular aufgebautes Hilfsprogramm

Zwar erfolgt die Programmbeschreibung dieses Programmes auch in diesem Abschnitt, aber anfangs sollten Sie sich erst etwas im Speicher des CPC's umsehen und das Programm nur benutzen.

Beim Gesamtprogramm handelt es sich um ein Basic-Programm, welches durch Maschinensprache-Routinen unterstützt wird. Sie können es aber auch anders herum betrachten, denn die Hauptarbeit wird durch die Maschinenprogramme erledigt.

Damit Sie keine größeren Probleme bekommen, gehen Sie bitte nun folgendermaßen vor:

Zunächst tippen Sie den Hexlader des Maschinenprogrammes ab und speichern ihn sicherheitshalber sofort ab. Damit Sie aber wissen, was Sie überhaupt abtippen und auf welche Punkte Sie dabei zu achten haben, sollen Sie zunächst erfahren was dieser Hexlader soll und was er beinhaltet. Wie schon angeführt, erfolgt die Beschreibung des eigentlichen Maschinenprogrammes später.

##### 4.3.1 Programmbeschreibung des Hexladers MON1.HEX

Der Hexlader ist nur ein Hilfsprogramm, um das Maschinenprogramm in die vorgesehenen Speicherstellen zu bringen.

Damit das vom Hexlader erzeugte Maschinenprogramm aber nicht von Basic aus überschrieben wird, muß es vor dem Basic-Zugriff geschützt werden. Deshalb erfolgt in Zeile 100 die Speicherbegrenzung auf &9fff.

In Zeile 880 steht die Anfangs- und Endadresse des Maschinenprogrammes, sowie die Ausgangszeilen-Nummer (ga) für die Datazeilen, um beim Programmlauf des Hexladers Abtippfehler zu erkennen. Fehler in einem Maschinenprogramm erkennt ein Computer nicht, da auf der Maschinensprache-Ebene weder syntaktische, noch irgendwelche anderen Eingabeprüfungen stattfinden.

Fehler im Maschinencode führen in den meisten Fällen zu größeren Systemfehlern. Diese können sich so auswirken, daß der CPC sich "aufhängt", daß er irgendwelche wirren Zeichen auf dem Bildschirm zeigt, oder auch nach allen möglichen Effekten einen Reset durchführt.

Ja, Sie können sich auch einen "Computervirus" eingefangen haben, der sich solange nicht bemerkbar macht, bis eben diese fehlerhafte Routine einmal benutzt wird. Deshalb ist es gerade für Anfänger enorm wichtig, sehr sorgfältig Programme abzuschreiben, denn - und das gilt auch für Basic-Programme - die Fehler zeigen sich oft erst viel, viel später.

Nochmals also die Bitte an Sie, prüfen Sie alle "Abtipparbeiten" sehr sorgfältig, denn wir wissen, daß die meisten in Büchern und Zeitschriften veröffentlichten Programme ablauffähig sind. Im Gegensatz zu früheren Zeiten, wo sie noch mit der Schreibmaschine abgeschrieben wurden, werden die Listings heute ja direkt

über einen - am Computer angeschlossenen - Drucker ausgegeben und entsprechen damit dem Originalprogramm.

Daß ein Programm selbst, trotzdem noch Fehler haben kann, ist eine andere Sache. Selbst größte Softwarehäuser sind nicht davor gefeit, Fehler im Programm zu haben; das für Sie beste Beispiel hierfür ist Ihr eigener CPC.

### Doch weiter mit dem Programm MONI.HEX

Ab Zeile 2000 finden Sie die hexadezimalen Werte des Maschinencodes und am Ende jeder Zeile eine Prüfsumme.

Diese Prüfsumme ist jeweils die Summe der anderen Datastatements dieser Zeile.

Die restlichen Zeilen dienen nur zur Übertragung des Programmes in den dafür vorgesehenen Speicherbereich. Wird beim Programmaufbau ein Fehler in den Datastatements erkannt, dann wird der weitere Programmaufbau abgebrochen und die entsprechende Zeilennummer mitgeteilt.

Die Prüfung erfolgt jeweils immer am Ende einer abgearbeiteten Datazeile und wird durch das "&"-Zeichen ausgelöst (Zeile 890). Diese Prüfung hat nichts mit den im CPC vorhandenen Prüfroutinen für ein Basic-Programm zu tun. Die Prüfung auf korrekte Datastatements ist etwas völlig anderes.

Die richtige Zeilennummer bei Erkennen eines Fehlers kann Ihnen aber nur dann mitgeteilt werden, wenn die von Ihnen eingegebenen Zeilennummern mit den abgedruckten übereinstimmen.

Seien Sie deshalb sehr sorgfältig beim Abtippen, denn dieses Programm wird auch später noch benötigt und bildet die Basis für weitere Kombinationsprogramme (Basic und Maschinensprache).

Ein kleiner Tip für diejenigen, die noch wenig Erfahrung mit Hexzahlen haben: Alle Datastatements ab Zeile 2000 können nur Ziffern von Null bis Neun und Buchstaben von A bis F beinhalten. Ob die Buchstaben in Klein- oder Großschrift eingegeben werden ist unerheblich.

Ist nach dem Programmstart durch RUN die Generierung des Maschinencodes fehlerfrei abgelaufen, dann wird das Maschinencode-Programm sofort als Binärfile abgespeichert.

Ein Binärfile ist der genaue "Speicherauszug", das heißt die Daten werden so auf Kassette oder Diskette geschrieben, wie sie im Computerspeicher stehen.

Beim Programmaufbau werden die Datastatements in Dezimalzahlen umgewandelt und auch auf dem Bildschirm ausgegeben. Dies dient nur zur Kontrolle auf korrekte Statements und weiterhin zur Anzeige, daß der CPC arbeitet. Die für die Ausgabe zuständige Zeile können Sie auch entfernen (Zeile 940).

Der Abspeichervorgang geschieht durch Zeile 9999 automatisch. Wenn Sie dies nicht wollen, dann lassen Sie diese Zeile nicht einfach weg, sondern machen Sie sie zu einer "Bemerkungszeile" indem Sie nach der Zeilennummer ein Hochkomma ' eingeben. Dadurch brauchen Sie, wenn Sie die Zeile später doch noch aktivieren wollen, diese nicht neu einzugeben, sondern nur das Hochkomma wieder zu entfernen.

Sie können also auch durch Eingabe der Befehlssequenz:

```
Save"moni.bin",b,&a000,&16b
```

oder durch Entfernen des eingefügten REMARK-Zeichens und RUN 9999 den Abspeichervorgang auslösen.

Beachten Sie bitte, daß es sich bei den beiden Zahlenangaben nach dem Buchstaben "b" um Hexadezimalzahlen handelt, das heißt, das kaufmännische Und-Zeichen (&) darf nicht fehlen.

Durch obige Befehlssequenz oder durch den Programmlauf haben Sie das direkte Maschinenprogramm abspeichern lassen, welches dann von einem Basic-Programm aus geladen und aufgerufen werden kann.

Der Hexlader wird dann eigentlich nicht mehr benötigt, denn er hat seine Schuldigkeit getan, nämlich das Maschinenprogramm erzeugt. Aber heben Sie sich ihn vielleicht doch noch auf, denn man weiß ja nie ...

Für Besitzer des "reinen" CPC 464 empfiehlt es sich, das Binärfile ggf. auf eine separate Kassette abzuspeichern, damit beim Lade-Aufruf der späteren Basic-Programme dann diese Kassette eingelegt werden kann. Dadurch muß nicht hinter jedem Basic-Programm, welches mittels dieses Binärfiles arbeitet, dieses abgespeichert sein. Andererseits aber müssen Sie dann jedesmal einen Kassettenwechsel durchführen. Wie Sie es handhaben bleibt Ihnen überlassen, aber mit der Zeit werden Sie, wenn Sie länger mit einem CPC arbeiten, schon Routine bekommen.

#### 4.3.2 Listing: MONI.HEX

```
100 MEMORY &9FFF
870 REM MONI.HEX
880 a= 40960:e= 41324:zb=2000
890 FOR i =a TO e:READ d$:IF LEFT$(d$,1)<>"&" THEN 920
900 IF ps<>VAL(d$) THEN PRINT"Fehler Zeile "zb:END
910 ps=d$:d$="":IF i=e THEN 9999:ELSE i=i-1:zb=zb+1:GOTO 950
920 d$="&" + d$
930 POKE i,VAL(d$):ps=ps+VAL(d$)
940 PRINT VAL(d$)
950 IF i < e THEN NEXT i
2000 DATA C3,0F,A0,00,00,00,00,00,00,00,00,00,00,FE,&0270
2001 DATA 03,C0,DB,22,0B,A0,01,06,00,2A,0B,A0,11,03,A0,ED,&04EA
2002 DATA B0,2A,07,A0,22,0D,A0,ED,5B,03,A0,19,22,09,A0,3A,&0559
2003 DATA 05,A0,32,3B,A0,DF,39,A0,C9,3C,A0,00,3E,1E,CD,5A,&0692
2004 DATA BB,CD,B7,A0,CD,C6,A0,CD,CC,A0,2A,07,A0,06,10,CD,&08FF
2005 DATA D2,A0,CD,C6,A0,CD,B7,A0,CD,C6,A0,CD,41,A1,CA,7A,&0AEF
```

```

2006 DATA A0,CD,05,A1,2A,0D,A0,CD,1D,A1,22,0D,A0,CD,41,A1,&06F3
2007 DATA CA,7A,A0,23,CD,C1,A0,C3,5B,A0,CD,C1,A0,CD,B7,A0,&0A45
2008 DATA 3E,01,CD,90,BB,3E,00,CD,96,BB,CD,4B,A1,2A,07,A0,&073D
2009 DATA CD,41,A1,CA,A0,06,10,CD,59,A1,CD,C1,A0,E5,CD,&097C
2010 DATA 50,A1,E1,C3,90,A0,3E,14,CD,72,BB,CD,C1,A0,3E,07,&0804
2011 DATA CD,5A,BB,CD,7B,BB,C9,06,50,3E,8F,CD,5A,BB,10,FB,&08BE
2012 DATA C9,3E,0A,CD,5A,BB,3E,0D,CD,5A,BB,C9,3E,0A,CD,6F,&076D
2013 DATA BB,C9,7D,E6,0F,CD,E2,A0,CD,FF,A0,2C,10,F4,CD,C1,&0A6F
2014 DATA A0,C9,4F,0F,0F,0F,0F,E6,0F,CD,F3,A0,79,E6,0F,CD,&0784
2015 DATA F3,A0,C9,FE,0A,38,02,C6,07,C6,30,CD,5A,BB,C9,3E,&084A
2016 DATA 20,CD,5A,BB,C9,2A,0D,A0,7C,CD,E2,A0,7D,CD,E2,A0,&0939
2017 DATA CD,FF,A0,CD,FF,A0,CD,FF,A0,CD,FF,A0,C9,06,10,E5,&0B74
2018 DATA CD,9C,BB,E1,CD,FF,A0,CD,41,A1,C0,7E,CD,E2,A0,CD,&0B82
2019 DATA FF,A0,23,10,F2,CD,FF,A0,E5,CD,9C,BB,E1,22,0D,A0,&09E9
2020 DATA C9,B7,ED,5B,09,A0,E5,ED,52,E1,C9,3E,04,CD,72,BB,&097B
2021 DATA 3E,40,CD,6F,BB,CD,5A,BB,C9,7E,23,FE,20,30,02,3E,&074F
2022 DATA 90,CD,41,A1,CD,5A,BB,C8,10,EF,C9,00,&06B1
9999 SAVE"moni.bin",b,&A000,&16B

```

Wir haben den Filenamen des Programmes mit der Extension (Erweiterung) ".HEX" versehen, obwohl diese Extension normalerweise für Hex-Dateien des INTEL-Formates reserviert ist.

Aber, um in den Directorys (Inhaltsverzeichnissen der Disketten) gleich zu erkennen, daß es sich um einen "Hexlader" handelt, haben wir dieser Empfehlung entgegen gehandelt. CP/M-Freaks bitten wir um Nachsicht, denn für reine Basic-Programmierer ist diese Festlegung vielleicht gerade am Anfang eine große Hilfe.

Wenn Sie dieses Programm fehlerfrei abgetippt haben und es einwandfrei abgelaufen ist, dann kann das Programm auch gleich getestet werden. Denken Sie aber vorher immer an Sicherheitskopien der Programme, damit bei einem Fehler und bei einem eventuellen Rechnerabsturz die Tipparbeit nicht umsonst war. Gerade bei Beginnern können wir nicht oft genug darauf hinweisen und "alte Hasen" sind selbst schuld, wenn sie es vergessen.

Der Aufruf des Programmes erfolgt durch:

CALL Adresse,Start,Status,Länge

Vorher aber sollten Sie den MODE 2 (80-Zeichen-Modus) ausgewählt haben.

Der Befehl CALL bewirkt einen Aufruf eines Maschinenprogrammes an der Adresse. Die Aufrufadresse des Programmes liegt in diesem Falle bei &A000.

Der Start ist bei diesem Programm der Beginn, ab welcher Speicherstelle der Hexdump (Speicherauszug) ausgeführt werden soll.

Der Status ist der sogenannte ROM-State, darauf gehen wir später noch näher ein. Geben Sie für den Status bitte 255 ein.

Die Länge ist die Anzahl der auszugebenden Speicherstellen. Geben Sie hier bitte 256 ein.

Der Aufruf sollte also so aussehen:

```
CALL &A000,&A000,255,256
```

Spätestens jetzt zeigt sich, ob trotz der Prüfsummen grobe Fehler im abgetippten Programm enthalten sind. Denn trotz des Vergleichs mittels der Zeilensummen, können Sie beispielsweise ja Statements in einer Zeile vertauscht haben, dann stimmt zwar die Prüfsumme, aber nicht das Programm.

Durch den CALL-Aufruf gibt das Maschinen-Programm also einen Hex-dump (mit ASCII-Darstellung) aus und zwar ab der Adresse, die hinter dem eigentlichen Call-Aufruf steht. Durch die weiteren Zahlen wurde festgelegt, daß RAM-Speicher ausgewählt wurde und genau eine PAGE, also 255 Speicherstellen dargestellt wurden. Die Differenz von 1 ergibt sich durch den Programmaufbau. Dies wird im Basic-Teil dann korrigiert.

Wenn also das Programm einwandfrei ist, dann sollte auf Ihrem Bildschirm in etwa Abbildung 1 zu sehen sein.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
A000	C3	0F	A0	00	01	FF	00	00	A0	00	A1	F8	BF	00	A0	FE
A010	03	C0	DD	22	0B	A0	01	06	00	2A	0B	A0	11	03	A0	ED
A020	B0	2A	07	A0	22	0D	A0	ED	5B	03	A0	19	22	09	A0	3A
A030	05	A0	32	38	A0	DF	39	A0	C9	3C	A0	FF	3E	1E	CD	5A
A040	BB	CD	B7	A0	CD	C6	A0	CD	CC	A0	2A	07	A0	06	10	CD
A050	D2	A0	CD	C6	A0	CD	B7	A0	CD	C6	A0	CD	41	A1	CA	7A
A060	A0	CD	05	A1	2A	0D	A0	CD	1D	A1	22	0D	A0	CD	41	A1
A070	CA	7A	A0	23	CD	C1	A0	C3	5B	A0	CD	C1	A0	CD	B7	A0
A080	3E	01	CD	90	BB	3E	00	CD	96	BB	CD	4B	A1	2A	07	A0
A090	CD	41	A1	CA	A6	A0	08	10	CD	59	A1	CD	C1	A0	E5	CD
A0A0	50	A1	E1	C3	90	A0	3E	14	CD	72	BB	CD	C1	A0	3E	07
A0B0	CD	5A	BB	CD	7B	BB	C9	06	50	3E	8F	CD	5A	BB	10	FE
A0C0	C9	3E	0A	CD	5A	BB	3E	0D	CD	5A	BB	C9	3E	0A	CD	6F
A0D0	BB	C9	7D	E6	0F	CD	E2	A0	CD	FF	A0	2C	10	F4	CD	C1
A0E0	A0	C9	4F	0F	0F	0F	0F	E6	0F	CD	F3	A0	79	E6	0F	CD
A0F0	F3	A0	C9	FE	0A	30	02	C6	07	C6	30	CD	5A	BB	C9	3E

Abbildung 1: Ein Teil des Maschinenprogrammes als Hex-ASCII-Dump

Es ist also das Maschinenprogramm, das durch den Hexlader erzeugt wurde und nun für Sie arbeitet.

Benutzen Sie dies aber nicht zum Vergleich mit dem Hexlader, denn durch den Programmaufbau haben sich einige Werte geändert und stimmen deshalb nicht mehr mit dem Original überein!

Um sich nun beispielsweise ein Basic-Programm ab Basic-Beginn anzusehen, müsste der Aufruf lauten:

```
CALL &A000,&170,&ff,&ff
```

oder auch

CALL &A000,368,255,255.

Wenn Sie im Mode der 80-Zeichendarstellung arbeiten und auch die anderen Werte korrekt sind, sehen Sie nun den ersten Teil des Hexladerinhaltes vor sich.

Haben Sie diesen aber durch den Befehl NEW schon gelöscht und noch kein anderes Programm eingeschrieben oder eingeladen, dann sehen Sie das Bild eines leeren Speicher vor sich:

Lauter "Doppelnullen".

Steht das Programm noch im Speicher dann sehen Sie auf dem Bildschirm nun Abbildung 2.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
0170	0A	00	64	00	AA	20	1C	FF	9F	00	0F	00	66	03	C5	20	..d..+...f..)
0180	4D	4F	4E	31	2E	48	45	58	00	28	00	70	03	0D	00	00	MON1.HEX-(.p....
0190	E1	EF	20	1F	00	00	00	20	90	01	00	00	00	E5	EF	20	01 .....^
01A0	1F	00	00	6C	21	90	01	0D	00	00	7A	E2	EF	1A	D0	07	..!!.....z+^
01B0	00	38	00	7A	03	9E	20	0D	00	00	E9	20	EF	0D	00	00	-8.z+. ...^
01C0	E1	20	EC	20	0D	00	00	E5	01	C3	20	03	00	00	E4	01	0 j .....^
01D0	A1	20	FF	75	28	03	00	00	E4	2C	0F	29	F2	22	26	22	hu(.....)^"
01E0	20	EB	20	1D	63	02	20	20	00	30	00	84	03	A1	20	0D	q.c.....
01F0	00	00	70	F3	F2	FF	1D	28	03	00	00	E4	28	20	EB	20	..p+...(.^)
0200	BF	22	46	65	68	6C	65	72	20	5A	65	69	6C	65	20	22	q"fehler Zeile "
0210	00	00	00	7A	E2	01	98	20	00	4B	00	8E	03	00	00	00	...z+...K.....
0220	70	F3	EF	0E	01	03	00	00	E4	EF	22	22	01	A1	20	0D	p+^.....
0230	00	00	E9	EF	0D	00	00	E5	20	EB	20	1E	0F	27	01	01	..01.....^
0240	97	20	0D	00	00	E9	EF	0D	00	00	E9	F5	0F	01	0D	00	...z+...^
0250	00	7A	E2	EF	0D	00	00	7A	E2	F4	0F	01	40	20	1D	B3	..z+...z+...^
0260	02	20	20	00	14	00	98	03	03	00	00	E4	EF	22	26	22	.....^"

Abbildung 2: Ein Hexdump des Hexladers

Durch Eingabe anderer Adressen können Sie auch andere Speicherbereiche ansehen. Das Maschinenprogramm als solches kann, wie Sie sehen, auch eigenständig benutzt werden. Da dies aber noch recht mühsam und auch noch sehr unkomfortabel ist, soll nun das Maschinenprogramm um ein "Basic-Steuerprogramm" erweitert werden.

Das Basic-Programm ist sehr kurz, sollte aber trotzdem mit Sorgfalt abgetippt werden und vor allem sollten Sie sich an die angegebenen Zeilennummern halten. Wer dies nicht tut muß später durch RENUMBER 100,10 diese Numerierung einstellen, denn in's Programm wird nachher noch etwas eingefügt.



## 4.3.3 Listing: Basicteil von MONI

```

100 ' MONI.bas =Hex- und Ascii-Dump
110 '
120 MODE 2:MEMORY &9FFF
130 status = 255
140 IF PEEK(&A000)<>&C3 THEN LOAD"moni.bin",&A000
150 adresse=&170
160 laenge=&FF:laenge=laenge+1
170 '
180 IF adresse <0 THEN adresse=adresse +2^16
190 IF adresse >2^16 THEN adresse =adresse -2^16
200 LOCATE 64,2:PRINT"Moment bitte "
210 CALL &A000,adresse,status,laenge
220 LOCATE 64,2:PRINT"Status : "status
230 LOCATE 1,21
240 PRINT"Seite vor = Cursor rechts      Seite zurueck = Cursor links
250 PRINT"Zeile vor = Cursor nach unten   Zeile zurueck = Cursor hoch
260 PRINT"Leertaste = Aenderung der Adresse S = Status";
270 PRINT TAB(73)CHR$(24);"E = Ende";CHR$(24)
280 LOCATE 1,24:PRINT CHR$(18):LOCATE 1,24
290 '
300 x#=INKEY$:IF x#="" THEN 300
310 x=ASC(x#)
320 IF LOWER$(x#)="e" THEN END
330 IF LOWER$(x#)="s" THEN INPUT"Status : ";status:GOTO 200
340 '
350 IF x=240 THEN adresse$ = STR$(adresse-16):' Cursor nach oben
360 IF x=241 THEN adresse$ = STR$(adresse+16):' Cursor nach unten
370 IF x=242 THEN adresse$ = STR$(adresse-laenge)' Cursor nach links
380 IF x=243 THEN adresse$ = STR$(adresse+laenge)' Cursor nach rechts
390 IF x=32 THEN INPUT"Adresse in hex (&);adresse$:adresse$="&"+adresse$
400 adresse=VAL(adresse$)
410 GOTO 180

```

So, nun haben Sie ein erstes Werkzeug, um Ihrem CPC auf den Zahn fühlen zu können. Glauben Sie uns, es werden interessante und aufschlußreiche Entdeckungsreisen, die Sie mit diesem kleinen Programm unternehmen können und Ihnen sehr viel Wissen über Ihren Computer vermitteln.

Zunächst aber erst einige Erklärungen zum Basicteil.

Die Zeilen 100 und 110 enthalten nur Bemerkungen und dies wird dem CPC entweder durch REM oder durch die im Programm gewählte Kurzform (das Apostroph-Zeichen) mitgeteilt. Er weiß dadurch, daß hinter diesen Zeichen keine Befehle sind, die er auszuführen hat.

In Zeile 120 wird auf die 80-Zeichen-Darstellung geschaltet und der von Basic benutzbare oberste RAM-Speicher auf &9fff gesetzt. Den Grund kennen Sie schon: Damit bei Programmläufen das Maschinenprogramm nicht überschrieben wird.

In der nächsten Zeile wird der RAM-/ROM-Status festgelegt. Der Wert 255 bedeutet, daß RAM ausgewählt ist.

In Zeile 140 prüft das Programm, ob in der Speicherstelle &A000

der Wert &C3 steht. Wenn ja, dann setzt das Programm einfach voraus, daß das Maschinenprogramm schon vorhanden ist. Wenn dort aber ein anderer Wert steht, dann wird das Maschinenprogramm (Binärfile) erst geladen.

Wurde der CPC nicht "jungfräulich" gestartet, kann es natürlich sein, daß dort zufällig auch schon der Wert &C3 steht. Dann kann es natürlich zu einem Fehlverhalten kommen. Da wir aber keine zu große Prüfroutine in's Programm einbauen wollten, und auch nicht bei jedem Programmstart der Maschinencodeteil "nachgeladen" werden sollte, haben wir der Einfachheit halber diese Art der Lösung gewählt.

In Zeile 150 steht die Startadresse, die willkürlich auf den Basicstart gelegt wurde.

In Zeile 160 steht dann die Länge des auszugebenden Dumps. Aufgrund der Eigenart des Maschinenprogrammes muß die Zahl eins addiert werden, dies wurde bereits angemerkt.

Diese Addition erfolgt deshalb im Programm, damit Sie es nicht andauernd berücksichtigen müssen.

In den nächsten Zeilen (180 und 190) erfolgt eine Prüfung, ob die Adresse kleiner als Null oder größer als 65535 ist. Durch diese Abfragen und die dann eventuell zu erfolgenden Korrekturen, kann dieser kleine Hex-Ascii-Monitor sowohl vorwärts als auch rückwärts über &0000 und &FFFF "hinweglaufen".

Die folgende Zeile positioniert den Cursor für eine Ausgabemeldung. Wohin positioniert wird, sehen Sie ja beim Programmlauf.

In Zeile 210 wird dann das Maschinenprogramm aufgerufen und die erforderlichen Werte übergeben. Nun erfolgt durch die Maschinenroutine die Ausgabe der Speicherinhalte.

Bei der Ausgabe dieser Adressinhalte zeigt das Programm eine kleine Kopfleiste, die nur dazu dient, bei den 16 ausgegebenen Werten (pro Zeile) eine Hilfestellung zur genauen Adreßbestimmung zu geben.

Um Ihnen aber umfassende Informationen zu liefern, folgt nun zum Abschluß dieses Teilabschnittes noch der Maschinencodeteil dieses kleinen Monitors kommentierter Form.

Selbst wenn Sie heute noch nicht sehr viel damit anfangen können, sehen Sie sich trotzdem das Listing etwas an und betrachten Sie auch die Erklärungen bzw. Kommentare. Irgendwann können Sie die Informationen wahrscheinlich doch noch sehr gut gebrauchen.

Außerdem sollen Sie ja auch soweit wie möglich, wirklich umfassend, über die in diesem Buch enthaltenen Programme informiert werden, damit Sie diese ggf. auch noch an Ihre Wünsche anpassen können. So zum Beispiel zur Anpassung eines speziellen Druckers.

Auch wenn Sie in die Maschinensprache-Programmierung einsteigen wollen, haben Sie durch die kommentierten Assemblerlistings be-

reits hervorragende Arbeitsunterlagen über lauffähige Programme, mit denen sich experimentieren und arbeiten läßt. Eine Arbeit könnte beispielsweise das Umschreiben für den Mode 1, also für den 40-Zeichen-Mode sein.

Bitte versuchen Sie nun nicht (aus Versehen natürlich) das Programm abzutippen, denn Sie bekommen es nicht in den CPC und vor allem könnte dieser ohne einen entsprechenden Assembler damit überhaupt nichts anfangen. Außerdem haben Sie das Programm ja schon, es wurde beim Programmaufbau des Hexladers erzeugt!

### Anmerkungen zum Assemblerlisting

Das Assemblerlisting wurde mit einem Texteditor erstellt, also mit nichts anderem, als einem Textverarbeitungsprogramm. Es kommt nun auf den Assembler an, ob er Texte eines Textverarbeitungsprogrammes ohne Probleme versteht oder nicht. Üblicherweise haben Assemblerprogramme einen Texteditor integriert und können ASCII-Files verarbeiten, also auch lesen und schreiben.

Am Anfang des Quellcodes (Quellcode deshalb, weil er der Ursprung, also die Quelle der eigentlichen Maschinenprogramme ist) steht der Name, sowie kurze Bemerkungen. Damit man sich später einmal, wenn einige Monate, oder sogar Jahre, vergangen sind, noch an den Sinn des Programmes erinnert, ohne es gleich komplett lesen zu müssen, sind derartige Kurzinformationen sehr sinnvoll.

Die Semikolons am Anfang einer Assemblerzeile haben den gleichen Sinn wie ein REM in Basic, sie sagen dem Assembler:

"Nichts zu arbeiten, reine Bemerkung".

Danach folgt eine Aufstellung der verwendeten Firmware-Routinen, also Maschinenspracheteile des Betriebssystems, die ausgenutzt werden sollen. Denn wenn in einem Computer verschiedene Routinen schon vorhanden sind, warum sollte man diese dann nochmals schreiben? Diese Zuweisungen sind denen von Basic ähnlich und bedeuten in etwa das gleiche wie: a=5:b=12..... usw.

Dann folgt die Assembleranweisung `ORG &A000`. Diese bewirkt, daß der Assembler den Maschinencode für diese Adresse assembliert, also erzeugt.

Der in der nächsten Programmzeile stehende Befehl `JP START1` bedeutet Springe zum Label `START1`. Er entspricht also vergleichsweise dem Basicbefehl "GOTO Zeilennummer".

JP ist der mnemonische Code für den Befehl Springe (engl.= jump).

Der Labelname ist willkürlich gewählt, wobei man sich als Programmierer am besten Kurznamen einfallen läßt, die einen Bezug zum Sinn haben und sich außerdem während der Programmierarbeiten leicht merken lassen. Labels sind also "Marken" die im Quellcode angesprochen werden. Da die direkten Adressen für das Maschinenprogramm ja noch nicht bekannt sind, ist es sehr einfach und auch komfortabel, mit derartigen "Merkern" zu arbeiten.

Eine kleine Erklärung noch, weshalb die Arbeit mit diesen Labeln eigentlich komfortabler ist als mit einer direkten Adresse. Nehmen wir wieder ein Basic-Programm und einmal nicht den so "basic-befehlsstarken" CPC.

"Basic-Sprünge" erfolgen meist durch GOTO Zeilennummer. Ändert man diese aber, dann kann der Sprung ja nicht mehr korrekt ausgeführt werden.

Ein Ändern von Zeilennummern ist aber nicht auf jedem Computer so komfortabel wie auf dem CPC, und um nun beispielsweise zwischen zwei Basic-Zeilen, die nur den Abstand eins auseinander liegen, eine Zeile dazwischenzuschieben, muß bei anderen Computern oft mühsam von Hand, ein Teil des Programmes umnummeriert werden.

Wieder andere Computer ändern beim RENUMBERN die den Befehlen GOTO oder GOSUB folgenden Zeilennummern nicht, usw.

Ein Arbeiten mit Labeln erfordert überhaupt keine Zeilennummern, deshalb der große Komfort. Dem Befehl ORG eine andere Adresse zugewiesen und schon kann der Assembler wieder loslegen, nichts sonst muß geändert werden.

Beim Assemblerlauf ordnet der Assembler, also das Programm, das den Quellcode übersetzt, die richtigen Adressen automatisch selbst zu.

Zwischen dem eigentlichen Programmstart und dem "JUMP-Befehl" geschieht die Festlegung von Merkern, also Hilfsspeicherstellen, die dazu dienen, was der Name aussagt, nämlich merken bestimmter Werte in den zugeordneten Adressen.

Nach diesen folgt dann das eigentliche Programm. Damit wollen wir die Erklärungen der Syntax eines Maschinenprogrammes an dieser Stelle abbrechen, denn für eine detailliertere Einführung in das Programm und in die Maschinenspracheprogrammierung ist dieses doch etwas zu umfangreich.

An anderer Stelle im Buch werden wir einige kürzere Programme in Maschinensprache weiter "zerpflücken" um Sie noch besser mit dieser Programmiersprache vertraut zu machen. Hier nun das Programm.

#### 4.3.4 Assemblerlisting: MON1.EDI

```
; *****
;          mon1.edi          ***
;  *** Dies ist ein Maschinenprogramm fuer ***
;  ***   Hex- und Ascii-Dumps           ***
;  *** Startadresse, Laenge und Select ***
;  *** sind von Basic aus zu uebergeben ***
;  *****
;
;Firmwareaufrufe-----
print equ &bb5a:.invers equ &bb9c:.sethor equ &bb8f:.setver equ &bb72
penset equ &bb90:.paperset equ &bb96:.curenab equ &bb7b
```

```

org &a000 ;startadresse festlegen-----
jp start1 ;Sprung zum eigentlichen Programmstart

;benutzte Marker-----

ziel
laenge defs 2:select defs 2:anfang defs 2
ende defs 2:quelle defs 2:zwadr defs 2

start1
;pruefen auf korrekte Parameteranzahl-----
cp a,3 ;falls keine 3 Parameter mitgegeben wurden
ret nz ;dann wieder zurueck zum Basic

ld (quelle),ix ;Parameteradresse merken
ld bc,&0006 ;Drei Parameter = 6 Bytes

;uebergebene Parameter in Merker laden und Block verschieben-----
ld hl,(quelle) ;in HL die Quelle
ld de,ziel ;in DE das Ziel
ldir ;verschieben

;Ende berechnen und Startadresse in Zwischenadressspeicher-----
ld hl,(anfang) ;Anfang in's HL-Register
ld (zwadr),hl ;und merken fuer spaeter
ld de,(laenge) ;uebergebene Laenge des gewünschten Dumps
add hl,de ;zum Anfang addieren und als
ld (ende),hl ;Endeadresse merken
ld a,(select) ;selekt zum Maschinenprg-Start
ld (stat),a ;und als Rom-/Ram-Status merken
;-----

begin ;Restart um auch ROMs
rst &18 ;auslesen zu koennen
defw mpbeginn ;hier ist die Adresse fuer das weitere Programm
ret ;-----

mpbeginn
defw start2
stat defb &00 ;hier steht spaeter der ROM-Select

start2
ld a,30 ;Cursor
call print ;in linke obere Ecke
call line ;weisse Trennlinie ausgeben
call cr ;und Wagenruecklauf
call locate1 ;Cursor positionieren
ld hl,(anfang) ;HL mit Anfangsadresse laden
ld b,16 ;B beinhaltet Anzahl der pro Zeile auszugebenden Stellen
call leiste ;jeweils das entsprechenden Byte als Kopfleiste
call cr ;und Wagenruecklauf
call line ;weisse Trennlinie ausgeben
call cr ;Wagenruecklauf

wieder
call check ;auf Ende pruefen

60

```

```

jp z,hexend      ;Hexausgabe beenden
call adresse     ;Speicheradresse ausgeben
ld hl,(zwadr)    ;hl mit dem Zwischenspeicher der Adresse laden
call hexzei      ;Hexzeichen ausgeben
ld (zwadr),hl    ;augenblickliche Adresse merken
call check       ;auf Ende pruefen
jp z,hexend      ;wenn ja dann Hexausgabe beenden
inc hl           ;Adresse erhoehen
call crlf        ;CR und LF ausgeben
jp wieder        ;wieder weitermachen

hexend
call crlf        ;CR und LF ausgeben
call line        ;Trennlinie ausgeben
ld a,i           ;Pen- und
call penset      ;
ld a,0           ;Paper-Farbe zuordnen
call paperset    ;
;-----hexausgabe beendet
;-----neu positionieren und Ascii-Ausgabe
call locate2     ;Cursor auf ASCII-Position bringen
ld hl,(anfang)   ;mit Anfangsadresse laden

weiter2
call check       ;auf Ende pruefen
jp z,pgende      ;Ende erreicht
ld b,16          ;16 Zeichen pro Zeile
call ascii       ;Ascii/Ausgabe aufrufen
call crlf        ;CR und LF ausgeben
push hl          ;hl merken da durch locate hl zerstoert wird
call locate3     ;neue Position
pop hl           ;Adresse wieder holen
jp weiter2       ;und weitermachen
pgende ld a,20    ;Programmende
call setver      ;Cursor auf Vertikalposition 20
call crlf        ;CR und LF ausgeben
ld a,7           ;Ausgabe fertig - akustisch
call print       ;mitteilen
call curetab     ;Anwendercursor wieder zulassen
ret              ;-----Ausgabe beendet
;-----Subroutinen-----

line             ;Balkenausgabe
ld b,80          ;80-mal
ld a,143         ;ausgefuelltes Rechteck ausgeben

line2
call print       ;ein Zeichen ausgeben
djnz line2       ;noch nicht fertig
ret              ;Balken ausgegeben

;-----CR und LF ausgeben
crlf
ld a,10
call print
cr
ld a,13

```

```

call print
ret

;-----Position fuer Hex
locatel      ;Horizontalposition 1
ld a,10      ;auf 10. Zeichen
call sethor  ;positionieren
ret

;-----Kopfleiste ausgeben
leiste
ld a,l      ;Low-Byte in Akku
and &f      ;Bit sieben
call hexout ;ausgeben
call space  ;Zwischenraum
inc l      ;Wert erhoehen
djnz leiste ;Ausgabe noch nicht fertig
call crlf   ;CR und LF ausgeben
ret         ;fertig

;-----Hexausgabe
hexout
ld c,a      ;Akkuinhalt retten
rrca       ;viermal
rrca       ;rechts
rrca       ;rotieren
rrca
and &0f     ;Bit 7 - 4 weg
call umsetz ;High-Nibble-Ausgabe
ld a,c     ;Akku wieder holen
and &0f     ;Bit 7 - 4 weg
call umsetz ;Low-Nibble-Ausgabe
ret        ;fertig

;-----
umsetz
cp &a      ;groesser 9
jr c,ziffer ;nein also Ziffer
add a,7    ;Buchstabenabstand addieren

;-----
ziffer
add a,48   ;fuer Hexzeichen 48 addieren
call print ;Hexzeichen ausgeben
ret

;-----
space      ;Leerzeichen
ld a,&20
call print ;ausgeben
ret

;-----
adresse    ;Adressenausgabe
ld hl,(zwadr) ;aktuelle Adresse holen
ld a,h     ;High-Byte
call hexout

```

```

ld a,l          ;Low-Byte
call hexout
call space      ;viermal
call space      ;Leerzeichen
call space      ;als Zwischenraum
call space      ;ausgeben
ret

;-----Hexzeichen ausgeben
hexzei
ld b,16
push hl         ;Da Invertierung HL zerstört "retten"
call invers     ;Darstellung invertieren
pop hl          ;HL wieder holen
call space      ;Leerzeichenausgabe

;-----Hexzeile ausgeben und Ende prüfen
zeilout
call check      ;auf Ende prüfen
ret z           ;Ende erreicht
ld a,(hl)       ;Akku mit Speicherinhalt laden
call hexout     ;Hexwandlung durchführen
call space      ;Leerzeichenausgabe
inc hl          ;Adresse erhöhen
djnz zeilout    ;noch nicht fertig
call space      ;Leerzeichen
push hl         ;retten
call invers     ;invertieren
pop hl          ;wieder holen
ld (zwadr),hl   ;Adresse in Zwischenadress-Merker
ret

;-----auf Ende prüfen
check
or a            ;Carry löschen
ld de,(ende)    ;Endadresse
push hl         ;retten
sbc hl,de       ;subtrahieren (Carry als Flag)
pop hl          ;Adresse wieder holen
ret

;-----Position fuer ASCII
locate2
ld a,4          ;Cursor vertikal auf Position 2
call setver
locate3         ;Cursor horizontal verschieben
ld a,64
call sethor
call print
ret

;-----
ascii
ld a,(hl)       ;Akku mit Inhalt des Speichers laden
inc hl          ;Adresse erhöhen
cp &20          ;keine Steuerzeichen
jr nc,pruef     ;zulassen

```



```
ld a,144      ;Sonderzeichen mittiger Punkt
pruef
call check    ;Ende erreicht
call print
ret z
djnz ascii    ;Ascii-Ausgabe noch nicht beendet
ret
END;-----
```

Damit haben Sie also alle Informationen über das bisherige Programm, mit dem Sie Ihren CPC untersuchen können. Diese Untersuchungen können sehr weitreichend sein. Dazu muß man teilweise aber schon einiges über seinen CPC wissen. Und genau um dieses Wissen zu erarbeiten, kann dieses Programm eingesetzt werden.

Bevor Sie aber nun wirklich mit Ihren Untersuchungen beginnen, stellen wir Ihnen gleich eine weitere Hilfsroutine zur Verfügung.

Man kann sich ja unmöglich alles merken und deswegen ist es sinnvoll, wenn man auch alles "Schwarz auf Weiß" hat.

Eine Druckeroutine für das bisherige Programm fehlt also noch. Am besten hierfür ist natürlich eine Hardcopy, damit man später bei Vergleichen ein genaues Abbild des Schirmes hat.

Wir hätten natürlich alles gleich in ein Programm integrieren können, aber dadurch, daß wir einzelne Module verwenden, können diese auch einzeln eingesetzt werden oder mit anderen Programmen zusammenarbeiten.

Das erspart Ihnen bei Ihren eigenen Programm-Entwicklungen sehr viel Arbeit, da Sie diese Module auch für Ihre eigene Software einsetzen können.

Auch wenn Sie schon über eine Hardcopy-Routine verfügen sollten, kommen Sie leider nicht um diese Version herum, denn erstens ist sie speziell auf das Monitorprogramm abgestimmt und zweitens bietet sie gegenüber ähnlichen Programmen auch noch einen großen Vorteil; warum erfahren Sie noch.

Diese Ausdrucksmöglichkeit ist wie schon geschrieben, nicht unbedingt erforderlich, deswegen können Sie auch ohne diese loslegen, aber früher oder später brauchen Sie doch einen Ausdruck, deswegen die Hardcopy-Routine wieder in der schon bekannten Hexladerform.

Zum Ausdruck Hexlader sei hier angemerkt, daß der Name HEXLADER ein willkürlich gewählter Ausdruck ist, der nur darauf hindeutet, daß ein Programm, welches Statements in hexadezimaler Form enthält, diese in die entsprechenden Speicherstellen POKE't, also "läd". Für derartige "Programmgeneratoren" werden unterschiedlichste Namen benutzt. So sind auch die Begriffe BASIC-LADER, CODEGENERATOR und noch einige weitere Bezeichnungen in den Veröffentlichungen zu finden.

Wir fanden den Begriff HEXLADER oder HEXLOADER am treffendsten und auch leicht zu merken. Außerdem wird ja kein Basic "geladen"

sondern ein Basic-Programm, in welchem der Maschinencode in den Datastatements steht. Wenn man vom ursprünglichen Sinne des "Ladens" ausgeht, wird natürlich ein Basic-Programm geladen, das dann wiederum erst den Maschinencode erzeugt.

Leser, die bisher einen anderen Namen benutzen mögen uns verzeihen, aber z.B. der Assembler-Befehl

LD A,C

heißt nun einmal, lade den Akkumulator mit dem Inhalt des Registers C, und was wir mit dem Programm erreichen wollen, ist, daß das Maschinenprogramm an den richtigen Platz gebracht (geladen) wird.

Hätten wir die Datastatements in Dezimal, so hätten wir ein derartiges Programm wahrscheinlich DEZLADER getauft. Welcher Ausdruck nun treffend ist, bzw. welcher sich einbürgert und dann deshalb gültig wird, überlassen wir der Zukunft. Wir selbst wenden Übrigens auch oft andere Namen an, wie Sie in diesem Buch sehen können!

Wie dem auch sei, hier unser Hexlader für die Hardcopy-Routine.

#### 4.3.5 Listing: HC.HEX eine Hardcopy-Routine

```
870 REM HC.HEX
880 a= 41324:e= 41449:zb=2000
890 FOR i =a TO e:READ d$:IF LEFT$(d$,1)<>"&" THEN 920
900 IF ps<>VAL(d$) THEN PRINT"Fehler Zeile "zb:END
910 ps=d$:d$="":IF i=e THEN END:ELSE i=i+1:zb=zb+1:GOTO 950
920 d$="&"&d$
930 POKE i,VAL(d$):ps=ps+VAL(d$)
940 PRINT VAL(d$)
950 IF i < e THEN NEXT i
2000 DATA FE,02,C0,21,DE,A1,CD,D2,A1,EB,DD,5E,02,DD,56,03,&00FE
2001 DATA E5,ED,52,CB,7C,28,02,E1,C9,21,E2,A1,CD,D2,A1,11,&0934
2002 DATA 01,00,E1,0E,00,CD,C4,A1,28,02,CB,D9,CD,C4,A1,28,&074A
2003 DATA 02,CB,D1,CD,C4,A1,28,02,CB,C9,CD,C4,A1,28,02,CB,&08B5
2004 DATA C1,79,CD,2B,BD,30,FA,13,E5,21,80,02,A7,ED,52,E1,&087B
2005 DATA 28,B8,01,08,00,09,18,CB,D5,E5,C5,CD,F0,BB,C1,E1,&086E
2006 DATA D1,2B,2B,FE,00,C9,46,23,7E,CD,2B,BD,30,FA,23,10,&06E7
2007 DATA F7,C9,03,1B,33,0C,05,0A,1B,4C,7F,02,00,&0314
9999 SAVE "HC.BIN",b,&a16c,124
```

Wie der Hexlader arbeitet, dürfte Ihnen aufgrund des ersten Teiles des kleinen Monitors schon bekannt sein, denn sonst wären Sie noch nicht soweit vorgedrungen mit Ihrer Programmierarbeit.

Deshalb hier keine weiteren Erklärungen über den Programmaufbau, sondern lediglich die wichtige Information, wohin der Hexlader dieses Maschinenprogramm "lädt".

Der Maschinencodeteil wird unmittelbar hinter MON1.BIN geschrieben, also direkt hinter den ersten Maschinenspracheteil, der von &A0000 bis &A16A liegt.

Zwischen den beiden Maschinen-Programmtteilen haben wir ein Byte freigelassen, um beispielsweise bei Betrachtungen mit einem Disassembler, eine Grenze erkennen zu können.

Die Hardcopyroutine "residiert" deshalb von &A16C bis &A1E7.

Nach dem Programmlauf können Sie durch:

RUN 9999

das entsprechende Binärfile erzeugen. Sie können aber auch den Befehl END in Zeile 310 durch 9999 ersetzen. Dadurch wird der Abspeichervorgang automatisiert. Wie dies aussieht, können Sie dem Programm MON1.HEX entnehmen (Zeile 910).

Wie Sie aus obigen Beispiel (HC.HEX - Zeile 9999) sehen, können Dezimal- und Hexadezimalzahlen auch gemischt eingesetzt werden. Geben Sie also vor der Zahl 124 kein &-Zeichen ein!

Bevor wir nun zum ersten Einsatz des kleinen Monitorprogrammes auffordern, wenden wir uns nun erst einmal der Beantwortung der noch offenen Frage, warum dieses Hardcopy-Programm - gegenüber anderen - eigentlich Vorteile hat, zu.

#### 4.3.6 Ein starkes Hardcopy-Programm

Beim Aufruf dieses Maschinenprogrammteiles müssen, wie schon beim Monitorprogramm, ebenfalls Parameter "mitgegeben" werden.

Diese Parameter bestehen aus der Angabe der untersten Pixelzeile, die ausgegeben werden soll und aus der obersten. Das zeigt schon, wie "stark" auch dieses Maschinenprogramm ist.

Oft beschränkt sich der Informationsanteil, den man haben möchte, nur auf einen Teil der Bildschirmdarstellung. Die üblicherweise veröffentlichten Hardcopyroutinen aber drucken immer den gesamten Schirminhalt.

Dies ist ein gravierender Nachteil der meisten Hardcopyroutinen.

Diese "Hardcopy" aber bietet Ihnen die - wenn auch begrenzte - Auswahlmöglichkeit, den Schirmteil zu definieren, den Sie zu Papier bringen wollen.

Leider hängt es aber natürlich vom Drucker ab, ob er Ihnen überhaupt eine Hardcopy erlaubt. Die Routine läuft sicher auf dem NLQ 401 und auf "Epson-Kompatiblen".

Damit Besitzer anderer Drucker aber die Routine ggf. anpassen können, hier gleich, bevor es weitergeht, das Assemblerlisting.

#### 4.3.7 Assemblerlisting: HC.EDI - Ein Hardcopyprogramm

```

;*****
;##          HC.EDI          ##
;##          Aufruf mit      ##
;## Calli ORG, unterste,oberste Pixelzeile ##
;*****

ORG      &a16c

testxy   equ    &bbf0        ;Test (x,y)
druck    equ    &bd2b        ;Ausgabe #8

;Aufruf call ymin, ymax

start    cp      2           ;Beide Parameter da?
         ret     nz          ;nein ->Basic
         ld      hl,text1    ;Druckereinstellung
         call    sub2        ;auf 12/216 Zoll
         ex      de,hl       ;HL ymax

main     ld      e,(ix+2)    ;DE ymin
         ld      d,(ix+3)
         push    hl          ;Wenn ymax-ymin kleiner
         sbc     hl,de       ;0 ist, dann Ende.
         bit     7,h
         jr      z,main1
ende     pop     hl
         ret

```

```

main1  ld    hl,text2    ;Drucker auf Bit-
      call  sub2         ;ausgabe stellen

      ld    de,1         ;x=1
      pop   hl

main2   ld    c,0         ;Ausgaberegister ist 0
      call  sub1
      jr    z,bit2
      set   3,c

bit2    call  sub1
      jr    z,bit1
      set   2,c

bit1    call  sub1
      jr    z,bit0
      set   1,c

bit0    call  sub1
      jr    z,ausgab
      set   0,c

ausgab  ld    a,c         ;C wird an den
      call  druck        ;Drucker ausgegeben.
      jr    nc,ausgab
      inc   de           ;x=x+1
      push  hl

      ld    hl,640        ;Wenn DE 640 ist, dann
      and   a            ;wird die naechste
      sbc   hl,de        ;Zeile kopiert.
      pop   hl
      jr    z,main

      ld    bc,8         ;y wird wieder her-
      add   hl,bc        ;gestellt.
      jr    main2

;*** Unterprogramme

sub1    push  de         ;Jedes Pixel wird auf
      push  hl          ;seine Farbe getestet.
      push  bc
      call  testxy
      pop   bc
      pop   hl
      pop   de
      dec   hl          ;y=y-2
      dec   hl
      cp    0           ;Ist Pixel gesetzt?
      ret

sub2    ld    b,(hl)      ;Ein Text wird zum Druck-
      inc   hl          ;ker geschickt. B ist
sub22   ld    a,(hl)      ;Zaehler, in A steht das
      call  druck        ;Zeichen.
      jr    nc,sub22

```

```

        inc     hl
        djnz   sub22
        ret

text1   defw   &1b03      ;ESC, "3", 12
        defw   &0c33

text2   defw   &0a05      ;&0a, ESC, "L"
        defw   &4c1b      ;639
        defw   &027F

        end
;-----

```

Für eine Druckereinstellung sind gerade die letzten Zeilen dieses Assemblerlistings wichtig. In diesen stehen nämlich die relevanten Bytes zur Anpassung.

Bei einem Drucker bestimmen Sie (oder ein Programm) nämlich in vielen Fällen durch die sogenannten Steuer-Sequenzen, wie er sich verhalten soll.

Diese Befehle an den Drucker werden - abgesehen von wenigen anderen Steuermittellungen - meist durch den Befehl ESC eingeleitet. Deshalb spricht man auch von ESCape-Sequenzen.

Ein Befehl zum Seitenvorschub in Basic lautet beispielsweise:

```
Print #8,chr$(27);chr$(12);
```

man kann ihn aber auch anders schreiben, nämlich

```
Print #8,&1B,&0C;
```

oder auch

```
Print #8,chr$(12);
```

Die letzte Befehlsform wird aber nicht von allen Druckern in dieser Weise akzeptiert, da der ESCAPE-Befehl fehlt.

Chr\$(27) oder chr\$(&1B) ist der Code für ESC.

#### 4.3.6 Falls Sie einen speziellen Drucker haben

Wir gehen nun deshalb etwas näher auf die Druckeranpassung ein, weil wir aus der Erfahrung heraus wissen, daß bei sehr vielen Computeranwendern, die Druckerhandbücher nicht nur mit sieben Siegeln versehen, sondern oft auch noch "verklebt", "verschraubt" und "vernietet" sind.

Dabei ist alles gar nicht so schwer. Nur die Mühe, das Handbuch zu lesen und gegebenenfalls auch ein paar Beispiele durchzuarbeiten, sollte man sich schon machen, auch wenn manche Handbücher nicht gerade dazu geeignet sind, einem den Drucker "spielend"

näherzubringen.

Steuerbefehle an den Drucker werden also sehr oft mit dem Befehl `chr$(27)` eingeleitet. Versuchen Sie aber nun bitte nicht, den Befehl mit der ESC-Taste zum Drucker zu schicken! Das klappt bestimmt nicht.

Wie Sie wissen entspricht der Dezimalwert 27 dem hexadezimalen Wert `&1B`. Um dem Drucker mitzuteilen, daß nun eine Steuersequenz kommt, muß deshalb zunächst dieser Wert geschickt werden. Dabei ist es meist gleichgültig, ob das in Dezimal oder Hexadezimal geschieht.

#### Zeilenvorschub einstellen

Um dem Drucker zu sagen, daß er das Papier bei einem Zeilenvorschub nicht mehr so weit verschieben soll wie bisher, (bei der Hardcopy muß er ja geringer sein) muß ihm also zunächst der ESC-Befehl geschickt werden.

Also: `Print #8,chr$(27);`

Das Semikolon am Ende heißt für den Drucker keinen Zeilenvorschub nach diesem Befehl ausführen.

Als nächstes muß dann die Mitteilung, welchen Zeilenabstand Sie wünschen, geschickt werden. Für unsere Hardcopy soll ein Zeilenabstand von 12/216-Zoll gewählt werden.

Deshalb muß nach ESC der Befehl, der dem Drucker dies mitteilt, gesandt werden. (Besitzer des NLQ 401 können nun auf Seite 26 des Handbuches zum Drucker nachschlagen.)

Der Befehl für n/216-Zoll lautet ESC 3 n.

Der Kleinbuchstabe n ist die gewünschte Anzahl der Punktschritte. Die Ziffer 3 bedeutet 1/3 Punktschritte. Da wir 12 Punktschritte wünschen, muß die komplette Befehlssequenz lauten:

Lt. Handbuch	ESC 3 12	oder
in Dezimal	27 3 12	oder
in Hexadezimal	1B 03 0C	

Also: `print #8,chr$(27);chr$(3);chr$(12);` oder  
`print #8,chr$(&1b);chr$(&03);chr$(&0c);`

Nun betrachten Sie bitte die Zeichenfolge bei text1 ziemlich am Ende des Assemblerlisting, denn dort kann, wenn es sein muß, die entsprechende Befehlsfolge, verändert werden. Dies gilt für Besitzer eines Assemblers.

Aber auch ohne Assembler kommt man zurecht, denn man kann die Werte ja auch POKEn.

Deswegen kann es auch im Hexlader leicht geändert werden, nur die

Prüfsummen müssen dann angepasst oder ganz entfernt und die Prüfroutine selbst "stillgelegt" werden.  
Die entsprechenden Bytes befinden sich auch im Hexlader ziemlich am Ende des Programmes.

#### Grafikmodus einstellen

Auch die Befehle für den Grafikmodus sind ziemlich am Ende des Programmes (text2).

Dieser Teil bewirkt, daß der Grafikmodus mit doppelter Dichte eingeschaltet wird. (NLQ-Handbuch Seite 50.)

Hier lautet der allgemeine Befehl:

ESC L n1 n2

Dabei ist zu berücksichtigen, daß  $n1 + n2 * 256$  nicht größer sein darf als lt. Handbuch zugelassen, denn sonst wird der Befehl nicht akzeptiert!

Beim NLQ 401 muß der Wert also kleiner oder gleich 960 sein. Die Folge

1B 02 7F

ist dafür zuständig.

Wenn Sie diese Werte umrechnen, erhalten Sie die Zeilenbreite.

Der Wert für die gesamte Zeile ist:

$02 * 256 + \&7F = 639$ , d. h. so breit ist eine Druckzeile!

Weiter können wir "Fremddrucker"-Besitzern leider nicht helfen, denn wir wissen ja nicht, welcher Drucker angepasst werden muß und haben auch nicht dessen Handbuch vorliegen.  
Hier hilft nur eigenes Handbuchstudium weiter. Voraussetzung ist aber, daß der Drucker diese Funktionen überhaupt durchführen kann.

Diejenigen, die einen Drucker haben, der direkt mit unserer Hardcopy-Routine arbeitet, mussten nun leider etwas Geduld haben, denn wir wollten erst sicherstellen, daß auch Besitzer anderer Drucker eine Chance haben, Bildschirm-Ausdrucke zu erstellen, sofern der Drucker dies zuläßt.

#### Nun geht es wieder für alle weiter

Wie die Abspeicherung als Binärfile zu erfolgen hat, haben Sie schon erfahren. Um nun mit dem Gesamtprogramm arbeiten zu können, gibt es verschiedene Vorgehensweisen. Am bequemsten ist sicherlich, eine entsprechende Basic-Zeile für den Ladevorgang in das Programm "einzubauen". Am einfachsten geschieht dies dadurch, diese in das Basic-Listing direkt einzuschreiben. Wir wollen aber gleich, um auf eine weitere Besonderheit beim CPC 464 hinzuweisen, einen anderen Weg gehen.



## 4.3.9 Umgehung des MERGE-Fehlers beim 464

Der CPC 464 hat nämlich (in Verbindung mit der Diskettenstation) einen Fehler in der MERGE-Routine, der zwar durch "patchen" korrigiert werden kann, aber wer findet gleich immer das Patchprogramm hierfür?

Um das Auftreten dieses Fehlers zu vermeiden, gibt es einen viel einfacheren Weg. Der Effekt tritt nämlich bei als ASCII-File abgespeicherten Programmen nicht auf. Es ist deshalb möglich auch lange Programmteile miteinander zu vermischen (mergen).

Um die Vorgehensweise kennenzulernen, schreiben Sie deshalb bitte folgendes einzeiliges Programm:

```
145 IF PEEK(&A16C) <> &FE THEN LOAD "hc.bin",&A16C
```

und speichern es als ASCII-File auf Ihren Massenspeicher, also auf Kassette oder Diskette.

Zum Beispiel folgendermaßen:

```
SAVE"zeile",a
```

Haben Sie dies getan, dann laden Sie den Basicteil von "mon1.bas" in den CPC. Sobald dieser geladen ist, "mischen" Sie "zeile" in diesen Teil hinein. Also:

```
MERGE "zeile".
```

Vergewissern Sie sich, daß der Teil korrekt in das Listing des ersten Teiles übernommen wurde und speichern Sie das dadurch erhaltene Programm unter dem alten Namen oder auch unter einem neuen wieder ab. Fertig. Ihr neuer Basicteil wird dann beim Programmlauf, wenn erforderlich sein sollte, die beiden Maschinenprogramme einlesen.

Beim Kassettenbetrieb müssen die Programme in folgender Reihenfolge abgespeichert sein: Basicteil, M-Code für die Dumpausgabe, M-Code für die Hardcopy

```
(MON1.BAS,MON1.BIN,HC.BIN).
```

Wenn Sie nun an diesem Punkt angelangt sind, verfügen Sie zwar schon über ein äußerst leistungsfähiges Programm für Ihre weiteren Untersuchungen, nur Ausdrucken können Sie noch nicht. Eine Ausgabemöglichkeit - aus dem Auswahlmenu heraus - ist in diesem Programm noch nicht enthalten und auch der Aufruf der Druckroutine ist noch nicht "eingebaut".

Aber auch dies kann schnell geschehen, indem Sie die entsprechende Basic-Zeile noch hinzufügen.

Beispiel:

```
340 IF LOWER$(x$)="d" THEN CALL &A16C,0,400:GOTO 300
```

Ein klein wenig Denk- und Programmierarbeit dürfen Sie nun noch selbst aufwenden, indem Sie auch einen entsprechenden Hinweis im Menü unterbringen. Sie wollen doch hoffentlich mitarbeiten?

Mit dem Programm arbeiten und ausdrucken können Sie allerdings auch ohne eine entsprechende Menüausgabe. Sie brauchen nur die Taste "d" zu drücken, dann wird "hardkopiert".  
(Wir haben aber im Menüfeld extra noch Platz gelassen, damit Sie den entsprechenden Hinweis noch hinzufügen können.)

Wer bei den Hardcopies nicht immer den gesamten "Schirm" zu Papier gebracht haben will, sollte in Zeile 340 die untere Pixelreihe mit 100 und die oberste mit 350 festlegen. Dadurch wird der Ausdruck des Kopf- und Fußteiles unterlassen und eine lückenlose Aneinanderreihung der Dumps ist möglich.

#### 4.3.10 Die Bedienung des Monitor-Programmes.

Nach dem Programmstart sehen Sie im obersten Bildschirmteil - eingerahmt zwischen den beiden inversen Balken - die hexadezimale Zahlenreihe von 00 bis 0F.

Diese Zahlenreihe soll Ihnen helfen, die genaue Speicherstelle zu erkennen. Der Beginn der Reihe ist von der Startadresse abhängig.

Der Schnittpunkt dieser Reihe mit der Adressenangabe im Dumpfeld ist die jeweilige Adresse.

Beachten Sie dabei aber, daß dann, wenn der Beginn nicht auf Null liegt (letzte Ziffer der Adresse), hinter &FF immer Hex &10 addiert werden muß.

Rechts neben dieser Zahlenfolge sehen Sie welcher Status der Speicherkonfiguration ausgewählt ist. Der Wert 255 heißt, daß der Monitor die RAM-Speicherstellen zeigt.

Im mittleren Teil sehen Sie links die jeweilige Adresszeile, dann invertiert dargestellt die hexadezimalen Inhalte und rechts die Darstellung in ASCII.

Steuerzeichen werden allerdings ausgeblendet und durch einen mit-tigen Punkt ersetzt.

Nachstehende Hardcopy (Abbildung 3), zeigt das Bild, das sich Ihnen nach dem Start des Programmes zeigt.

Im unteren Teil ist das Menüfeld. Das Programm bietet Ihnen die dort genannten Möglichkeiten.

Mit den Cursor-Steuertasten können Sie entweder zeilen- oder seitenweise "blättern".

Nach Betätigung der Leertaste (Spacetaste) können Sie eine neue Startadresse eingeben. Geben Sie nur Return oder Enter ein, dann bricht das Programm mit einer Fehlermeldung ab.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Status : 255
A000	C3	0F	A0	00	01	FF	00	00	A0	00	A1	F8	BF	00	A0	FE	1. . . . .
A010	03	C0	DD	22	0B	A0	01	06	00	2A	0B	A0	11	03	A0	ED	1. . . . .
A020	B0	2A	07	A0	22	0D	A0	ED	5B	03	A0	19	22	09	A0	3A	1. . . . .
A030	05	A0	32	3B	A0	DF	39	A0	C9	3C	A0	FF	3E	1E	CD	5A	1. . . . .
A040	BB	CD	B7	A0	CD	C6	A0	CD	CC	A0	2A	07	A0	06	10	CD	1. . . . .
A050	D2	A0	CD	C6	A0	CD	B7	A0	CD	C6	A0	CD	41	A1	CA	7A	1. . . . .
A060	A0	CD	05	A1	2A	0D	A0	CD	1D	A1	22	0D	A0	CD	41	A1	1. . . . .
A070	CA	7A	A0	23	CD	C1	A0	C3	5B	A0	CD	C1	A0	CD	B7	A0	1. . . . .
A080	3E	01	CD	90	BE	3E	00	CD	96	BE	CD	4B	A1	2A	07	A0	1. . . . .
A090	CD	41	A1	CA	A6	A0	06	10	CD	59	A1	CD	C1	A0	ES	CD	1. . . . .
A0A0	50	A1	E1	C3	90	A0	3E	14	CD	72	BB	CD	C1	A0	3E	07	1. . . . .
A0B0	CD	5A	BB	CD	7B	BB	C9	06	50	3E	0F	CD	5A	BB	10	EB	1. . . . .
A0C0	C9	3E	0A	CD	5A	BB	3E	0D	CD	5A	BB	C9	3E	0A	CD	5F	1. . . . .
A0D0	BE	C9	7D	E6	0F	CD	E2	A0	CD	FF	A0	2C	10	F4	CD	C1	1. . . . .
A0E0	A0	C9	4F	0F	0F	0F	0F	E6	0F	CD	F3	A0	79	E6	0F	CD	1. . . . .
A0F0	F3	A0	C9	FE	0A	38	02	C6	07	C6	30	CD	5A	BB	C9	3E	1. . . . .

Seite vor = Cursor rechts

Seite zurueck = Cursor links

Zeile vor = Cursor nach unten

Zeile zurueck = Cursor hoch

Leertaste = Aenderung der Adresse S = Status

E = Ende

Abbildung 3: Die Darstellung mit Menü

Damit Sie mit dem Programm etwas vertraut werden, spielen Sie alle gegebenen Möglichkeiten einfach einmal durch.

Denken Sie aber bitte daran, daß dann, wenn Sie die Taste "d" drücken ein Drucker sowohl angeschlossen als auch eingeschaltet sein muß, denn sonst "wartet" Ihr CPC auf die nicht erfolgreiche Druckerrückmeldung und arbeitet nicht weiter.

Zum Status wollen wir nun anmerken, daß Sie z.B. folgende Auswahlmöglichkeiten haben.

- 0 = eingebauter Basic-Interpreter ab &C000 alles andere RAM
- 1-6 = evtl. angeschlossene Erweiterungs-ROMs für die Adressen 1 bis 6 im Bereich von &C000 bis &FFFF, sonst RAM
- 7 = Disketten-ROM im Bereich ab &C000, sonst RAM
- 252 = unteres Firmware-ROM im Bereich von &0000 bis &3FFF  
RAM von &4000 bis &BFFF  
BASIC-ROM von &C000 bis &FFFF
- 254 = &0000 bis &3FFF unteres ROM  
&4000 bis &BFFF RAM  
&C000 bis &FFFF RAM
- 255 = Alles RAM

Wie Sie sehen, können Sie in allen Bereichen von ROM und RAM beliebig "herumstöbern".

## 4.4 Erste Analysen

Wir wollen Ihnen nun aufzeigen, was Sie beispielsweise untersuchen können und vor allem auch wo Ihre Suche beginnen kann. Und deshalb laden Sie nun bitte den Basisteil des Programmes - falls es nicht mehr im Speicher steht - und starten mit RUN.

Auf der Diskette müssen auch die Binärteile vorhanden sein. Die Kassettenbenutzer bitte daran denken, entweder später eine "getrennte" Kassette einzulegen oder hinter dem Basisteil müssen sich die Maschinencodeteile, also nicht die Hexlader befinden.

Nach dem Programmstart sehen Sie einen Speicherausgang, der beim Basic-Start beginnt. Siehe Abbildung 4.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Status : 255
0170	26	00	64	00	01	C0	20	4D	4F	4E	31	2E	62	61	73	20	&d.. MONI.bas
0180	20	3D	46	65	78	2D	20	75	6E	64	20	41	73	63	63	69	=Hex- und Ascci
0190	2D	44	75	6D	70	00	07	00	6E	00	01	C0	00	0E	00	78	-Dump...n...x
01A0	00	40	20	10	01	AA	20	1C	FF	9F	00	13	00	82	00	0D	...q...+...0..
01B0	0A	00	73	74	61	74	75	F3	20	EF	20	19	FF	00	24	00	..statu...+...\$
01C0	3C	00	A1	20	FF	12	29	1C	00	40	29	F2	1C	C3	00	20	...+...+...+...
01D0	EB	20	A8	22	6D	6F	6E	31	2E	62	69	8E	22	2C	1C	00	q"monl.bin",..
01E0	A0	00	13	00	98	00	0D	19	00	61	64	72	65	73	73	E5	...r...address
01F0	EF	1C	70	01	00	28	00	A0	00	0D	27	00	6C	61	65	6E	...p...^...laen
0200	67	E5	EF	1C	FF	00	01	0D	27	00	6C	61	65	6E	67	E5	g...+...laeng
0210	EF	0D	27	00	6C	61	65	6E	67	E5	F4	0F	00	07	00	AA	...laeng...a
0220	00	01	C0	00	32	00	B4	00	A1	20	0D	19	00	61	64	72	...2...adr
0230	65	73	73	E5	20	F1	0E	20	EB	20	00	00	00	61	64	72	ess...q...adr
0240	65	73	73	E5	EF	0D	00	00	61	64	72	65	73	73	E5	20	ess...address
0250	F4	10	F8	19	10	00	36	00	BE	00	A1	20	0D	19	00	61	...6...a
0260	64	72	65	73	73	E5	20	F0	10	F8	19	10	20	EB	20	0D	dress...+...q...

Seite vor = Cursor rechts      Seite zurueck = Cursor links  
 Zeile vor = Cursor nach unten      Zeile zurueck = Cursor hoch  
 Leertaste = Aenderung der Adresse      \$ = Status

B. 2.10.6

Abbildung 4: Sie sehen einen Dump ab dem Basic-Start

Sie können nun die ersten Analysen über Ihr Basic-Programm machen und feststellen, wie ein Basic-Programm codiert wird.

Dazu brauchen Sie nun zunächst die folgenden Informationen.

Der Beginn des Basic-Programmstarts liegt üblicherweise (es kann aber auch anders sein, wenn die Basic-Zeiger verstellt wurden!) bei &170.

Die ersten beiden Bytes geben die Zeilenlänge an. Denken Sie bitte immer "HEXADEZIMAL".

Die Zeilenlänge ist &0026. Das heißt, die erste Basic-Zeile ist &26 = 38 Zeichen lang.

Eine Basic-Zeilenlänge kann zwar eigentlich nie länger als 255

Zeichen sein, deshalb würde auch eine einzige Speicherstelle ausreichen, in der die Zeilenlänge steht, aber wenn Sie beispielsweise einmal dem Programm eine Zeile 1 voranstellen, und in diese solange das Fragezeichen im Wechsel mit dem Doppelpunkt eingeben, bis Ihr CPC piepst und dann die Enter-Taste betätigen, dann sehen Sie die ersten beiden Bytes belegt.

Eine komprimiert eingegebene Zeile kann also durchaus länger als 255 Zeichen sein, bzw. der CPC benötigt manchmal mehr Bytes als eingegeben zur Abspeicherung. Siehe Abbildung 5.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Status : 255
0170	03	01	01	00	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	...Q.Q.Q.Q.Q.Q.
0180	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
0190	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
01A0	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
01B0	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
01C0	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
01D0	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
01E0	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
01F0	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
0200	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
0210	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
0220	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
0230	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
0240	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
0250	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.
0260	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	BF	01	Q.Q.Q.Q.Q.Q.Q.Q.

Seite vor = Cursor rechts      Seite zurueck = Cursor links  
 Zeile vor = Cursor nach unten      Zeile zurueck = Cursor hoch  
 Leertaste = Aenderung der Adresse      S = Status      E = Ende

Abbildung 5: Sie sehen die Überlange Basic-Zeile

Wenn Sie nach der Direkteingabe dieser Zeile das Programm starten, dann läuft es ohne Probleme ab, obwohl der CPC bei LIST 1 zeigt, daß diese Zeile nicht komplett übernommen wurde. Am Ende der Zeile steht völlig einsam ein "P". Eigentlich müsste dies zu einem Syntax-Error führen. Dies geschieht aber erst, wenn Sie in dieser Zeile etwas ändern.

Entfernen Sie nun diese mühsam eingegebene Zeile 1 wieder und starten Sie das Programm durch RUN.

Als nächstes folgt in den Speicheradressen &172 und &173 die Zeilennummer. Diese hat den Wert &64 = 100.

In den nächsten beiden Speicherstellen (&174,&175) sehen Sie den Code für das Apostrophzeichen und der darauf folgende Wert &20 ist ein Space, also ein Leerzeichen.

#### 4.5 Die Token

Haben Sie anstelle des Apostroph ein REM gesetzt, dann steht in der Speicherstelle &175 der Wert &c5. Es gibt aber keine drei Speicherstellen, in denen das Wort REM zu finden wäre. Basic-Befehls- worte werden im Speicher der CPCs als sogenannte TOKEN ab- gelegt. Diese Token sind Einbytewerte, die für einen Basic-Befehl im Speicher abgelegt werden. Also eine Art "Stenokürzel" für Com- puter.

Da der CPC-Basic-Befehlssatz aber sehr umfangreich ist, und auch durch den Aufbau des Interpreters bedingt, werden verschiedene Befehle auch durch 2 Bytes codiert. Diese haben also auch noch ein "Vortoken".

Wer nun Unterschiede zwischen seiner Bildschirmdarstellung und unserer Hardcopy bzw. unseren Angaben festgestellt hat, dem kön- nen wir nur sagen, daß sein Programm nicht mit unserem überein- stimmt. Falls es einwandfrei läuft wurde nur anders programmiert, so zum Beispiel ein Space mehr eingeben oder ähnliches.

Aus diesem Grunde können wir nun keine genauen Adressangaben mehr machen, denn diese würden bei vielen Lesern abweichen. Deshalb nun ohne Adressangabe weiter.

Nach dem schon genannten Space, ist die in ASCII abgelegte Bemer- kung zu sehen. Betrachten Sie hierzu auch die ASCII-Darstellung im rechten Bereich des mittleren Teiles.

Nochmals zurück zur Zeilenlänge. Wer nun, bei der ersten Basic- Speicherstelle beginnend, die Anzahl an Speicherstellen abzählt, die in &170 steht, wird dann in der erreichten Speicherstelle eine Doppelnulld finden. Dies ist für den CPC, bzw. dessen Basic- Interpreter das Kennzeichen, daß eine Basic-Zeile zu Ende ist. Unmittelbar dahinter beginnt die nächste Basic-Zeile.

Das ganze Spiel beginnt dann wieder von vorne, bis zum Programm- ende. Dieses ist durch weitere zwei leere Speicherstellen gekenn- zeichnet. Das heißt, daß an einem Programmende insgesamt drei Bytes mit den Inhalten "00" stehen.

Fassen wir zusammen. Ein Basic-Programm beginnt bei &170 und ist nach folgendem Schema aufgebaut:

```
Zeilenlänge, Zeilennummer, Statement 1: Statement2:...Zeilenende
.....
Zeilenlänge, Zeilennummer, Statements...:Zeilenende,Programmende.
```

Mit diesem Wissen können Sie nun schon ein kleines Experiment machen. Die erste Zeilennummer muß zwangsläufig in den Speicher- stellen &172 und &173 abgelegt sein.

##### 4.5.1 Eine Zeile läßt sich nicht listen

Poken Sie in &172 doch einfach einmal eine Null und lassen Sie das Programm dann listen. Wie Sie sehen werden, fehlt dann die erste Zeile. Das können Sie mit allen Zeilennummern tun wenn Sie

wollen und manch einer wird nun sagen: Wunderbar, ein kleiner Dreh um ein Programm vor dem Einblick anderer zu schützen! Doch weit gefehlt. POKE n Sie die Zahl &64 wieder als erste Zeilennummer, dann ist die Zeile wieder aufzulisten.

Noch einfacher geht es aber durch den Basic-Befehl RENUMBER!

Also, so können wir nicht schützen, obwohl es von vielen trotz allem so gemacht wird. Diese haben aber anscheinend den Umstieg von einem Computer eines anderen Herstellers noch nicht geschafft. Außerdem sehen Sie ja, nachdem Sie sich nur ein klein wenig mit Ihrem CPC vertraut gemacht haben, daß dies wirklich nur eine Spielerei ist.

Sie können nun bereits ein wenig selbst herumspielen, welche Auswirkungen es hat, wenn verschiedene Speicherinhalte durch "POKE n" verändert werden.

Nachdem Sie Ihre Programme ja hoffentlich schon abgespeichert haben, ist es nicht weiter tragisch, wenn Ihre Aktivitäten das Programm vollkommen durcheinanderschmeißen. Ändern Sie ruhig einmal die Zeilenlänge der ersten Zeile, um zu sehen, was dann passiert.

Die Zahl Null in diese Speicherstelle gePOKEt, selbst wenn Sie dann wieder die Ursprüngliche einschreiben, hat fatale Wirkungen.

Starten Sie dann aber trotzdem das Programm einmal mit RUN!

Wenn Sie sich genügend damit amüsiert haben, dann können wir wieder ernsthaft weitermachen.

Wir haben schon angedeutet, daß Basic-Schlüsselworte durch Token codiert werden. Ein Basic-Programm wird also bei der Eingabe bereits "tokenisiert" (engl.: tokenist).

Durch einen kleinen Einzeiler und durch "POKE n" in die Speicherstelle &174 können Sie dies sich ja mal etwas näher ansehen. Ein Großteil der Token hat Werte über 127. Beginnen Sie einfach einmal mit dem Wert 128 (=After).

So könnte der Einzeiler aussehen:

```
10 REM*****
```

Und nun:

```
POKE &174,128:LIST
```

usw.

Wie schon erwähnt gibt es auch noch solche mit Vortoken. Auch hierfür ein Beispiel:

```
POKE &174,&FF:POKE &175,1
```

Als nächstes wollen wir uns mit den Variablen beschäftigen, denn auch diese werden codiert abgelegt. Am einfachsten ist es nun, dem Programm MON1.BAS einfach zusätzliche Programm-Zeilen voranzustellen, in denen verschiedene Variablen definiert und Inhalte (Werte) zugewiesen werden.

Mit dem Monitor selbst können Sie dann alles untersuchen.

Hier tritt nun aber eine Besonderheit ein, die Sie vielleicht nicht so schnell bemerkt hätten, und die auch gar nicht so einfach zu bemerken ist.

Im Gegensatz zu anderen Computern wird beim CPC eine Methode angewandt, die man auch gar nicht so ohne weiteres durchschauen kann.

Um Ihnen das zu zeigen müssen wir einen kleinen Programmiertrick anwenden, denn sonst wird es schwierig, Ihnen das zu erklären. Sie können dies dann den nachfolgenden Abbildungen entnehmen.

Vor den Monitor bringen wir die nachstehenden Basic-Zeilen ein.

```
10 a%=153:a$="3":a!=2:PRINT FRE(0)
20 PRINT HEX$(a a%),HEX$(a a%),HEX$(a a!)
50 REM ***** (Die Zeile 50 hat etwa 80 *)
```

Um die Unterschiede deutlich zu machen, nun bitte folgende Befehle im Direktmodus eingeben:

```
FOR I = &170 TO &1BA: POKE I-&70,PEEK(i):NEXT I
```

Dadurch wird bewirkt, daß die beiden vorangestellten Basic-Zeilen ab &100 aufwärts so abgelegt werden, wie sie nach der Eingabe im Speicher stehen. Dieser Ablagebereich unterhalb des Basic-Programmes dient nur als "Merkzettel" für spätere Vergleiche.

Wenn der Monitor nun mit RUN 100 gestartet wird, dann werden die Zeilen davor ja nicht abgearbeitet. Das heißt Sie wurden vom CPC so belassen, wie sie ursprünglich waren.

Zum Beweis stellen Sie bitte den Monitor z.B. mittels Cursor-Steuerung oder auch im Programm selbst (in Zeile 160) auf &100 (Siehe Abbildung 6).

Wenn Sie nun die Inhalte der Speicherstellen ab &100 und ab &170 miteinander vergleichen, dann werden Sie keine Unterschiede feststellen.

Und nun starten Sie bitte das Programm nicht mit RUN 100 sondern mit RUN. Auf den ersten Blick übersieht man nun evtl. sogar die Unterschiede (Auf &100 stellen!).

Doch bei genauerem Hinsehen ist nun doch etwas sehr interessantes festzustellen, (Abbildung 7).

Wenn Sie nun bitte wieder den Vergleich durchführen, werden Sie sehen, daß beispielsweise die Speicherstelle &175 nicht mehr mit



der Speicherstelle &105 übereinstimmt, obwohl dies aber doch sein müsste. Gleiches gilt auch für &17d und &100d oder &186 und &116!

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Status : 255
0100	24	00	0A	00	02	00	00	E1	EF	19	99	01	03	00	00	E1	\$.....01.J.....0
0110	EF	22	33	22	01	04	00	00	E1	EF	10	01	BF	20	FF	09	1"3"....01..0 +.
0120	28	0E	29	00	26	00	14	00	BF	20	FF	73	28	40	20	02	(.)..&...0 +s(0 .
0130	00	00	E1	29	2C	FF	73	28	40	03	00	00	E1	29	2C	FF	..0), +s(0...0), +
0140	73	28	40	20	04	00	00	E1	29	00	57	00	00	00	00	00	s(0 ...0).H.....
0150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0170	24	00	0A	00	02	00	00	E1	EF	19	99	01	03	00	00	E1	\$.....01.J.....0
0180	EF	22	33	22	01	04	00	00	E1	EF	10	01	BF	20	FF	09	1"3"....01..0 +.
0190	28	0E	29	00	26	00	14	00	BF	20	FF	73	28	40	20	02	(.)..&...0 +s(0 .
01A0	00	00	E1	29	2C	FF	73	28	40	03	00	00	E1	29	2C	FF	..0), +s(0...0), +
01B0	73	28	40	20	04	00	00	E1	29	00	57	00	32	00	C5	20	s(0 ...0).H.2.)
01C0	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	*****
01D0	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	*****
01E0	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	*****
01F0	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	*****

Seite vor = Cursor rechts      Seite zurueck = Cursor links  
 Zeile vor = Cursor nach unten      Zeile zurueck = Cursor hoch  
 Leertaste = Aenderung der Adresse      S = Status      E = Ende

Abbildung 6: Die "Basic-Zeilen" sind identisch!

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Status : 255
0100	24	00	0A	00	02	00	00	E1	EF	19	99	01	03	00	00	E1	\$.....01.J.....0
0110	EF	22	33	22	01	04	00	00	E1	EF	10	01	BF	20	FF	09	1"3"....01..0 +.
0120	28	0E	29	00	26	00	14	00	BF	20	FF	73	28	40	20	02	(.)..&...0 +s(0 .
0130	00	00	E1	29	2C	FF	73	28	40	03	00	00	E1	29	2C	FF	..0), +s(0...0), +
0140	73	28	40	20	04	00	00	E1	29	00	57	00	00	00	00	00	s(0 ...0).H.....
0150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0170	24	00	0A	00	02	00	00	E1	EF	19	99	01	03	00	00	E1	\$.....01.J.....0
0180	EF	22	33	22	01	04	12	00	E1	EF	10	01	BF	20	FF	09	1"3"....01..0 +.
0190	28	0E	29	00	26	00	14	00	BF	20	FF	73	28	40	20	02	(.)..&...0 +s(0 .
01A0	05	00	E1	29	2C	FF	73	28	40	03	00	00	E1	29	2C	FF	..0), +s(0...0), +
01B0	73	28	40	20	04	12	00	E1	29	00	57	00	32	00	C5	20	s(0 ...0).H.2.)
01C0	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	*****
01D0	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	*****
01E0	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	*****
01F0	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	*****

Seite vor = Cursor rechts      Seite zurueck = Cursor links  
 Zeile vor = Cursor nach unten      Zeile zurueck = Cursor hoch  
 Leertaste = Aenderung der Adresse      S = Status      E = Ende

Abbildung 7: Nur kleine Unterschiede im Speicher

Sie sehen, durch den Programmlauf haben sich die ursprünglich eingegebenen Basic-Zeilen, d.h. Ihre abgespeicherten Inhalte verändert.

Still und heimlich hat der CPC an "unseren Programmzeilen" Änderungen durchgeführt. Welcher Sinn steckt da dahinter? Das erfahren Sie dann im nächsten Abschnitt, betrachten Sie sich zunächst erst einmal die Abbildungen.

## 5. Einblick in den RAM-Speicher

### 5.1 Die Programmablage

Im Gegensatz zu manchen anderen Computern besitzt der CPC keinen sogenannten Fullscreen-, sondern einen Line-Editor.

Dieser Line-Editor nimmt Eingabezeilen auf und übergibt diese dann erst in den dafür vorgesehenen Speicherbereich. Begeben wir uns mit dem Monitor einmal auf die Suche nach dem Speicherbereich, in dem der Editor die Eingaben zwischenspeichert. Wenn wir bei Null anfangen kommen wir sehr schnell zu einem Ergebnis.

Ab &40 rührt sich etwas. Dort sind immer Reste von dem zu sehen, was wir vorher eingegeben haben. Mittels des kleinen Monitors können Sie das ganz hervorragend beobachten.

Ändern Sie deshalb die Zeile 150 bitte derart ab, daß die Startadresse für die Dumpausgabe &0040 ist.

Nach dieser Änderung geben Sie ohne irgendwelche Zeilennummer so lange das gleiche Zeichen z. Beispiel "a" ein, bis Sie der CPC Sie durch seine Tonausgabe darauf aufmerksam macht, daß er nun wirklich keinerlei Eingaben (also Zeichen) mehr annehmen will.

Dann betätigen Sie die Enter-Taste und starten das Programm durch Eingabe von RUN.

Der zwischenzeitlich gemeldete Syntax-Error soll Sie nicht stören. Viel interessanter ist, was Sie dann auf dem Bildschirm zu sehen bekommen. Von &44 bis einschließlich &140 sehen Sie das Zeichen, bzw. den Code des Zeichens, den Sie bis zum akustischen Signal des CPC eingegeben haben.

Weil nach dieser Eingabe aber auch noch der Befehl RUN gegeben wurde, steht in den ersten Bytes der Code (Token) für RUN (&CA) und je nachdem was Sie noch so alles eingegeben haben. (Wir wissen ja nicht, wie genau Sie diese Anregungen durchführten. Wenn Sie lauter Doppelpunkte eingeben und am Ende dann den Befehl RUN, dann sieht es ja wieder anders aus!)

Wichtig ist, daß Sie nun wissen, daß sich dort ein wichtiger Bufferbereich befindet. Dies heißt aber gleichzeitig, daß Basic-Programme unterhalb von &170 möglichst nichts zu POKEN haben, obwohl auch dies möglich ist, wie wir Ihnen schon gezeigt haben und später nochmals zeigen werden.

Durch kleine Test können Sie nun selbst feststellen, ob dieser Editorbuffer auch bei Eingaben während des Programmlaufes benutzt wird. Über den Menüpunkt "Änderung der Adresse" ist dies ja leicht möglich.

Damit Sie es etwas leichter haben, sollten Sie den Buffer mit irgendeinem Zeichen "auffüllen", denn dadurch werden "Erkenntnisse" erleichtert. Wie dies geht hatten wir vorher schon gezeigt.

Wie Sie unschwer feststellen werden, ist dort von Eingaben während eines Programmlaufes überhaupt nichts zu sehen, deswegen muß irgendwo noch ein Buffer sein, der Zeichen aufnimmt.

Je nachdem welchen CPC Sie besitzen müssen Sie nun bei &ACA4 (CPC 464) oder bei &AC8A nachsehen.

Auch hier können Sie sich mit einem kleinen Trick eine bessere Übersicht verschaffen. Über den Menüpunkt Änderung der Adresse und nach Eingabe der gewünschten Adresse (&ACA4 oder &AC8A) quittieren Sie noch nicht mit der ENTER-Taste, sondern drücken solange die Leertaste bis der CPC wieder einmal akustisch seinen Unwillen kundtut. Erst dann betätigen sie die ENTER-Taste.

Bei der darauf folgenden Ausgabe wird zwar - bedingt durch das "scrollen" des Bildschirms - die Ausgabemaske zerstört, aber was Sie erkennen können, sollte über die zerstörte Ausgabemaske hinwegtrösten. Links oben angefangen (bei der ASCII-Ausgabe), sehen Sie im Klartext den von Ihnen eingegebenen Wert und anschließend lauter "Spaces".

Durch eigene Experimente werden Sie nun ganz schnell feststellen, daß dies der wirkliche Edit-Buffer ist, der andere Buffer war nämlich "nur" der Buffer für den Programm-Editor. Bitte verzeihen Sie uns den kleinen Umweg. (Auch wir hatten bei der Suche danach erst den Buffer für den Programm-Editor gefunden. Und bei unseren ersten Analysen gab es noch kein Firmware-Handbuch und auch noch keine kommentierten ROM-Listings.)

Aber erst mit diesem Wissen sind Sie, falls Sie es nicht schon vorher wußten, soweit, zu verstehen, was bei der Eingabe einer Basiczeile abläuft.

Eine eingegebene Zeile wird also schon bei der Übernahme aus dem Edit-Buffer in den Buffer des Programm-Editors teilerinterpretiert. Das heißt, alle ihre Eingaben werden überprüft, und dann bereits tokenisiert an den Editorbuffer bei &40 übergeben.

Nun wieder zurück zum Ausgangspunkt, eigentlich wollten wir ja untersuchen, wie ein Basicprogramm abgelegt wird. Sehen Sie sich deshalb bitte wieder die Abbildungen 6 und 7 an.

Wie Sie wissen, werden in diesen beiden Bildern zwei Programmzeilen (der Rest ist unwichtig) gezeigt und zwar einmal vor einer selbständigen Veränderung durch den CPC und einmal hinterher.

Die Veränderung dieser Zeilen im Speicher erfolgte durch den Programmlauf.

Jede Basic-Programmzeile wurde bei der Eingabe bereits "vorbereitet", das heißt, daß Basic-Schlüsselwörter durch die jeweiligen Token ersetzt wurden. (Dies spart einmal Speicherplatz und ist außerdem beim Lauf eines Programmes schneller "abzuarbeiten".)

Warum aber veränderte der CPC unser bereits doch schon tokenisiertes Programm beim Programmlauf?

Damit Sie dies nun besser verstehen, wenden wir wieder einen kleinen Trick an.

Dazu sollten die Maschinenspracheteile des Monitors im CPC stehen, denn diese werden nun benötigt. Allerdings nicht der Basic-teil.

Wer den Monitor bereits komplett geladen hat, braucht deshalb nur NEW einzugeben, denn durch diesen Befehl wird zwar ein Basic-Programm gelöscht, aber nicht geschützte Maschinenprogramme. Wer den Monitor noch nicht geladen hat, lädt und startet diesen. Über den Menüpunkt "E" dann aus dem Programm aussteigen und ebenfalls durch NEW löschen.

Damit ist nun gewährleistet, daß die Speicherstellen ab &170 aufwärts, bis zu HIMEM auf Null gesetzt wurden aber die Maschinenprogramme sind noch vorhanden.

Nun müssen folgende Zeilen eingegeben werden:

```
1 aaa%+=32767
2 bbb%=-32767
3 bbbbb1%=1
4 bbbbb2%=-1
5 aaaaa%+=154
6 aaaab%+=155
7 aaaaa!=155
8 aaaaa=155
9 a=65334.122:.....
10 b=653.34122:.....
11 a$="test"
```

Bitte dieses Programm nun noch nicht starten, sondern zunächst erst weiterlesen!

Dadurch, daß die beiden Maschinenspracheroutinen zur Verfügung stehen, können Sie sich nach dem Prinzip "vorher/nachher" ansehen, wie diese Zeilen abgelegt werden und dann auch wie sie nach dem Befehl RUN im Speicher stehen.

Wenn Sie obiges Programm genauso abgeschrieben haben, wie es dort steht, dann sollten Sie durch die Befehlssequenz:

```
MODE 2:CALL &A000,&170,255,256
```

exakt die Darstellung bekommen, die in Abbildung 8 zu sehen ist.

Ist dies nicht der Fall, dann bestehen zwischen Ihren Eingaben und unseren Angaben Unterschiede.

Entweder korrigieren Sie dann die Zeilen, oder sie verfolgen es in diesem Buch.

Wer ebenfalls gleich ausdrucken will, muß an die obengenannte Befehlssequenz noch folgenden Zusatz anfügen: ":CALL &A16B,0,400"

Nun können die Analysen beginnen

Zum Vergleich betrachten Sie bitte die Abbildungen 8 und 9.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
0170	10	00	01	00	02	00	00	61	61	E1	EF	F4	1A	FF	7F	00	.....aa
0180	10	00	02	00	02	00	00	62	62	E2	EF	F5	1A	FF	7F	00	.....bb
0190	10	00	03	00	02	00	00	62	62	62	62	62	B1	EF	0F	00	.....bbbb
01A0	10	00	04	00	02	00	00	62	62	62	62	B2	EF	F5	0F	00	.....bbbb
01B0	10	00	05	00	02	00	00	61	61	61	61	E1	EF	19	9A	00	.....aaaa
01C0	10	00	06	00	02	00	00	61	61	61	61	E2	EF	19	9B	00	.....aaaa
01D0	10	00	07	00	04	00	00	61	61	61	61	E1	EF	19	9B	00	.....aaaa
01E0	10	00	08	00	00	00	00	61	61	61	61	E1	EF	19	9B	00	.....aaaa
01F0	20	00	09	00	00	00	00	E1	EF	1F	3B	1F	36	7F	90	01	.....
0200	01	C0	27	27	27	27	27	27	27	27	27	27	27	27	27	00	.....
0210	20	00	0A	00	00	00	00	E2	EF	1F	3C	D6	55	23	8A	01	.....
0220	01	C0	27	27	27	27	27	27	27	27	27	27	27	27	27	00	.....
0230	10	00	0B	00	03	00	00	E1	EF	22	74	66	73	74	22	00	.....
0240	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0250	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0260	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Abbildung 8: Aufruf durch MODE 2:CALL &A000,&170,255,256

Von &170 bis &1E0 sehen Sie in den jeweils ersten beiden Bytes die Länge der einzelnen Programmzeilen. Jede Zeile ist 16 Zeichen lang. Die Längen der weiteren Zeilen sehen Sie in &1F0,&1F1, in &210,&211 und in &230,&231. Die Zeilennummern sehen Sie in den jeweils immer folgenden zwei Bytes.

Dadurch, daß wir die Programmzeilen in unmittelbar aufeinanderfolgender Numerierung haben, sind diese, deren Anfänge und auch die Enden leicht zu erkennen. Bedingt durch die Ablage der Variablen in den Zeilen zehn und elf, mußten - um den Überblick zu behalten - diese Zeilen mit Hochkommas angepaßt werden.

Durch die verschiedenen Variationen unserer Variablenzuweisungen ist nun sehr schön zu erkennen, welche Kennzeichen die einzelnen Variablentypen haben. Diese stehen unter der Spalte vier der Kopfzeile jeweils untereinander. Beachten Sie aber auch den größeren Speicherbedarf dieser Zeilen!

# Obersicht der Typenkennzeichnungen

INTEGER-	Variable	(Kennzeichen %)	mit "02".
STRING-	Variable	(Kennzeichen \$)	mit "03".
REAL-	Variable	(Kennzeichen !)	mit "04".
Unmarkierte Variable			mit "0D".

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
0170	10	00	01	00	02	07	00	61	61	E1	EF	F4	1A	FF	7F	00	.....aaB1A+*
0180	10	00	02	00	02	0F	00	62	62	E2	EF	F5	1A	FF	7F	00	.....bbB1A+*
0190	10	00	03	00	02	1A	00	62	62	E2	EF	F5	1A	FF	7F	00	.....bbbbbB1A+*
01A0	10	00	04	00	02	24	00	62	62	E2	EF	F5	1A	FF	7F	00	.....\$bbbbB1A+*
01B0	10	00	05	00	02	2E	00	61	61	E1	EF	19	9A	00	00	00	.....\$aaaaB1A+*
01C0	10	00	06	00	02	38	00	61	61	E1	EF	19	9B	00	00	00	.....\$aaaaB1A+*
01D0	10	00	07	00	04	42	00	61	61	E1	EF	19	9B	00	00	00	.....BaaaaB1A+*
01E0	10	00	08	00	00	42	00	61	61	E1	EF	19	9B	00	00	00	.....BaaaaB1A+*
01F0	20	00	09	00	00	4B	00	E1	EF	1F	3B	1F	3B	7F	90	01	.....XBA1A+*
0200	01	C0	27	27	27	27	27	27	27	27	27	27	27	27	27	00	.....I+1A+*
0210	20	00	0A	00	00	54	00	E2	EF	1F	8C	D6	55	23	8A	01	.....I+1A+*
0220	01	C0	27	27	27	27	27	27	27	27	27	27	27	27	27	00	.....I+1A+*
0230	10	00	0B	00	03	5D	00	E1	EF	22	74	65	73	74	22	00	.....I+1A+*
0240	00	00	00	00	41	41	C1	01	FF	7F	00	00	42	42	C2	01	.....AA\BBB1A+*
0250	01	80	03	00	42	42	42	42	42	91	01	01	00	11	00	42	.....BBBBB1A+*
0260	42	42	42	92	01	FF	FF	01	00	41	41	41	41	C1	01	9A	BBB-AAA\BBB1A+*
0270	00	26	00	41	41	41	41	C2	01	3B	00	30	00	41	41	41	AAA\BBB1A+*
0280	41	C1	04	00	00	1B	88	3A	00	C1	04	3B	1F	3B	7F	00	AAA\BBB1A+*
0290	90	1C	00	C2	04	8C	D6	55	23	8A	47	00	C1	02	04	3A	AAA\BBB1A+*
02A0	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Abbildung 9: Aufruf durch MODE 2:CALL &amp;A000,&amp;170,255,320

Aber auch andere Kennzeichnungen sind möglich, nämlich dann, wenn durch DEFINT, DEFREAL oder DEFSTR erst bei der Programmausführung einer bisher nicht markierten Variablen eine definierte Zuweisung erfolgt. Diese Kennzeichen sind z.B. 05,0B,0C.

Durch eigene Experimente können Sie nun ja herausfinden, wann welche Kennzeichen benutzt werden. Wie es geht, haben wir ja gezeigt.

Zwischen diese Token und den Variablennamen wird erst beim Programmablauf ein "Displacement" (Abstand oder Offset) eingefügt, der dazu dient, den Variableneintrag schneller zu finden. Dies können Sie durch Abbildung 9 gut erkennen.

In beiden Abbildungen ist auch zu sehen, daß die Variablennamen so übernommen werden, wie eingegeben und nur beim letzten Zeichen das höchstwertige Bit gesetzt wird. Dies erkennt der CPC dann als Ende eines Variablennamens. Nach den Variablennamen folgen Bytes, an deren genauer Bedeutung wir selbst noch etwas knobeln, denn diese haben wir noch nicht sehr eingehend untersucht. Einiges ist uns zwar schon klar, aber eben nicht alles und deswegen sind die nachfolgenden Aussagen etwas mit Vorsicht zu betrachten.

So beinhalten z.B. die Bytes &17E und &17D zwar den Wert &7FFF, also 32767 wie im Programm eingegeben, aber wie die Werte in &17A und &17B nun ganz genau zu interpretieren sind, entzieht sich noch unseren Kenntnissen. &F4 scheint mit positivem und &F5 mit negativen Vorzeichen zusammenzuhängen. Die Werte &EF scheinen von der Größe der Zahl abhängig zu sein.

REAL-Werte werden mit Vorzeichen, Mantisse und Exponent abgelegt. Bei der Stringvariablen a\$, die nach dem Programmlauf Kennzeichen ab &29C hat, ist in dieser ersten Speicherstelle der Name zu finden. Allerdings wurde gegenüber der vorigen Kennzeichnung nun Bit 5 zurückgesetzt. Das nächste Byte scheint auf den Typ hinzuweisen. Das alles haben wir aber selbst noch nicht so genau untersucht! Ab jetzt sind wir uns wieder sicher!

Das nächste Byte gibt die Länge des Strings an. Die beiden nachfolgenden zeigen, wo der String im Speicher zu finden ist. Und nun kommt eine weitere Besonderheit. Entfernt man nämlich eine Zeile, oder fügt eine hinzu, dann wird der String in den oberen Speicher kopiert. Dies hat zur Wirkung, daß unmittelbar hinter dieser Angabe dann die Adresse zu finden ist, an die der String kopiert wurde.

Dies heißt aber gleichzeitig, daß "bearbeitete" Stringvariable unterhalb von HIMEM abgelegt werden und dann von oben nach unten, den anderen im Programm vorkommenden Variablen entgegenlaufen. Stringvariablen, die nicht bearbeitet wurden, werden nicht nach oben "geschrieben".

Wird eine Stringvariable weiterbearbeitet, dann wird der Eintrag (unterhalb von HIMEM) nicht geändert, sondern es erfolgt eine weitere Kopie unterhalb der ersten und der Zeiger im Bereich der Variableneinträge wird dann auf diese neue Adresse gestellt.

Ganz nebenbei bemerkt, zeigen all diese Tatsachen, daß Benchmarktests, wenn der Rechner nicht intern bekannt ist, mit großer Vorsicht zu genießen sind, denn was ist, wenn der eine Computer durch einen vorherigen Testlauf seine Ausgangsbedingungen schon zu seinem Vorteil verändert hat?

Was geschieht nun aber, wenn die - von oben nach unten laufenden - Stringvariablen den Ablage-Bereich der anderen Variableneinträge erreichen? Nun, dann kommt es zu der - auch bei anderen Computern - so gefürchteten Aktivierung der GARBAGE-COLLECT-Routine, die dazu führt, daß ein Computer scheinbar nichts mehr tut und oft für lange, lange Minuten "steht".

In dieser Zeit hat der Computer nichts anderes im Sinn, als seinen Speicher wieder in Ordnung zu bringen und die "Stringleichen" zu entfernen.

Amstrad hat diese Routine beim CPC 664 und CPC 6128 verbessert, denn während man bei großem Datenanfall, den CPC 484 sehr leicht für einige Minuten beschäftigt, ist bei den Nachfolgern - bei den gleichen Programmen - kein längerer "Stillstand" mehr feststellbar.



Mit dieser letzten Bemerkung über die Verbesserung bzw. Änderung der Stringmüll-Beseitigungsroutine, wollen wir Sie dann mit Ihren eigenen Untersuchungen über die Ablage eines Basic-Programmes alleine lassen und uns wieder anderen Dingen zuwenden. Das Werkzeug für Ihre eigenen Untersuchungen haben Sie ja.

### 5.2 Wir sehen uns im Speicher um

Nachdem wir nun wissen, wo der Basicstart und wo das Ende des frei benutzbaren RAM-Bereiches liegt, wollen wir uns noch etwas weiter im Adressbereich des Prozessors umsehen.

Als nächstes stellen wir erst einmal fest, wo sich der CPC denn den "Basicbereich" merkt, denn für alle für ihn wichtigen Werte hat er ja irgendwo seine Merkspeicherstellen.

Wenn wir es nicht schon wüssten und selbst erst suchen müssten, dann würden wir nun ein kleines Programm schreiben, das in zwei - unmittelbar aufeinanderfolgenden - Speicherstellen den Wert &170 in Low-/Highbyte-Konfiguration sucht. Diese Suche wäre aber nicht sehr erfolgreich, denn der CPC merkt sich den Basic-Beginn nicht mit dem Wert &170 sondern mit &16F! Hätten wir sie aber doch gefunden, würden wir in der Umgebung der gefundenen Adressen auch noch zwei Speicherstellen suchen müssen, in der die Werte für die oberste Programm-Speicherstelle stehen usw.

Tatsächlich haben wir zu der Zeit, als es noch kein Firmware-Handbuch und andere weiterführende Literatur gab, auch wirklich mittels kleiner Programme nach den verschiedensten "Merkern" gesucht und waren damit auch sehr erfolgreich.

Diese Mühe können wir uns aber ersparen, denn diese Informationen liegen vor und deshalb können wir nun unseren Monitor ganz gezielt auf diesen Bereich stellen.

CPC 464-Besitzer sehen nun bitte ab &AE7B und die anderen bei &AE5E nach. Denn ab diesen Adressen stehen einige interessante Zeiger.

Um nun allen CPC-Besitzern gleichermaßen "Schwierigkeiten" zu bereiten, nennen wir die eben genannten Adressen einfach "Grundadressen" (GA) und geben alles weitere relativ an.

Die Zeiger für HIMEM befinden sich in GA und GA+1.  
Die Zeiger für Ende des freien RAMs stehen in GA+2 und GA+3.  
Die Zeiger für LO-RAM stehen in GA+4 und GA+5.

Die nächsten Bytes (auch immer jeweils zwei) sind folgende Marker:

Programmstart  
Programmende  
Variablenstart  
Start der Felder,  
Ende der Felder.

Wir können Ihnen nun unmöglich alle RAM-Speicherstellen und deren Bedeutung an Beispielprogrammen aufzeigen, aber durch ein paar ausgewählte hoffen wir, Ihnen andeuten zu können, was mit "intimieren" Kenntnissen machbar ist.

Deshalb folgt nun ein Programm, das auf der Basis des bereits bekannten aufbaut. Ein Programm zur Entfernung von REM-Statement-Zeilen. Wir haben es KILLREM getauft, weil dieser Name deutlich aufzeigt, welchen Sinn diese Utilitie (Hilfsroutine) hat.

### 5.2.1 Listing: KILLREM.HEX

```

100 MEMORY &A4FF
110 REM killrem.hex
120 IF PEEK(&AC01)=&C9 THEN cpcflag=4
130 a= 42240:e= 42466:zb=2000
140 FOR i =a TO e:READ d$:IF LEFT$(d$,1)<>"&" THEN 170
150 IF ps<>VAL(d$) THEN PRINT"Fehler Zeile "zb:END
160 ps=d$:IF i=e THEN 230:ELSE i=i-1:zb=zb+1:GOTO 220
170 d$="&"&d$:ps=ps+VAL(d$)
180 IF cpcflag = 4 THEN 210
190 IF VAL(d$)=&81 THEN d$="&64"
200 IF VAL(d$)=&83 THEN d$="&66"
210 POKE i,VAL(d$)
220 IF i < e THEN NEXT i
230 PRINT"Soll das Programm 'KILLREM.BIN' abgespeichert werden ?"
240 x$=LOWER$(INKEY$):IF x$="" THEN 240
250 IF x$="j" THEN SAVE"killrem",b,&A500,&E1
260 END:-----
270 '
2000 DATA 21,B3,A5,7E,FE,00,CA,10,A5,CD,5A,BB,23,C3,03,A5,&07E4
2001 DATA CD,18,BB,FE,27,CA,20,A5,FE,52,CA,2D,A5,C3,10,A5,&08B8
2002 DATA 3E,05,32,60,A5,3E,C0,32,62,A5,C3,37,A5,3E,04,32,&05C4
2003 DATA 60,A5,3E,C5,32,62,A5,2A,81,AE,23,22,AB,A5,2A,83,&06DC
2004 DATA AE,22,A9,A5,2A,AB,A5,DD,2A,AB,A5,DD,46,01,DD,4E,&083E
2005 DATA 00,ED,43,AD,A5,ED,4A,22,AF,A5,ED,5B,AB,A5,DD,7E,&0922
2006 DATA 00,FE,FF,CA,75,A5,2A,AF,A5,22,AB,A5,3A,AD,A5,FE,&095B
2007 DATA 00,C8,C3,3E,A5,2A,83,AE,ED,5B,AD,A5,ED,52,22,A9,&086D
2008 DATA A5,2A,83,AE,ED,5B,AF,A5,ED,52,22,B1,A5,2A,AF,A5,&08D1
2009 DATA ED,5B,AB,A5,ED,4B,B1,A5,ED,B0,22,83,AE,22,85,AE,&096B
2010 DATA 22,87,AE,22,89,AE,C3,6C,A5,00,00,00,00,00,00,00,&0484
2011 DATA 00,00,00,4B,49,4C,4C,52,45,4D,3A,20,57,61,73,20,&03B5
2012 DATA 73,6F,6C,6C,20,65,6E,74,66,65,72,6E,74,20,77,65,&063C
2013 DATA 72,64,65,6E,20,27,20,6F,64,65,72,20,52,45,4D,20,&04DE
2014 DATA 3F,00,&003F

```

Der Hexlader stellt selbständig fest, welcher CPC-Typ vorliegt und führt die entsprechenden Anpassungen durch, wenn das Programm auf einem anderen Typ als dem CPC 464 läuft. Und hier auch gleich das Assemblerlisting für diejenigen, die es genauer wissen wollen, wie dieses Programm arbeitet oder auch eigene Anpassungen durchführen wollen. Anschließend werden wir etwas detaillierter auf das Programm selbst eingehen und Ihnen zusätzliche Informationen über das Programm geben.

## 5.2.2 Assemblerlisting: KILLREM.EDI

```

;*****
;*          KILLREM.EDI          *
;*-----*
;* entfernt die Basiczeilen, in denen *
;* denen das Zeichen ' oder REM *
;* unmittelbar hinter der Zeilen- *
;* nummer vorhanden ist. *
;* Das Zeichen kann beim Aufruf der *
;* Routine ausgewaehlt werden *
;*****

org &A500

basstart equ &ae81;pende equ &AE83;output equ &bb5a;waitkey equ &bb18

ld hl,ausgtext          ;Zeiger auf Fragetext

frage
ld a,(hl)                ;Akku mit Zeichen laden
cp a,0                   ;auf Textende pruefen
jp z,antwort             ;Textende ab zur Arbeit
call output              ;noch nicht Textende, Zeichen ausgeben
inc hl                   ;naechstes Zeichen
jp frage                  ;weitermachen

antwort                  ;auf Eingabe
call waitkey             ;warten
cp "'":jp z,hoch          ;Eingabe '
cp "R":jp z,rem           ;Eingabe R
jp antwort                ;keines der beiden Zeichen eingegeben

hoch
ld a,5                   ;Offset auf 5, da ' mit 01 c0
ld (ixoffset+2),a        ;codiert
ld a,&c0                  ;Code fuer '
ld (zeichen+1),a         ;beim CP-Befehl
jp kill

rem
ld a,4
ld (ixoffset+2),a
ld a,&c5
ld (zeichen+1),a

kill
ld hl,(basstart)         ;Basicstart - 1 holen
inc hl                   ;Basicstart berechnen
ld (aktzeile),hl         ;und als aktuellen Zeilenstart merken

start ld hl,(pende)      ;Programmende holen
ld (endmerk),hl          ;und in Merker ablegen
ld hl,(aktzeile)         ;aktuellen Zeilenstart ins
ld ix,(aktzeile)         ;IX-Register
ld b,(ix+1)

```

```

ld c,(ix)          ;BC auf aktuelle Zeilenlaenge 'stellen'
ld (akzeilen),bc   ;und merken

adc hl,bc          ;naechsten Zeilenstart berechnen
ld (neuzeile),hl   ;und merken
ld de,(aktzeile)   ;aktuelle Zeile

ixoffset
ld a,(ix+&00)      ;Zeichen der akt. Zeile holen (00=Dummywert)

zeichen
cp a,&FF           ;mit Zeichen vergleichen (FF = Dummywert)
jp z,entferne      ;wenn Zeichen - dann Zeile entfernen

weiter ld hl,(neuzeile) ;weitermachen
ld (aktzeile),hl   ;naechste Zeile ist aktuell

endpruef          ;Pruefung ob an Programmende angelangt
ld a,(akzeilen)   ;Laenge der aktuellen Zeile
cp a,0            ;Laenge 0
ret z             ;ja-> Ende
jp start          ;weitermachen mit naechster Zeile

entferne          ;Neues Ende berechnen
ld hl,(pende)     ;Programmende in HL
ld de,(akzeilen)  ;Laenge der aktuellen Zeile in DE
sbc hl,de         ;subtrahieren - Ergebnis = neues Ende
ld (endmerk),hl   ;und merken
ld hl,(pende)     ;Programm-Ende eine Zeile
ld de,(neuzeile)  ;hoeher, da aktuelle Zeile entfernt wird
sbc hl,de         ;berechnen wieviele Bytes zu
ld (restbyte),hl  ;verschoben sind
ld hl,(neuzeile)  ;Zum verschieben mit LDIR muss in HL die Quelle
ld de,(aktzeile)  ;in DE das Ziel und in BC die Anzahl
ld bc,(restbyte)  ;stehen
ldir              ;restliches Programm verschieben
ld (pende),hl     ;Zeiger Programmende      korrigieren
ld (pende+2),hl   ;Zeiger auf Variablenstart  "
ld (pende+4),hl   ;Zeiger auf Start der Felder  "
ld (pende+6),hl   ;Zeiger auf Ende der Felder   "
jp endpruef       ;Sprung zur Pruefung auf Ende

endmerk defs 2:aktzeile defs 2:akzeilen defs 2
neuzeile defs 2:restbyte defs 2
ausgtext text "KILLREM: Was soll entfernt werden ' oder REM ?":defb &00
;-----

```

Ein ganz wichtiger Punkt, den es bei der Benutzung des Programmes zu beachten gibt, ist der, daß ein Programm noch nicht gelaufen sein darf. Denn der CPC bewirkt durch sein Compreter-Verhalten, daß Zeilennummern, die im Programm "angesprochen" werden, beim Programmlauf in Speicheradressen umgewandelt werden.

Deshalb also muß das Programm, aus dem die Bemerkungszeilen entfernt werden sollen, frisch eingeladen werden.

Die Bedienung des Programmes selbst ist nicht schwierig. Nach dem Aufruf des Maschinenprogrammes (CALL &A500) werden Sie gefragt was entfernt werden soll. Sie haben dann die Möglichkeit zwischen Zeilen mit REM und Zeilen mit dem Hochkomma (das Zeichen auf der Zifferntaste 7, das mit SHIFT 7 eingegeben wird) zu wählen. Entfernt werden nur Zeilen, in denen diese REM-Kennzeichnungen direkt nach der Zeilennummer stehen. Dadurch können Sie erreichen, daß Zeilen mit REMarks, die im Programm verbleiben sollen, nicht durch KILLREM gelöscht werden.

Auch Basiczeilen in denen das betreffende Zeichen nicht unmittelbar auf die Zeilennummer folgt, bleiben vom Programm KILLREM unberührt.

An diesem Programmbeispiel können Sie erkennen, welch leistungsfähige Programme mit dem wenigen - bisher erarbeiteten Wissen über die CPCs - bereits geschrieben werden können. Prinzipiell, wäre es auch möglich, eine Routine zur REM-Entfernung auch in Basic zu schreiben, aber der Zeitbedarf für einen Programmlauf wäre dann bei umfangreichen Programmen doch sehr lange!

### Erklärungen zum Assemblerlisting: KILLREM

Zu Beginn stehen wieder die üblichen Vorbemerkungen, Festlegungen und Definitionen.

Das eigentliche Programm beginnt mit dem Befehl:

```
LD HL,ausgtext.
```

Hierdurch wird das Doppelregister HL mit der Anfangsadresse des Textes: "KILLREM: Was soll entfernt werden ..." geladen. HL dient als Zeiger auf den jeweiligen Buchstaben des Textes.

Durch LD A,(HL) wird dieser Buchstabe in den Akkumulator gebracht und der Inhalt des Akkus dann durch CP A,0 mit "0" verglichen. Zu Beginn des Programmlaufes zeigt HL auf den ersten Buchstaben des Textes also auf "K" und im Akku steht der Code für "K", also keine "0".

Da der Vergleich erbracht hat, daß im Akku keine "0" steht, wird der Akkuinhalt durch CALL output ausgegeben. Nun wird HL durch den Befehl INC HL um eins erhöht und zeigt damit auf den zweiten Buchstaben. Daraufhin wird dann zum Label Frage zurückgesprungen und solange Zeichen geholt, verglichen, ausgegeben und HL wieder erhöht, bis im Akku die Null des Textendes steht.

Daraufhin herrscht beim Vergleich Übereinstimmung und es wird durch JP Z,antwort bei diesem Punkt weitergemacht.

Durch CALL waitkey wartet der CPC auf eine Tasteneingabe. Wird die Taste "R" gedrückt, dann wird zur Routine "rem" gesprungen. Wird das Hochkomma gewählt dann zu "hoch". Wird eine andere Taste gedrückt, erfolgt eine Verzweigung zurück zum Label Antwort und es wird wieder gewartet.

Wurde Hochkomma gewählt, wird nun der Akku mit 5 geladen, bei REM mit 4. Der Grund ist, daß das Hochkomma mit der Folge 01 C0 und REM mit C5 codiert wird (Token).

Eine Basiczeile enthält in den ersten beiden Bytes die Länge der Zeile und in den nächsten beiden die Zeilennummer. Im fünften Byte ist das erste Zeichen der Basiczeile zu finden. Bei REM steht dort also C5, bei ' aber 01 C0. Als Indikator ob eine Zeile entfernt werden soll, werden die Codes C5 und C0 benutzt. Deshalb werden also die unterschiedlichen Werte bei den Labels "hoch" oder "rem" in den Akku geladen. Dieser Wert wird dann an die Speicherstelle ixoffset+2 geschrieben. Danach wird der Akku mit dem ausgewählten Code geladen und nach zeichen+1 gebracht. Dadurch steht dann bei "zeichen" entweder CP A,&C0 oder CP A,&C5!

Die beiden Routinen "hoch" und "rem" sind also nur dazu da, um nach der Auswahl die erforderlichen Werte in das Hauptprogramm zu schreiben. Ist dieses geschehen, beginnt bei "kill" erst das eigentliche Entfernungs-Programm.

HL wird mit dem Basicprogrammstart (- 1!) geladen und wird deshalb durch INC HL korrigiert. Es wäre zwar auch durch einen direkten Ladebefehl LD HL,&0170 das gleiche Ergebnis erzielt worden, aber die Zeiger auf den Basicstart können ja auch verstellt werden und so sind wir sicher, daß der aktuelle Basicstart geholt wird.

Dieser Wert wird dann in den Merker "aktzeile" gebracht, da die erste Basiczeile beim Start die aktuelle ist und folglich der Beginn der ersten Zeile mit dem Basicstart zusammenfällt.

Beim Label "start" geht es dann weiter. Zuerst wird das Ende des Basicprogrammes geholt (pende) und ebenfalls in einen Merker übertragen.

Die Befehle LD HL,(aktzeile) und LD IX,(aktzeile) holen wieder die aktuelle Speicheradresse in die beiden Doppelregister.

Durch LD B,(IX+1) und LD C,(IX) steht in BC die Zeilenlänge, da die ersten beiden Bytes ja diese enthalten. Auch die Länge wird für den späteren Bedarf zwischengespeichert (akzeilen = aktuelle Zeilen LEN; LEN steht für Länge).

Durch ADC HL,BC wird die Startadresse der nächsten Basiczeile berechnet und nach "neuzeile" gebracht. Danach wird DE mit der Adresse der ersten Speicherstelle geladen.

Durch LD A,(IX+...) wird das erste Zeichen in der Basiczeile in den Akku geholt und dann durch CP A,... verglichen. Ist es kein REM oder Hochkomma, dann wird HL mit der ersten Speicherstelle der nächsten Basiczeile (neuzeile) geladen und zur aktuellen Zeile gemacht. Dann wird durch LD A,(akzeilen) und CP A,0 überprüft, ob das Programmende schon erreicht wurde. Falls ja erfolgt durch RET Z der Rücksprung zu Basic. Falls nein, geht alles beim Label "start" wieder weiter.

Wie sieht es aber aus, wenn das Programm "fündig" geworden ist? Dann führte der Vergleich bei "zeichen" zur weiteren Programmabarbeitung bei "entferne" und da wird es ein klein wenig kompliziert.

Zuerst wird in HL die Speicherstelle des Programmendes benötigt.

In DE wird die Länge der Zeile gespeichert. Wenn dann die Länge von HL subtrahiert wird, dann erhält man die oberste Programmspeicherstelle, die nach der Entfernung der Zeile das neue Programmende ist. Dieser Wert kommt nach "endemerke".

Durch

```
LD HL,(pende)
LD DE,(neuzeile)    und
SBC HL,DE
```

erhält man die Anzahl der insgesamt mit dem LDIR-Befehl zu verschiebenden Bytes. Dieser Wert kommt nach "restbyte".

Nun beginnt der Verschiebevorgang. Dazu muß, um den LDIR-Befehl verwenden zu können in HL die Quelle, in DE das Ziel und in BC die Anzahl stehen und dann wird mit LDIR verschoben. Die Basiczeile mit dem REM oder ' wird dadurch überschrieben.

Nun müssen noch die Variablenzeiger korrigiert werden. Dies geschieht durch die Befehle LD (pende),HL und die nachfolgenden Befehle. Dann wird geprüft, ob das Basicprogrammende schon erreicht ist und je nach Ausgang dieser Prüfung geht es dann wie bekannt weiter.

### 5.3 Der Tastaturbuffer der CPCs

Wir hatten Ihnen nun schon aufgezeigt, wo verschiedene Buffer des CPC liegen, die mit Eingaben zu tun haben. Es gibt aber einen weiteren, der für "Automatik"-Programme von großem Interesse ist. In Verbindung mit der Definitionsmöglichkeit der Funktionstasten können damit Programme geschrieben werden, die sich selbst verändern.

Diese Programmtechnik kann beispielsweise dazu benutzt werden, um aus einem im Speicher befindlichen Programm einen Hexlader zu machen oder auch ein Kassetten-Inhaltsverzeichnis automatisch erstellen zu lassen.

#### Warum einen Hexlader?

Wie Sie wissen, können Binärfiles im Gegensatz zu Basic-Listings nicht ausgedruckt werden. Deshalb bedient man sich meist eines Hilfsmittels, wenn man derartige Programme weitergeben oder veröffentlichten will.

Die einfachste Methode ist, einen Hexdump des gewünschten Speicherbereiches auszudrucken. Voraussetzung beim "Empfänger" ist aber, daß dieser mit einem Maschinensprache-Monitor oder einem anderen Hilfsprogramm diese "Listings" eingeben kann.

Noch einfacher ist aber die Erzeugung eines "Hexladers", der beim späteren Programmlauf auch noch eine einfache Prüfung auf die Richtigkeit der Datastatements enthält. Solche Hexlader kennen Sie bereits, denn verschiedene Maschinenprogramme haben Sie vermutlich ja schon in Ihren CPC "eingegeben".

Zur Listingweitergabe von Binärfiles erzeugt man also etwas Ähnliches wie einen Hexdump, also einen hexadezimalen Speicherausgang des Speicherbereiches, in dem das Binärfile abgelegt ist. Dies kann beispielsweise gleich direkt auf Kassette oder Diskette erfolgen.

Nun sollen Sie aber eine etwas ungewöhnliche Methode kennenlernen, die Sie auch bei anderen Programmen sinnvoll einsetzen können:

#### 5.3.1 Wir manipulieren den Tastaturpuffer!

Bevor wir dies aber tun können, müssen wir erst einmal wissen, wo sich dieser eigentlich befindet. Um Ihnen mühsame Arbeit zu ersparen liefern wir Ihnen gleich die Lösung.

Der Tastaturpuffer des CPC 464 liegt ab &B514 und beim 664/6128 ab &B65E. Ab dieser "Grundadresse" werden also die eingegebenen Tastendrücke abgespeichert. Da maximal 255 Tastendrücke gespeichert werden und jeder Tastendruck (wie Sie sich durch das Programm TCODE überzeugen können) die Belegung von 2 Bytes bewirkt, muß dieser Buffer 510 Zeichen groß sein.



Dadurch ergeben sich die folgende Endadressen der Buffer:

CPC 464 = &B53B und CPC 664/6128 = &B685.

Zwischen diesen Anfangs- und Endwerten liegen also die "Tastenkoordinaten" der gedruckten Tasten. Damit der CPC aber weiß, welche Tastendrücke von ihm schon abgearbeitet wurden und welche noch zur Bearbeitung anstehen, braucht er weitere Informationen.

Auch diese sind in Speicherstellen, die sich in unmittelbarer Nähe des schon genannten Bufferbereiches befinden, untergebracht. Doch davon später, zuerst wollen wir einmal herausfinden, wie denn die Tastendrücke überhaupt codiert werden, und wie man diese zur Benutzung in einem späteren Programm erhält. Dazu ist es erforderlich zu wissen, wo denn diese stehen bzw. wo sich der CPC diese Eingaben merkt. Um aber nun nicht andauernd die Angaben für den 464 und 664/6128 machen müssen, legen wir nun einfach nur die Grundadressen fest und der Rest wird immer relativ angegeben.

Die Grundadresse des CPC 464 ist &B50A und bei den beiden anderen Typen ist es &B654! In GA (Grundadresse) und GA+1 findet der CPC den Tastencode der letzten gedruckten Taste. Deshalb können wir nun schon ein kleines Programm einsetzen, welches uns bei der "Suche" nach den Codes und auch dem Erkennen dieser hilft.

### 5.3.2 Listing: TCODE

```

100 *****
110 * TCODE - Programm zur Anzeige des Tastencodes und der ASC-Werte *
120 *****
130 '
140 MODE 1:PRINT" Tastencodes und ASC-Werte der zuletzt":PRINT
150 PRINT" gedruckten Taste bei den CPCs
160 '
170 'Feststellen ob CPC 464 oder anderer CPC-Typ
180 ga=&B50A:IF PEEK (&AC01)<> &C9 THEN ga=&B654
190 '
200 LOCATE 2,8:PRINT STRING$(38,"*"):LOCATE 2,12:PRINT STRING$(38,"*")
210 LOCATE 6,25:PRINT"Programm-Abbruch mit 2 * ESC
220 '
230 LOCATE 8,19:PRINT"Bitte eine Taste druecken"
240 '
250 'Warten bis Zeichen von der Tastatur kommt
260 x$=INKEY$:IF x$="" THEN 260
270 '
280 'Alte Werte loeschen
290 LOCATE 1,10:PRINT STRING$(39," ")
300 '
310 'Werte holen und in Hex umrechnen
320 as=ASC(x$):a=PEEK(ga):a$=HEX$(a,2):b=PEEK(ga+1):b$=HEX$(b,2)
330 '
340 'Ausgabe
350 LOCATE 6,10:PRINT"Code = ";:PRINT a$ "b$";:PRINT" ASC-Wert= ";as
360 '
370 GOTO 230:'weitermachen

```

## 5.3.3 Programmbeschreibung TCODE

Dieses Programm ist auf allen CPCs lauffähig! Wir gehen nur auf die wichtigsten Programmzeilen ein, denn der Rest dürfte von jedermann verstanden werden.

Zeile 180: Hier geschieht die Festlegung der Grundadresse (GA) für den CPC 464. Anschließend wird geprüft, ob ein anderer CPC-Typ vorhanden ist und wenn ja, dann wird die Grundadresse geändert.

Zeile 290: Die alten Ausgaben werden gelöscht.

Zeile 320: Die erforderlichen Werte werden aus dem Speicher ausgelesen und umgewandelt.

Zeile 350: Ausgabe der Werte

Für viele wird es sehr schwierig sein festzustellen, wie diese Werte innerhalb des CPC zustande kommen, deshalb hier die erforderlichen Hintergrundinformationen.

Die Tastatur des CPC ist hardwaremäßig in Spalten und Reihen aufgeteilt. Es ist also eine Matrixanordnung vorhanden und genau diese Matrixwerte werden im Ringbuffer abgelegt. Damit Sie sich dies noch detaillierter ansehen können erfolgt nun eine kleine (aus drucktechnischen Gründen nicht ganz vollständige) Übersicht dieser Matrix.

	!	01	02	04	08	10	20	40	80
09	!								DEL
08	!	1	2	ESC	Q	TAB	A	CL	Z
07	!	4	3	E	W	S	D	C	X
06	!	6	5	R	T	G	F	B	V
05	!	8	7	U	Y	H	J	N	SPACE
04	!	0	9	O	I	L	K	M	,
03	!	^	-		P	;	:	/	.
02	!	CLR		ENT		4	SH		CTRL
01	!	CL	CY	7	8	5	1	2	0
00	!	CH	CR	CD	9	6	3	ENT	.

Erklärungen: So ist die hardwaremäßige Dekodierung der Tastatur. Wenn Sie die Taste E drücken, erhalten Sie als Ausgabewert "07 04". Bei W "07 08", bei S "07 10", bei F "06 20" usw.

Wer will, kann nun in seinem CPC-Handbuch einmal die Darstellung der Nummernzuordnung zu Tasten und Joystick-Signalen näher ansehen, denn mit obiger Aufstellung wird klar, daß die Zuordnung nicht wahllos geschah, sondern wegen der "verdrahteten" Anordnung sich die dortige "Unübersichtlichkeit" ergibt.

Wenn Sie aus obiger Darstellung einmal die Zeile mit der Zeichenfolge 6,5,R,T,G... mit der Zuordnung der Nummern im Handbuch

vergleichen, dann sehen Sie, daß die Nummern exakt der Matrixanordnung entsprechen.

Versuchen Sie nun doch einmal selbst etwas mehr über die Arbeits- und Wirkungsweise dieser Ablage und deren Abarbeitung herauszufinden. Der Ihnen schon bekannte Monitor kann dabei hervorragende Dienste leisten.

Mit dem Programm MONI können Sie dies alles sehr gut verfolgen. Geben Sie beispielsweise bei einer Adresseneingabe (vor Betätigung der Enter-Taste) lauter Doppelpunkte ein, variieren Sie mit der Anzahl der Zeichen usw. vielleicht finden Sie - ebenso wie wir damals - die Lösung.

Als wir diese Analysen machten, gab es noch keine internen Informationen über die CPCs, geschweige denn ein Firmware-Handbuch oder gar ein dokumentiertes ROM-Listing. Nur durch Untersuchungen des Verhaltens der einzelnen Speicherplätze bei Eingaben fanden wir heraus, wie dies alles abläuft. Wenn wir damals dadurch auch nicht alles herausfanden, es reichte um in der "CPC-Anfangszeit", den CPC 464 sehr komfortabel und nutzbringend zu überlisten, wie Sie beim Programm DATAGEN sehen und erleben können.

Nun aber wieder zu den weiteren Erklärungen.

Die Ablage der eingegebenen Tasten läuft in diesem Buffer im Kreis, es handelt sich nämlich um einen sogenannten Ringbuffer, in dem die Zeichen der Reihe nach abgelegt werden. Dabei laufen bei den Eingaben und auch bei der Abarbeitung Zeiger mit, die auf die jeweiligen Zeichen zeigen, bzw. die Zeichen zählen usw.

Diese Speicherstellen liegen unmittelbar hinter dem Ringbuffer, also ab der Speicherstelle GA+40.

Um dem CPC also ein automatisches Arbeiten beizubringen, muß auch noch bekannt sein, wie diese Speicherstellen manipuliert werden müssen, das heißt, dass die Bedeutung dieser Speicherstellen geklärt werden muß. Auch dies haben wir bereits für Sie getan.

Die Bedeutung dieser Speicherstellen:

GA+40 = Anzahl der freien Ringbuffer-Einträge + 1  
GA+41 = Schreibzeiger  
GA+42 = Anzahl der Einträge im Buffer  
GA+43 = Lesezeiger  
GA+44 = Anzahl der Einträge

So, und nun steht dem Programm eines Datagenerators mit manipuliertem Tastaturpuffer nichts mehr im Wege.

#### 5.3.4 Listing: Datagenerator für alle CPCs

```
100 REM DGEN.BAS - Datagenerator mit Tastaturmanipulation
110 '
120 MODE 2: ' 80 Zeichen-Darstellung
130 zn=2000: 'Zeilennummer fuer Data-Start
140 znh=INT(zn/256):zn=zn-znh*256: ' High- und Low-Byte berechnen
```

```

150 POKE &160,znh:POKE &161,znl:' Zeilennummer merken
160 CALL &BB00:' Tastaturverwaltung initialisieren
170 ident=PEEK(&AC01):' Feststellen welcher CPC
180 '
190 'Annahme 464, falls nicht dann Identmerker aendern
200 POKE &16F,0:IF ident <> &C9 THEN POKE &16F,1
210 '
220 PRINT"Datagenerator fuer die CPC's von Schneider":PRINT
230 '
240 INPUT"Anfang      : & ";anfang$:INPUT"Ende      : & ";ende$
250 INPUT"Programmname : ";name$
260 '
270 'Merker berechnen und poken
280 anfang=VAL("&"+anfang$):ende =VAL("&"+ende$)
290 IF anfang <0 THEN anfang = anfang + 2^16
300 IF ende <0 THEN ende =ende +2^16
310 anfanghigh=INT(anfang/256):anfanglow=anfang-anfanghigh*256
320 endehigh=INT(ende/256):endelow=ende-endehigh*256
330 'Anfangs- und Endadressen merken
340 POKE &162,anfanghigh:POKE &163,anfanglow
350 POKE &164,enderhigh:POKE &165,endelow
360 '
370 'Zeilen mit Filenamen, sowie Anfang und Ende schreiben
380 KEY 0,"870 rem "+name$
390 KEY 1,"880 a="+STR$(anfang)+"e="+STR$(ende+1)+"zb=2000
400 KEY 2,"goto 420"+CHR$(13):GOTO 710
410 '
420 RESTORE:CLS:CLEAR
430 zn=PEEK(&160)*256+PEEK(&161):'Zeilennummer holen
440 zn$=STR$(zn):' Aus Zeilennummer String bilden
450 zn=zn+1:' Zeilennummer fuer naechsten Durchlauf erhoehen
460 'Neue Zeilennummer merken
470 POKE &160,INT(zn/256):POKE &161,(zn-INT(zn/256)*256)
480 'Speicheradresse holen
490 adresse =PEEK(&162)*256+PEEK(&163)
500 IF adresse < 0 THEN adresse = adresse + 2^16
510 '
520 'Endadresse holen
530 ende=PEEK(&164)*256+PEEK(&165)
540 IF ende <0 THEN ende = ende +2^16
550 'String mit Hexwerten bilden
560 h$=h$+HEX$(PEEK(adresse),2)+","
570 pruef=pruef+PEEK(adresse)
580 '
590 adresse =adresse+1:zaehler = zaehler+1
600 POKE &162,(INT(adresse/256)):POKE &163,(adresse-INT(adresse/256)*256)
610 IF zaehler = 16 THEN IF adresse >ende THEN 830 ELSE GOTO 660
620 IF adresse > ende THEN 830
630 GOTO 490
640 :
650 REM Keyfunktionen als Puffer benuetzen
660 g$=zn$+" data "+h$+"&"+HEX$(pruef,4)
670 KEY 0,g$:REM Zeilennummer+Data+Datas+Pruefsumme
680 KEY 1,"goto 420"+CHR$(13)
690 :
700 REM Tastaturpuffer manipulieren - Keyfolge 0,1,2
710 IF PEEK(&16F) = 0 THEN b=&B514:c=&B53C:REM CPC 464

```

```

720 IF PEEK(&16F) = 1 THEN b=&B65E:c=&B686:REM CPC 664 oder 6128
730 '
740 'Die Tastendruecke simulieren
750 FOR i = 0 TO 11:READ a$:a$="&"&a$:POKE b+i,VAL(a$):NEXT i
760 DATA 02,04,01,80,02,04,01,20,02,04,01,40
770 '
780 'Anzahl der Tasten manipulieren
790 FOR i = 0 TO 3:READ a$:a$="&"&a$:POKE c+i,VAL(a$):NEXT i
800 DATA 10,05,06,00
810 END
820 '
830 g$=zn$+" data "+h$+"&"&HEX$(pruef,4)
840 KEY 0,"":KEY 1,""
850 KEY 2,g$+CHR$(13)+"delete -860"+CHR$(13)+"call &bb00"+CHR$(13)
860 END
870 REM Ende Generatorteil
880 '
890 FOR i =a TO e:READ d$:IF LEFT$(d$,1)<>"&" THEN 920
900 IF ps<>VAL(d$) THEN PRINT"Fehler Zeile "zb:END
910 ps=0:d$="":IF i=e THEN END:ELSE i=i-1:zb=zb+1:GOTO 950
920 d$="&"&d$
930 POKE i,VAL(d$):ps=ps+VAL(d$)
940 PRINT VAL(d$)
950 IF i < e THEN NEXT i

```

### 5.3.5 Programmbeschreibung DGEN.BAS

Die Zeilen bis einschließlich 120 dürften keine Fragen aufwerfen. In Zeile 130 wird in der Variablen ZN der Wert für die erste Datazeilennummer festgelegt. Wer also will, daß seine Datastatements bei 5000 beginnen, kann dies dort ändern.

In Zeile 140 wird diese Zeilennummer in High- und Lowbyte zerlegt und in Zeile 150 unterhalb (!) des Basic-Programmes abgelegt. Wie wir Ihnen bereits versprochen hatten können Sie nun sehen, daß auch Speicherstellen unterhalb eines Basic-Programmes genutzt werden können.

Üblicherweise reserviert man immer (durch Begrenzung mit MEMORY) Speicherplatz oberhalb der Stringvariablen, hier zeigen wir mit diesem Programmbeispiel, daß es auch unterhalb des Basicbeginnes möglich ist ein paar Bytes zu benutzen.

In Zeile 160 wird die Tastaturverwaltung initialisiert, d.h., daß alle Manipulationen des Tastaturpuffers und auch der Key-Definitionen "zurückgestellt" werden. Warum dies sein muß erfahren Sie etwas später.

Durch Zeile 170 wird festgestellt, welcher CPC-Typ gerade mit dem Programm "arbeiten" darf, um dann in der Zeile 200 den korrekten Wert für den Typ abzulegen.

In Zeile 240 erfolgen die Angaben, welcher Speicherbereich in Datastatements umgewandelt werden soll. Die Eingaben sollen in hexadezimaler Form geschehen, aber ohne Eingabe des &-Zeichens.

In Zeile 250 erfolgt die Eingabe des Programmnamens.

Durch die Zeilen 270 bis 350 werden diese bei jedem Programmstart benötigten Werte umgerechnet und in "Merkspeicherstellen" (auch unterhalb des Basicprogrammes) gePOKEt.

Und nun kommen in den nächsten Zeilen weitere kleine "Vorbereitungs"-tricks des Programmes.

Der Funktionsstaste "0" (Ziffernblock), wird der String "870 REM Programmname" zugewiesen. (Dies bedeutet, daß dann, wenn diese Taste gedrückt wird, dieser String auf dem Bildschirm ausgegeben wird.)

Die Funktionstaste "1" wird mit der Zeilen-Nummer 880, der Zuordnung a=, dann mit dem eingegebenen Wert für den Anfang, dann einem Doppelpunkt usw. belegt. Dies bedeutet, daß dann, wenn diese Taste gedrückt wird die Zeile 880 mit den entsprechenden Werten ausgegeben wird.

Ähnliches geschieht in Zeile 400.

Würde das Programm nicht weitergehen, sondern hier enden, dann könnten Sie nun folgendes manuell tun:

Die Funktionstaste 0 drücken. Dadurch würde die Zeile 870 mit der Bemerkung des Programmnamens auf dem Bildschirm zu sehen sein. Würden Sie nun die ENTER-Taste betätigen, dann würde diese Basiczeile ins Programm übernommen werden.

Als nächstes würden Sie die Funktionstaste 1 drücken. Dadurch und durch anschließendes Drücken der ENTER-Taste würden Sie die Basiczeile 880 mit dem Anfangs- und Endwert des gewählten Speicherbereiches, sowie der ersten Zeilennummer für die Datastatements erzeugen und in's Programm nehmen.

Wenn Sie nun noch die Funktionstaste 2 betätigen würden, dann würde das Programm durch GOTO 420 wieder starten.

Was wir damit nun aufzeigen wollten, ist die Funktionsweise des Programmes. In Wirklichkeit läuft der Datagenerator also nicht wie ein anderes Programm ab, sondern läuft bis zu einer bestimmten Zeile und bricht dann eigentlich ab.

Da aber durch die Manipulation des Tastaturpuffers genau das geschieht, was wir eben beschrieben haben, läuft das Programm immer an und weiter, bis die Endbedingung erreicht ist.

Nun zu den weiteren Zeilen. Beim ersten Mal läuft das Programm selbst über die Zeilen 400, dann 710 weiter.

In Zeile 710 werden nun die Speicherstellen, deren Bedeutung wir schon erklärt haben für den jeweiligen CPC-Typ festgelegt.

Die Zeilen 750 bis 800 sind dann für die Manipulation des Tastaturpuffers und seiner zugehörigen Speicherstellen zuständig.

Wenn Sie sich die Werte in Zeile 760 einmal mittels der Tabelle decodieren, erhalten Sie folgende "Eingaben":

ENTER-Taste	(02 04)
Funktionstaste 0	(01 80)
ENTER-Taste	(02 04)
Funktionstaste 1	(01 20)
ENTER-Taste	(02 04)
Funktionstaste 2	(01 40)

Durch die Zeilen 790 und 800 werden nun noch die Zeiger des Ringbuffers "gestellt". Was dadurch geschieht, sehen Sie beim späteren Programmlauf dann selbst. Diese Manipulationen bewirken, daß der CPC die Funktionstasten für Sie "drückt", das ist alles.

Wird das Ende des auszugebenden Bereiches erreicht, so wird dies in Zeile 810 oder 820 erkannt und dann zu Zeile 830 gesprungen. Dort erfolgt die Aufbereitung der letzten Statementzeile.

Dann werden die Key-Definitionen der Funktionstasten 0 und 1 gelöscht und die Funktionstaste 2 neu "belegt".

Der erste Teil der Neubelegung ist die letzte zu erzeugende Zeile mit "angehängter" ENTER-Taste =chr\$(13). Dann folgt der Befehl DELETE -860 (ebenfalls mit CR =Carriage Return, also der ENTER-Taste) und dann noch die Rücksetzung der Tastaturverwaltung, denn sonst würden Sie bei Betätigung der Tasten 0 und 1 des Ziffernblockes keine Eingaben erhalten und durch die Taste 2 würde bei Betätigung immer die letzte erzeugte Zeile und der Befehl DELETE - 860 ausgegeben werden.

Die nicht beschriebenen Zeilen dürften Ihnen aufgrund Ihres Wissens und auch der Beschreibungen der anderen Basiczeilen dieses Programmes nun keine Verständnisschwierigkeiten mehr bereiten.

Am Ende der vielen Neustarts und Programmläufe, die der CPC für Sie durchführt, verbleibt im CPC noch der reine Hexlader übrig, den Sie dann abspeichern oder ausdrucken können.

(Einige Hexlader-Listings dieses Buches wurden mit diesem Programm erzeugt.)

Wie Sie sich im Programmlisting überzeugen können, ist beim Datagenerator keinerlei Maschinensprache eingesetzt, sondern nur das Wissen über den CPC selbst, hat es ermöglicht, dieses Programm zu schreiben.

Daß mit dieser Technik auch viele andere trickreiche Programme geschrieben werden können, zeigt das nun folgenden Programmbeispiel, das Ordnung in die Kassettensammlung bringen soll.

#### 5.4 Kassetten-Inhaltsverzeichnis für die Schneider CPC's

Wenn die Anzahl der Kassettenprogramme und damit auch die Zahl der Programmkassetten für Ihren CPC immer mehr werden, geht sehr schnell der Überblick über eine Programmsammlung verloren. Wer

schreibt denn wirklich immer gleich die neuen Programmnamen auf die Label seiner Kassetten? Wer macht sich außerdem noch die Mühe aufzuschreiben, um welchen Filetyp es sich handelt, wo der Programmstart liegt usw.? Deshalb ist es gut, wenn man ein Programm hat, das in der Lage ist, einen Großteil dieser Arbeiten automatisch zu erledigen. Hier ist es.

Bei diesem Programm werden einige kleine Programmiertricks angewandt, die es aber in sich haben, wie Sie noch feststellen werden. Doch immer der Reihe nach. Sie brauchen dieses Programm nur zu laden, zu starten und dann der Anweisung zu folgen. (Das Programm erkennt selbst, ob es sich um einen CPC 464, 664 oder 6128 handelt.)

Alles was Sie während des Programmlaufes tun müssen, ist: eine Kassette nach der anderen einlegen. Durch das Programm erhalten sie eine komplette Aufstellung aller Ihrer Datenfiles.

Diese Katalogisierung können Sie so ganz nebenbei erledigen lassen. Wenn eine Kassette abgelaufen ist, dann wenden Sie diese oder legen die nächste ein. Sind alle Kassetten "abgearbeitet", dann bitte zweimal die Escape-Taste drücken.

Alle erfassten Programme mit den dazugehörigen Informationen stehen - als Datastatements - ab Zeilennummer 100000 im Speicher des CPC und können weiterverarbeitet werden. Durch Löschen der Programmzeilen des ursprünglichen Programmes (Delete -2000) und Ersatz durch eine Ausgaberroutine (Merge "Ausgabe") können Sie sich auch sofort Ausdrucke erstellen lassen. Dieses Ausgabe-Hilfsprogramm ist ebenfalls mit abgedruckt, wobei Sie bei diesem zwischen Drucker- und Bildschirmausgabe wählen können.

Falls Sie über einen Drucker, der Ausgabemöglichkeiten für komprimierte Schriften hat, verfügen, können Sie das Programm so umschreiben, daß nach Abtrennen bzw. Ausschneiden der jeweiligen Aufstellung diese gleich als Kassettenetikett benutzt werden kann.

### Ein sinnvoller Tip für Beginner

Wer mit seiner Kassetten-Programmsammlung erst beginnt, kann durch einen kleinen Dreh die Übersichtlichkeit noch wesentlich erhöhen. Er kann nämlich jeder Kassette einen Namen geben, indem er als erstes Programm ein "Blindfile" abspeichert. Also ohne ein Programm im Speicher zu haben, einfach mit Save "Kassettenname" abspeichern. Dieser Kassettenname wird dann auf das Etikett geschrieben und fertig. Aber bitte nicht auf schon "beschriebene" Kassetten diesen Namen speichern, denn sonst würde der erste Teil Ihres ersten Programmes ja verloren gehen!

Wird beim Programm-Namen eine Kennzeichnung, die bei keinem anderen Programm vorkommt, mit abgespeichert, kann eine erweiterte Ausgabe- oder Druckroutine dies berücksichtigen und durch entsprechende zusätzliche Ausgaben die Übersichtlichkeit noch weiter erhöhen.



#### 5.4.1 Programmbeschreibung: Kassetteninhaltsverzeichnis

Obwohl dem Listing selbst schon eine gewisse Anzahl von Informationen zu entnehmen sind, hier doch noch einige Hinweise.

Zeile 170: Besitzer eines CPC 464 ohne angeschlossene Diskettenstation müssen den ersten Befehl entfernen, da es sonst zu einer Fehlermeldung kommt, denn erst durch den korrekten Anschluß der Floppystation wird dieser RSX-Befehl eingebunden. Das Programm wird durch Tastaturpuffermanipulation laufend neu gestartet. Um die aktuelle Zeilennummer, die beim automatischen Programmlauf immer wieder gebraucht wird zu retten, wird diese oberhalb des Basicprogrammspeichers abgelegt. Der Befehl MEMORY begrenzt den von Basic aus erreichbaren Speicherplatz. Außerdem wird in dieser Zeile der Wert für die Zeilennummer bestimmt, die als erste Datazeile erzeugt werden soll.

Zeile 180: Die Zeilennummer der ersten Datazeile wird in High- und Low-Byte zerlegt.

Zeile 190: Die berechneten Werte werden in die Speicherstellen &A000 und &A001 gepoke't. Dann erfolgt die Umschaltung in Mode 2 (80-Zeichen-Darstellung).

Zeile 230: Diese Zeile dient zu einer sehr simplen Abfrage, um welchen Computertyp es sich handelt. Bei den Typen 664 und 6128 steht dort kein &C9.

Um bei den Autostartvorgängen immer die korrekten Werte für die PEEK-Adressen zu erhalten, wird in &A002 entweder die Null (464) oder eine Eins (664-6128) abgelegt.

Zeile 250 bis 300: Diese dienen zur Definition der Fenster und zur Festlegung der Ausgabetexte.

Zeile 320: Hier erfolgt die Ausgabe dieser Texte im Fenster 1.

Zeile 330: Warten bis die Return- bzw. die Enter-Taste gedrückt wird.

Zeile 340 bis 360: Ausgabe der "Bildschirm-Maske", also der Kopfleiste, der Trennstriche usw.

Zeile 380: Der Call-Befehl setzt die Tastatur zurück, damit die (beim Programmlauf undefinierten) Funktionstasten wieder die ursprünglichen Werte erhalten. Außerdem bewirkt der ON BREAK STOP-Befehl, daß durch zweimalige Betätigung der ESC-Taste das Programm abgebrochen wird.

Zeile 390: Dieser Befehl bewirkt, daß trotz des Fehlers, der durch Zeile 420 bewirkt wird, das Programm weiterarbeitet.

Zeile 400: Hier wird die Zeilennummer für die spätere Generierung geholt.

Zeile 410: Bewirkt, daß der noch freie Speicherplatz mitgeteilt wird.

Zeile 420: Ein kleiner Trick, um die File-Header in den Header-buffer zu bekommen. Der auftretende "Fehler" wird durch Zeile 390 abgefangen.

Zeile 440: Bestimmung der Grundadresse "ga" für den CPC-Typ.

Zeile 460 bis 470: Hier geschieht die Umwandlung der Speicherinhalte in die entsprechenden ASCII-Zeichen. Steuerzeichen (kleiner 32) werden durch Blanks ersetzt. Außerdem werden Zeichen mit Werten größer 128 unterdrückt und durch ein Fragezeichen ersetzt. Der Druckeranschluß hat nur sieben Bit!

Zeile 490 bis 550: Erkennen des Dateityps. Da die Möglichkeit besteht, auch fremde Dateitypen zu erzeugen, ist auch ein "unbekannter Filetyp" vorgesehen um ggf. darauf aufmerksam zu machen.

Zeile 570: Berechnung der Datenbytes im Record.

Zeile 590 bis 620: Feststellen woher der File stammt.

Zeile 650 bis 660: Bei mit Save"name",b,Adresse,Länge,Startadr. abgespeicherten Programmen, die Startadresse berechnen.

Zeile 680 bis 710: Berechnung der Programmlänge. Diese Information fehlt im Header von ASCII-Files!

Zeile 730 bis 770: Berechnung der Gesamtblockanzahl aus der Bytelänge.

Zeile 790: Ausgabestring für die Bildschirmausgabe zusammenfügen und ausgeben.

Zeile 820: Die Zeilennummer für die aktuelle Datazeile festlegen und für den weiteren Automatiklauf merken.

Zeile 870: kompletten Zeilenstring aufbauen.

Zeile 900: Zeilenstring auf die Funktionstaste 0 legen. Auf Funktionstaste 1 den Befehl "RUN 320" legen. Cursor positionieren.

Zeile 920 bis 960: Diese Zeilen manipulieren den Tastaturpuffer derart, daß dem CPC vorgegaukelt wird, Sie hätten die mit dem kompletten Zeilenstring belegte Funktionstaste 0, dann ENTER und dann die Funktionstaste 1 und nochmals ENTER gedrückt. Die Wirkung ist also genauso, als wenn sie dem Programm eine neue Datazeile zugefügt und dann run 320 eingegeben hätten.

Da die Tastaturbuffer bei den CPC's unterschiedlich liegen, muß in Zeile 920 deren Adresse festgelegt werden.

Wenn Sie nun alle Kassetten durchlaufen haben lassen, dann betätigen Sie bitte zweimal die ESC-Taste. Daraufhin bricht der Computer das Programm ab. Geben Sie nun bitte den Befehl MODE 2 ein. Nun können Sie das Programm mit den automatisch hinzugefügten Zeilen ohne Probleme listen. Um nun weiterzumachen, sollten Sie die Programmzeilen wie schon erwähnt löschen, und die Ausgaberoutine "hinzumischen" (mergen).

Da der CPC 464 in Verbindung mit der Diskettenstation den schon angesprochenen MERGE-Fehler hat, müssen die Besitzer dieser Gerätekombi-configuration das Ausgabeprogramm als ASCII-File abgespeichert haben, sonst klappt das "mergen" nicht.

Nun steht einem evtl. Ausdruck nichts mehr im Wege.

Damit Sie nun nicht jedesmal, wenn Sie eine neue Kassette in's Verzeichnis aufnehmen wollen, all die anderen, schon aufgenommenen, Kassetten wieder "durchlaufen" lassen müssen, gibt es verschiedene Möglichkeiten.

Entweder Sie speichern das Programm nach dem Programmlauf ab, laden es beim nächsten Mal, setzen die Zeilennummer (edzn) höher als die höchste bereits aufgenommene und lassen dann das Programm laufen, oder Sie lassen die neue Kassette alleine "laufen" und "mergen" dann nach Umnummerierung diese dem alten Inhaltsverzeichnis hinzu. Selbstverständlich können Sie auch "Handeinträge" vornehmen.

Irgendwann ist die "Aufnahmekapazität" des Programmes natürlich erschöpft, dann legen Sie einfach ein zweites Verzeichnis an. Das Programm können Sie getrost noch erweitern, bzw. Ihren Wünschen anpassen, aber beachten Sie immer, daß der RUN-Befehl der Funktionstaste 1 auf die richtige Zeilennummer zeigt.

#### 5.4.2 Listing: Kassetteninhalt

```

100 '*****
110 '* Programm zur Kassettenanalyse und Erstellung von *
120 '*      Datazeilen mit den gefundenen Werten fuer *
130 '*      CPC 464 bis 6128 *
140 '*      20.01.86 *
150 '*****
160 '
170 'TAPE:MEMORY &9FFF:edzn = 9999 '      edzn = Erste Datazeilen-Nummer-1
180 hi=INT(edzn/256):lo = edzn-hi*256'      Zeilennummer zerlegen
190 POKE &A000,lo:POKE &A001,hi:MODE 2:CLEAR' Zeilennummer merken
200 '
210 ' In Zeile 230 wird geprueft ob CPC 464 oder 664/6128
220 ' und Kennzeichen nach &a002 gepoked (0 = 464, 1 = 664/6128)
230 IF PEEK(&A001)=&C9 THEN POKE &A002,0 ELSE POKE &A002,1
240 '
250 ' Ausgabestrings definieren
260 t1$="Kassetten-Analyse und Erzeugung eines Data-Inhaltsverzeichnisses
270 t2$="Bitte die PLAY- und dann <Enter>-Taste druecken"
280 ' Festlegung der Fenster
290 WINDOW #0,1,80,23,24:WINDOW #1,1,80,1,2: WINDOW #2,1,80,3,19
300 WINDOW #3,1,80,5,21: WINDOW #4,1,80,22,22: WINDOW #5,67,80,2,2
310 '
320 PRINT #1,t1$:PRINT #1,t2$
330 x$=INKEY$:IF x$ <> CHR$(13) THEN 330
340 PRINT #2,"PROGRAMM-Name BA Bytes Quelladr. MPG-Start Filetyp"
350 PRINT #2,STRING$(80,"-"):PRINT #4,STRING$(80,"-");
360 CLS #1:PRINT #1,T1$
370 '-----

```

```

380 CALL &BB00:ON BREAK STOP'                call &bb00 =Tastatur ruecksetzen
390 ON ERROR GOTO 440
400 znalt=PEEK(&A000)+PEEK(&A001)*256'         alte Zeilennummer holen
410 frei=FRE(0):PRINT #5,"Frei: "frei'        Freimeldung ausgeben
420 OPENIN "!"'                               so tun als ob
430 '
440 IF PEEK(&A002)=0 THEN ga =&B88C ELSE ga= &B1A4' CPC 464 oder 664/6128 ?
450 'Dateiname maximal 16 Zeichen Laenge
460 FOR i = ga TO ga+15:a=PEEK(i):IF a<33 THEN a=32 ELSE IF a>128 THEN a=63
470 n$=n$+CHR$(a):NEXT i
480 '
490 'Dateityp; auch andere als im Handbuch enthaltene Typen moeglich
500 dt=PEEK(ga+18)
510 IF dt=0 THEN dt$="Basic-File":GOTO 570
520 IF dt=1 THEN dt$="Basic-geschuetzt":GOTO 570
530 IF dt=2 THEN dt$="Binaer-File":GOTO 570
540 IF dt=22 THEN dt$="ASCII-File":GOTO 570
550 dt$="unbekannter Filetyp
560 '
570 adbr=PEEK(ga+19)+PEEK(ga+20)*256'         Anzahl der Datenbytes im Record
580 '
590 'Datenquelle = woher stammt der File urspruenglich ?
600 IF PEEK(&A002)=0 THEN dq=PEEK(&B8A1)+PEEK(&B8A2)*256' CPC 464
610 IF PEEK(&A002)=1 THEN dq=PEEK(&B134)+PEEK(&B135)*256' CPC 664/6128
620 h$=HEX$(dq,4)'                           Herkunft
630 :
640 'mpgst = evtl. Startadresse bei Maschinenprogrammen
650 mpgst=PEEK(ga+26)+PEEK(ga+27)*256:mpgst$=" "&HEX$(mpgst,4)
660 IF mpgst=0 THEN mpgst$="-----"
670 '
680 'Anzahl der Bytes (fehlt bei ASCII-Files !)
690 by = PEEK(ga+24)+PEEK(ga+25)*256:IF by=0 THEN by$="????":GOTO 730
700 by$ = MID$(STR$(by),2)
710 IF LEN (by$)<5 THEN by$=CHR$(32)+by$:GOTO 710
720 :
730 'blockanzahl (muss berechnet werden!)
740 bl =INT(by/2048):IF bl *2048 <> by THEN bl =bl +1
750 bl$=MID$(STR$(bl),2)
760 IF LEN(bl$)<2 THEN bl$="0"+bl$:GOTO 760
770 IF dt=22 THEN bl$="?":'Ascii-File / Laenge und Bloecke nicht im Header
780 '
790 x$=n$+" "+bl$+" "+by$+" "&h$+" "+mpgst$+" "+dt$:PRINT#3,x$
800 '
810 ' neue Zeilennummer poken
820 zn=znalt+1:znh=INT(zn/256):zn1=zn-znh*256:POKE &A000,zn1:POKE &A001,znh
830 '
840 RESTORE:cr$=CHR$(13)
850 :
860 ' hier den String zusammenbasteln
870 dat$=STR$(zn)+" data "+n$+" "+bl$+" "+by$+" "+h$+" "+mpgst$+" "+dt$
880 :
890 REM Keyfunktionen als Puffer benuetzen und positionieren
900 KEY 0,dat$:KEY 1,"run 380"+cr$:LOCATE 1,24
910 :
920 IF PEEK(&A002)=1 THEN b=&B65E ELSE b=&B514
930 REM Tastaturpuffer manipulieren - Keyfolge 0,1,2
940 FOR i = 0 TO 11:READ a:a$=" "&a$:a=VAL (a$):POKE b+i,a:NEXT i

```

```

950 IF PEEK(&A002)=1 THEN b=&B686 ELSE b=&B53C
960 FOR i = 0 TO 3:READ a$:a$="&"+a$:POKE b+i,VAL(a$):NEXT i
970 '
980 DATA 02,04,01,80,02,04,01,20,02,04,01,40
990 DATA 10,05,06,00
1000 END
1010 '-----

```

#### 5.4.3 Druckprogramm zum Kassetten-Inhaltsverzeichnis

```

100 'Diese kleine Routine gibt das Kassetteninhaltsverzeichnis aus
110 MODE 2: PRINT "Kassetteninhaltsverzeichnis":PRINT:PRINT
120 PRINT "Ausgabe auf dem Drucker j/n"
130 x$=LOWER$(INKEY$):IF x$="" THEN 130
140 IF x$="j" THEN o=8:GOTO 170
150 IF x$="n" THEN o=0:GOTO 170
160 GOTO 130
170 PRINT #o,"Programm-      Block-   Laenge   Herkunft   Start      Filetyp
180 PRINT #o,"Name          Anzahl    (dez)     (hex)      bei MPG
190 PRINT #o,STRING$(78,"-")
200 READ name$:IF name$="ende" THEN END
210 READ blockanzahl$,Laenge$,herkunft$,startadr2$,typ$
220 PRINT #o,name$;
230 PRINT #o,TAB(20)blockanzahl$;
240 PRINT #o,TAB(27);USING "#####";VAL(Laenge$);
250 PRINT #o,TAB(38)herkunft$;
260 PRINT #o,TAB(47)startadr2$;
270 PRINT #o,TAB(57) typ$
280 GOTO 200
60000 DATA ende

```

Wie Sie sehen, bietet die Methode der Manipulation des Tastaturpuffers, in Verbindung mit der Stringzuweisung auf Funktionstasten, sehr komfortable Möglichkeiten, den CPC für sich arbeiten zu lassen.

Es muß also nicht immer Maschinensprache sein, um Möglichkeiten, die der CPC (mit reinem Basicwissen) nicht von Hause aus zuläßt, zu nutzen. Ein paar interessante Speicherstellen, die von ihm benutzt werden und schon kann man Programme schreiben, die es in sich haben.

Sehen Sie sich nun mit dem Monitor weiter im Speicher um. Am besten ist es natürlich, wenn Sie sich ein kommentiertes ROM-Listing und das Firmware-Handbuch besorgen, denn dann können Sie gezielt vorgehen um Ihren CPC besser kennenzulernen.

## 6. Der Bildschirm und sein Speicher

Die meisten Ausgaben der CPCs finden üblicherweise wohl über den Bildschirm statt. Dabei spielt es keine Rolle, ob es sich um eine Programmeingabe, oder um ein Anwendungsprogramm handelt.

Für einen Dialog mit dem Benutzer ist die optische Ausgabe auf einem Sichtschirm wohl auch am sinnvollsten. Ein guter Grund also, sich mit diesem Teil des CPC etwas näher zu beschäftigen.

Normalerweise liegt der Bildschirmspeicher dem oberen Teil des Firmware-ROMs parallel. Er kann aber auch auf andere Adressbereiche gelegt werden. Dabei wird es in den meisten Fällen aber nur sinnvoll sein, ihn nach &4000 zu legen, denn sonst kommt es zu Problemen mit den überlagerten ROM-Bereichen.

Wir befassen uns im Augenblick nur mit dem Normalzustand und nehmen deshalb also an, daß der Bildschirmbereich ab &C000 beginnt.

Der Bildschirmspeicher der CPCs ist ca. 16 KB groß, ist pixelorientiert und kann gedanklich in 2 KByte-Blöcke eingeteilt werden.

Ein Pixel stellt einen Bildpunkt dar. Aus 8\*8, also aus 64 Pixeln sind die normalen Zeichen des CPC aufgebaut. Wie diese Zeichen "aussehen", wird dadurch bestimmt, welche der Bildpunkte in dieser Zeichenmatrix gesetzt und welche nicht gesetzt sind.

Durch diese Pixelorientierung ist es beispielsweise beim CPC 464 nicht so ohne weiteres möglich, ASCII-Zeichen vom Bildschirm zu lesen. Bei den beiden anderen CPC-Brüdern hat Locomotive Software aber einen entsprechenden Befehl eingebaut (COPY CHR\$). Für die CPC 464-Besitzer und auch für an Maschinenprogramm-Interessierte werden wir nachher aufzeigen, wie bei allen drei CPC-Typen mit einem Programm ASCII-Zeichen vom Bildschirm gelesen und an eine andere Stelle wieder geschrieben werden können. Damit kann dann bei Programmen, die für alle drei Schneider gedacht sind mit der gleichen Methode gearbeitet werden.

Die Pixelorientierung und die Verwaltung des Bildschirms erscheint auf den ersten Blick sehr kompliziert und nicht gerade komfortabel. Was auf der einen Seite zunächst aber als Handicap des CPC aussieht, hat andererseits aber auch Vorteile. Im Gegensatz zu vielen vergleichbaren Mitbewerber-Computern, ist es beim CPC sehr einfach, Grafiken und Texte auf dem Schirm zu mischen.

Damit Sie - ohne nun schon viel lesen zu müssen - sehen, was diese Pixelorientierung bzw. eine derartige Bildschirmverwaltung

in der Praxis bedeutet, hier ein sehr kleines Demonstrations-Programm, das Ihnen aufzeigt wie der Bildschirm - speichermäßig - organisiert ist.

Geben Sie dieses Programm ein, lassen Sie es laufen und lesen Sie erst dann weiter.

### 6.1 Listing: SCREEN.BAS

```
100 CLS
110 INPUT "MODE (0-2)";m
120 MODE m
130 start=49152
140 w=255
150 sad=start
160 IF sad > 85526 THEN 200
170 POKE sad,w
180 sad=sad+1
190 LOCATE 1,15:PRINT HEX$(sad)
200 start=start+1
210 w=RND(1)*255:IF w=256 THEN w=1
220 GOTO 150
```

Dieses kleine Programm zeigt Ihnen in allen drei Modes, wie der Bildschirm im RAM "liegt". Außerdem können Sie beim Programmlauf auch erkennen, daß sich der zu pokende Wert (Zeile 140 und 210) in jedem Mode anders auf dem Bildschirm darstellt. Durch Deaktivierung (REM) der Zeile 210 erhalten Sie immer das gleiche Ausgabebild, die Zeile 210 erzeugt nur Zufallswerte.

Sie können auch ganz deutlich sehen, daß die Organisation - wie schon erwähnt - nicht zeilenweise, sondern immer im Sprung von acht Pixelzeilen erfolgt. Dabei beinhaltet ein Block immer 25 Pixelzeilen.

In Zeile 130 wird der Start für das Poken in den Speicher auf dezimal 49152, dies entspricht &C0000, gelegt.

Bei jedem Programmlauf wird dadurch (links oben beginnend) nach und nach der Bildschirm Pixelzeile für Pixelzeile gefüllt.

Durch Zeile 190 erfolgt die Ausgabe der gerade "bearbeiteten" Speicherstelle als Hexadezimalwert.

Eine Besonderheit der CPCs darf hier nun nicht unerwähnt bleiben: Die Startadresse des Bildschirminhaltes ist nicht immer &C0000!

Dadurch aber, daß wir bei jedem Programmlauf durch die Programmzeile 120 den Mode "setzen" erhalten wir tatsächlich den Beginn in der linken oberen Ecke des Bildschirms. Hätte der CPC aber ein Bildschirmscrolling (ein Auf- oder Abrollen des Bildschirminhaltes) ausgeführt, dann wäre dem nicht mehr so. Auch seitliches (horizontales) Scrollen ist möglich und verändert ebenfalls den Beginn des Bildschirmauslesens aus dem Speicher.

Der Start des Bildschirm-RAMs wird nämlich aus Geschwindigkeitsgründen über Zeiger verwaltet und diese werden durch ein "Roller" (Scrolling) des gesamten Bildschirminhaltes verändert. Diese Methode ist wesentlich schneller, als ein Umkopieren der Speicherinhalte.

Beim "Bildschirmverschieben", also dem "Scrollen", wird durch die Firmware-Routine SCR SET OFFSET der Bildschirm-Offset auf den neuen aktuellen Wert gebracht.

Dieser Offset steht beim 464 in den Speicherstellen &B1C9/B1CA und bei den anderen CPCs in &B7C4/B7C5 und gibt die Distanz zum Normalbeginn (&C000) an.

Um sich diesen Offset ausgeben zu lassen, können Sie folgenden Basiczeilen in das Programm SCREEN.BAS einfügen

```
CPC 464:      195 PRINT HEX$((PEEK(&B1C9)+PEEK(&B1CA)*256))
```

```
CPC 664/6128: 195 PRINT HEX$((PEEK(&B7C4)+PEEK(&B7C5)*256))
```

Diese Offset-Methode kann aber zu Problemen führen, wenn man sie nicht kennt. Und sie ist bisher anscheinend wenigen CPC-Benutzern bekannt und deswegen gibt es bei den "Nichtwissenden" oft große Schwierigkeiten mit Hardcopy-Routinen, oder auch beim Einlesen von "Bildern".

Wie sich ein derartiger Offset auswirkt, können Sie durch Hinzufügen einer weiteren Basiczeile sehr gut erkennen.

```
125 FOR I = 1 TO 201:PRINT "/////////";:NEXT I
```

Hierbei "bestimmen" das Schleifenziel und auch die Anzahl der auszugebenden Zeichen, wo der CPC nun mit seinen Pixelreihen beginnt. Die Zahl 201 haben wir gewählt, damit auch im Mode 2 dieser Effekt auftritt!

Wenn Sie diese Werte ändern und das Programm wieder starten, dann werden Sie an den unterschiedlichsten Bildschirmstellen den Start der Pixelreihen verfolgen können. Der CPC wird Ihnen zwar immer den Bildschirm mit dem gewählten oder zufallserzeugten Pixelbyte(s) vollschreiben, aber der "Start" liegt unterschiedlich.

Nun verstehen Sie sicher auch, weshalb es bei Hardcopy-Routinen und beim Einlesen von abgespeicherten Bildern zu Problemen kommen kann: Die Startadresse beim Auslesen des Bildschirm-RAMs ist nicht immer &C000!

Die Einstellung des "Screenstarts" auf &C000 kann von Basic aus nur durch den Mode-Befehl erfolgen. Der Befehl CLS hilft hier nicht! Auch der Befehl CLG, der den Grafikbildschirm löscht, zeigt keinerlei Wirkungen.

Sie können den Offsetwert zwar von Basic aus auch durch "POKEN" in die beiden zuständigen Speicherstellen verändern, aber was dann passiert können Sie sich mit dem nun folgenden kleinen Programm ansehen.



Die Grundadresse (ga) ist für den CPC 664/6128 ausgelegt. 464-Besitzer müssen der Variablen ga (in Zeile 100) den Wert &B1C9 zuweisen.

## 6.2 Listing: Offset

```
100 ga=&B7C4
110 POKE ga,0:POKE ga+1,0
120 FOR i = 1 TO 255:PRINT i;:NEXT
130 POKE ga,1:POKE ga+1,255
140 FOR i = 1 TO 255:PRINT i;:NEXT
150 POKE ga,0:POKE ga,0
```

Mit diesem kleinen Demonstrationsprogramm können Sie die Auswirkungen des Offsets sehr gut erkennen.

Dieses Verhalten des CPC ist wichtig zu wissen, wenn man Probleme mit Bildschirm- oder Hardcopy-Versatz vermeiden will. Denken Sie also immer rechtzeitig daran, den Offset durch den Befehl MODE rechtzeitig auf Null zu setzen und vermeiden Sie dann ein Bildschirmrollen!

## 6.3 Noch mehr Details über den Bildschirmspeicher

Um nun auch aufzuzeigen weshalb wir beim Speicherbedarf den Wert mit circa 16 KByte angegeben haben, betrachten wir uns nun die Bildschirmorganisation etwas näher. Dazu legen wir einfach fest, daß wir den Befehl MODE 2 eingegeben haben. (Auf irgendetwas müssen wir uns festlegen sonst wird es zu kompliziert!) Durch diesen Befehl ist der Start des Bildschirmspeichers &C000 und wir haben die 80-Zeichen-Darstellung gewählt.

Wie Sie auch Ihrem Handbuch entnehmen können, ist die Anzahl der Pixel in diesem Mode  $640 * 200$  also 128000 Punkte! Als Anzahl der Zeichen haben Sie  $80 * 25 = 2000$ . Wenn Sie nun die Anzahl der Pixel durch die Anzahl der Zeichen dividieren, dann erhalten Sie die Anzahl der Pixel, die für ein Zeichen zur Verfügung stehen nämlich 64. Dies entspricht der Zeichenmatrix von 8\*8 Pixeln und genau mit dieser Anordnung arbeitet Ihr CPC.

Andererseits aber können Sie nun auch feststellen, daß der Bildschirmspeicher größer ist als eigentlich nötig. Denn 2000 Zeichen mal 64 Pixel sind zwar die schon genannten 128000 Punkte, aber dafür werden doch nur 16000 Bytes benötigt! Wie Sie aber wissen, bedeuten 16 KByte 16384 Bytes!

Tatsächlich ist es so, daß der CPC eigentlich 384 Bytes zuviel "verbraucht". Bevor wir Ihnen mitteilen können, warum dies so ist, müssen wir noch etwas ausholen.

Betrachten wir deshalb alles einmal von einer anderen Warte aus. Machen Sie bitte die Zeilen 125 und 210 des erweiterten Listings SCREEN.BAS zu reinen Bemerkungszeilen und starten Sie dann einmal das Programm in den drei Modis.

Im Mode 0 erhalten Sie blau-rot blinkende Pixelzeilen.  
Im Mode 1 dagegen durchgehende rote Pixelzeilen.  
Im Mode 2 sind diese Zeilen gelb.

Ersetzen Sie nun bitte den Wert w in Zeile 140 durch 170.

Also:

170 POKE sad,170 !

Diese 170 sind binär dargestellt die Folge 10101010. Dies bewirkt beim Programmlauf, daß die Pixel abwechselnd gesetzt und nicht gesetzt werden.

Bei Programmläufen erhalten Sie nun folgende Darstellungen:

Mode 0 rot blau-blinkende unterbrochene Linien  
Mode 1 rote punktierte Linien  
Mode 2 gelbe punktierte Linien

Im Mode 2 werden Sie diese Punktierung anfangs evtl. nur schwer erkennen, warten Sie deshalb bitte, bis der Schirm etwas "beschrieben" ist, denn dann erkennen Sie bei einem Farbmonitor - je nach Ablenkungseinstellung und Genauigkeit der Farbmaske und der Farbpunkte - ein mehr oder weniger starkes Muster dem einer Holzmaserung ähnlich.

Auch im Mode 1 können Farbmonitorbesitzer diese "Verzeichnung" sehen.

Dieser Effekt tritt - bedingt durch einen anderen physikalischen Aufbau - bei einem Grünmonitor nicht auf.

Bei einem Grünmonitor sind also aufgrund der Tatsache, daß keine Lochmaske vorhanden ist diese Verzeichnungen nicht zu sehen, dafür sind aber im Vergleich zum Farbmonitor die einzelnen Pixel und auch der Pixelabstand besser zu erkennen.

Bei allen Monitoren sind aber gegenüber dem vorigen Wert von 255 - mehr oder weniger deutlich - die Unterbrechungen der ausgegebenen Linien zu sehen.

Im Mode 0 wirkt sich das Blinken der Pixel störend aus und deswegen sollten hypnotisch empfindliche Mitmenschen ein derartiges Bild nicht zu lange ansehen.

Andererseits kann bei richtiger Farbauswahl (INK) und Verändern der Farbwechselgeschwindigkeit durch den Befehl SPEED INK (vor allem beim Betrachten mit einer sogenannten 3-D-Brille) sogar ein "Tiefeneffekt" erzeugt werden.

Der Zwischenraum zwischen den gesetzten Pixeln ist jeweils immer die Hintergrund- (Paper-) Farbe.

Sie können nun ruhig mit den verschiedensten Werten experimentieren und erreichen dadurch die unterschiedlichsten Muster auf dem Bildschirm.

Wie Sie sich durch das Programm SCREEN.BAS überzeugen konnten, ist bei der Organisation immer ein Pixelzeilensprung von acht vorhanden. Was ist also zu tun, um die Pixelzeilen direkt untereinander zu erzeugen?

Hierzu muß beim Einschreiben in den Bildspeicher (durch POKEN) zum richtigen Zeitpunkt eben die Korrektur erfolgen. Wir hatten schon erwähnt, daß der Bildschirmspeicher in Blöcke von je 2 KByte organisiert ist, deshalb muß nach jedem Pixel-Zeilenende dieser Wert zur vorherigen Startadresse der Pixelzeile hinzuaddiert werden.

Wie dies beispielsweise erfolgen kann, zeigen wir an einem kleinen Programm-Beispiel auf. Dieses Programm gibt aber auch gleich noch weitere Informationen, denn es werden Ihnen in jedem Mode die Adressen der Pixelzeilenanfänge, -Ende und die momentan in Arbeit befindliche Speicheradresse angezeigt.

Da trotz der langen Programmlaufzeit von ca. 15 Minuten die Zeit für die Anzeige der einzelnen Adressen sehr kurz ist, merkt sich dieses Programm alle Adressen und speichert diese dann ab. Dadurch können Sie sich diese durch ein weiteres Programm auch wieder ausgeben lassen und sich eine eigene Tabelle erstellen, die Ihnen bei direktem Poken sehr hilfreich sein kann.

#### 6.4 Listing: PIXZEIL.BAS

```
100 'pixzeil
110 '
120 CLS:DIM pza$(200),pze$(200)
130 INPUT"Mode ";m:MODE m:FOR i = 1 TO 25:PRINT i:NEXT
140 s=&C0000:posi(0)=16:posi(1)=36:posi(2)=76:posi=posi(m)
150 '
160 FOR x=0 TO 7
170 pza=s+x*&800:pze=pza+&4F
180 '
190 zaehler=zaehler+1:pza$(zaehler)=HEX$(pza):pze$(zaehler)=HEX$(pze)
200 FOR i =pza TO pze:LOCATE 2,zeile+1
210 PRINT HEX$(i):LOCATE posi,zeile+1:PRINT HEX$(pze)
220 IF i<=&FFCF THEN POKE i,255
230 IF i =&FFCF THEN 280
240 x$=LOWER$(INKEY$):IF x$="e" THEN END ELSE NEXT i:NEXT x
250 LOCATE 2,zeile+1:PRINT HEX$(i-&3850):LOCATE posi,zeile+1:PRINT HEX$(i-1)
260 zeile = zeile +1:s=s+80:GOTO 150
270 '
280 LOCATE 2,zeile+1:PRINT HEX$(i-&384F):CALL &BB06
290 WINDOW #1,1,(posi+4),1,9:CLS:o=1
300 '
310 y=1:zz=0:PRINT #o,"Zeichenzeile "y
320 FOR i = 1 TO 200
330 PRINT #o,pza$(i);" ";
340 PRINT #o,pze$(i):zz=zz+1
350 IF zz<8 THEN 370 ELSE y=y+1:IF o=1 THEN CALL &BB06
360 IF y=26 THEN 390 ELSE PRINT #o,"Zeichenzeile "y:zz=0
370 NEXT i
380 '
114
```

```

390 IF o=9 THEN CLOSEOUT:CLS:END
400 CLS #1:PRINT #1,"Abspeichern ?
410 x$=LOWER$(INKEY$):IF x$="" THEN 410
420 IF x$<>"j" THEN END ELSE OPENOUT "pixzadr.asc":o=9:GOTO 310

```

Dieses Programm kann in allen drei Betriebsmodis arbeiten und sollte von Ihnen auch in jeder dieser Betriebsart einmal gestartet werden.

Einmal sollten Sie das Programm sogar komplett ablaufen lassen, denn dann werden alle - für Sie wichtigen - Informationen berechnet und außerdem auch auf Ihren Massenspeicher abgespeichert.

Damit Sie diese abgespeicherten Werte auch wieder einlesen können, sollten Sie ein kleines Programm schreiben, das in etwa dem nachfolgenden entspricht:

```

100 'pixadles
110 '
120 OPENIN"pixzadr.asc"
130 INPUT #9,a$:PRINT a$
140 IF NOT EOF THEN 130
150 CLOSEIN

```

Durch Einfügen weiterer Programmzeilen können Sie dann das Ausgabeformat Ihren Wünschen anpassen.

Doch zunächst noch einmal zum Programm PIXZEIL.BAS.

Beim Programmstart werden Sie nach dem MODE gefragt und können diesen frei wählen, um sich die Unterschiede der Zeichenablage vorab anzusehen.

Nach der Wahl des MODE werden die Zeilennummern für normale Zeichen ausgegeben und dann erfolgt als nächstes die Ausgabe der ersten Adresse (Startadresse) der jeweiligen Pixelzeile als Hexzahl im linken Bildschirmteil. Gleichzeitig wird Ihnen am rechten Bildrand die Endadresse dieser Zeile angegeben.

Die linke Zahl aber wird dann für jede Speicherstelle hochgezählt und "läuft" deshalb. Sind acht unmittelbar untereinanderliegende Pixelzeilen ausgegeben, dann wird diese Zahl wieder auf die Anfangsadresse der ersten Pixelzeile dieses "Achtersegmentes" gestellt und mit den nächsten acht Pixelzeilen begonnen usw.

In der Zwischenzeit sehen Sie, wie sich der Bildschirm mehr und mehr füllt. Mit welcher Farbe diese Füllung erfolgt, hängt vom gewählten Mode ab.

Durch Betätigung der Taste "E" können Sie den Programmlauf aber auch beenden, wenn Sie der Meinung sind genug gesehen zu haben. Eine Aufstellung aller Zeilen mit den Anfangs- und Endadressen der Pixelzeilen haben wir mit diesem Programm für Sie bereits erstellt.

## 6.5 Tabelle der Pixelzeilenadressen

PZ	Zeichenzeile 1		Zeichenzeile 2		Zeichenzeile 3		Zeichenzeile 4	
	Beginn	Ende	Beginn	Ende	Beginn	Ende	Beginn	Ende
01	C000	C04F	C050	C09F	C0A0	C0EF	C0F0	C13F
02	C800	C84F	C850	C89F	C8A0	C8EF	C8F0	C93F
03	D000	D04F	D050	D09F	D0A0	D0EF	D0F0	D13F
04	D800	D84F	D850	D89F	D8A0	D8EF	D8F0	D93F
05	E000	E04F	E050	E09F	E0A0	E0EF	E0F0	E13F
06	E800	E84F	E850	E89F	E8A0	E8EF	E8F0	E93F
07	F000	F04F	F050	F09F	F0A0	F0EF	F0F0	F13F
08	F800	F84F	F850	F89F	F8A0	F8EF	F8F0	F93F

PZ	Zeichenzeile 5		Zeichenzeile 6		Zeichenzeile 7		Zeichenzeile 8	
	Beginn	Ende	Beginn	Ende	Beginn	Ende	Beginn	Ende
01	C140	C18F	C190	C1DF	C1E0	C22F	C230	C27F
02	C940	C98F	C990	C9DF	C9E0	CA2F	CA30	CA7F
03	D140	D18F	D190	D1DF	D1E0	D22F	D230	D27F
04	D940	D98F	D990	D9DF	D9E0	DA2F	DA30	DA7F
05	E140	E18F	E190	E1DF	E1E0	E22F	E230	E27F
06	E940	E98F	E990	E9DF	E9E0	EA2F	EA30	EA7F
07	F140	F18F	F190	F1DF	F1E0	F22F	F230	F27F
08	F940	F98F	F990	F9DF	F9E0	FA2F	FA30	FA7F

PZ	Zeichenzeile 9		Zeichenzeile 10		Zeichenzeile 11		Zeichenzeile 12	
	Beginn	Ende	Beginn	Ende	Beginn	Ende	Beginn	Ende
01	C280	C2CF	C2D0	C31F	C320	C36F	C370	C3BF
02	CA80	CACF	CAD0	CB1F	CB20	CB6F	CB70	CBBF
03	D280	D2CF	D2D0	D31F	D320	D36F	D370	D3BF
04	DA80	DACF	DAD0	DB1F	DB20	DB6F	DB70	DBBF
05	E280	E2CF	E2D0	E31F	E320	E36F	E370	E3BF
06	EA80	EACF	EAD0	EB1F	EB20	EB6F	EB70	EBBF
07	F280	F2CF	F2D0	F31F	F320	F36F	F370	F3BF
08	FA80	FACF	FAD0	FB1F	FB20	FB6F	FB70	FBBF

PZ	Zeichenzeile 13		Zeichenzeile 14		Zeichenzeile 15		Zeichenzeile 16	
	Beginn	Ende	Beginn	Ende	Beginn	Ende	Beginn	Ende
01	C3C0	C40F	C410	C45F	C460	C4AF	C480	C4FF
02	CBC0	CC0F	CC10	CC5F	CC60	CCAF	CCB0	CCFF
03	D3C0	D40F	D410	D45F	D460	D4AF	D4B0	D4FF
04	DBC0	DC0F	DC10	DC5F	DC60	DCAF	DCB0	DCFF
05	E3C0	E40F	E410	E45F	E460	E4AF	E4B0	E4FF
06	ECB0	EC0F	EC10	EC5F	EC60	ECAF	ECB0	ECFF
07	F3C0	F40F	F410	F45F	F460	F4AF	F4B0	F4FF
08	FBC0	FC0F	FC10	FC5F	FC60	FCAF	FCB0	FCFF

PZ	Zeichenzeile 17		Zeichenzeile 18		Zeichenzeile 19		Zeichenzeile 20	
	Beginn	Ende	Beginn	Ende	Beginn	Ende	Beginn	Ende
01	C500	C54F	C550	C59F	C5A0	C5EF	C5F0	C63F
02	CD00	CD4F	CD50	CD9F	CDA0	CDEF	CDF0	CE3F
03	D500	D54F	D550	D59F	D5A0	D5EF	D5F0	D63F
04	DD00	DD4F	DD50	DD9F	DDA0	DDEF	DDF0	DE3F
05	E500	E54F	E550	E59F	E5A0	E5EF	E5F0	E63F
06	ED00	ED4F	ED50	ED9F	EDA0	EDEF	EDF0	EE3F
07	F500	F54F	F550	F59F	F5A0	F5EF	F5F0	F63F
08	FD00	FD4F	FD50	FD9F	FDA0	FDEF	FDf0	FE3F

PZ	Zeichenzeile 21		Zeichenzeile 22		Zeichenzeile 23		Zeichenzeile 24	
	Beginn	Ende	Beginn	Ende	Beginn	Ende	Beginn	Ende
01	C640	C68F	C690	C6DF	C6E0	C72F	C730	C77F
02	CE40	CE8F	CE90	CEDF	CEE0	CF2F	CF30	CF7F
03	D640	D68F	D690	D6DF	D6E0	D72F	D730	D77F
04	DE40	DE8F	DE90	DEDF	DEE0	DF2F	DF30	DF7F
05	E640	E68F	E690	E6DF	E6E0	E72F	E730	E77F
06	EE40	EE8F	EE90	EEDF	EEE0	EF2F	EF30	EF7F
07	F640	F68F	F690	F6DF	F6E0	F72F	F730	F77F
08	FE40	FE8F	FE90	FEDF	FEE0	FF2F	FF30	FF7F

PZ	Zeichenzeile 25	
	Beginn	Ende
01	C780	C7CF
02	CF80	CFCF
03	D780	D7CF
04	DF80	DFCF
05	E780	E7CF
06	EF80	EFCF
07	F780	F7CF
08	FF80	FFCF

Durch diese Aufstellung können Sie nun auch die schon genannte Einteilung in Blöcke zu 2 KB erkennen. Der erste Block beginnt mit Pixelzeile (=PZ) 1 der Zeichenzeile 1 und umfasst weiterhin alle ersten Pixelzeilen aller weiteren Zeichenzeilen. Der zweite Block betrifft alle Pixelzeilen 2 usw.

Der hier verwendete Begriff der "Zeichenzeile" bedeutet eine ganz normale Zeile wie sie für die übliche Zeichendarstellung verwendet wird. Das Cursorzeichen beispielsweise ist acht Pixelzeilen (PZ), also eine Zeichenzeile hoch!

Wenn Sie nun diese Aufstellung genau überprüfen, dann können Sie feststellen, daß die Adressen

&C7D0 bis &C7FF	&CFD0 bis &CFFF
&D7D0 bis &D7FF	&DFD0 bis &DFFF
&E7D0 bis &E7FF	&EFD0 bis &EFFF
&F7D0 bis &F7FF	&FFD0 bis &FFFF

laut dieser Aufstellung nicht benutzt werden. Genau das sind die Adressen, die vom CPC anscheinend zuviel "reserviert" wurden.

Aber der Schein trügt, denn dies ist nur der Fall, wenn kein Offset des Bildschirms vorliegt. Ist ein Offset aber gegeben, dann kommen auch diese "brachliegenden" Bytes zum Einsatz und andere sind an deren Stelle ungenutzt.

Aus diesem Grunde können diese Bytes nur dann andersweitig benutzt werden, wenn gewährleistet ist, daß - während der Benutzung dieser Speicherstellen durch ein Anwenderprogramm - kein Schirmscrolling vorkommt.

Durch die vorherigen Experimente konnten Sie nun feststellen, daß die Verschlüsselung eines Bytes im Bildschirm-RAM je nach Mode unterschiedlich ist. Wie diese Codierung entschlüsselt aussieht erfahren Sie etwas später, denn zuvor wollen wir uns noch ein paar kleine Scherze erlauben, und den CPC etwas austricksen.

Geben Sie doch nachfolgende Programme einmal in Ihren CPC ein und lassen Sie sie laufen!

```
100 'MODEM01
110 '
120 CLS:PRINT"ist dies ein CPC 464 (j/n)
130 x$=LOWER$(INKEY$):IF x$="" THEN 130
140 IF x$="j" THEN modebyte =&B1C8 ELSE modebyte =&B7C3
150 '
160 MODE 0:a$="*****":GOSUB 180:POKE modebyte,1:GOSUB 180:POKE modebyte,2
170 GOSUB 180:POKE modebyte,0:PEN 1:END
180 FOR i = 1 TO 15:PEN i:PRINT a$;:NEXT:PRINT:RETURN
```

Wie Sie dem Programmlisting entnehmen können, soll außer der Frage nach dem CPC-Typ nur A\$ ausgegeben werden.

Allerdings wird zwischen den verschiedenen Ausgaben der Modi "umgePOKEt" und welche Effekte sich dadurch ergeben, sehen Sie auf dem Bildschirm.

Hier sind die Besitzer eines Farbmonitors besser bedient, denn diese können die gesamte Farbenpracht dieses kleinen Scherzes sehen. Ausgewählt wurde nämlich der Mode 0 und für diesen wurden die ersten fünf Zeilen auch korrekt ausgegeben. Die nächsten fünf Zeilen der Ausgabe kamen durch umPOKEt auf Mode 1 und die restlichen durch umPOKEt auf Mode 2 zustande.

Hat es Spaß gemacht? Falls ja, dann gleich noch das nächste Programm in Ihren Schneider eingeben.

```
100 MODE 1:a$="*****":GOSUB 200:POKE &B7C3,0:GOSUB 200:POKE &B7C3,2
150 GOSUB 200:POKE &B7C3,1:PEN 1:END
200 FOR i=1 TO 15:PEN i:PRINT a$;:NEXT:PRINT:RETURN
```

(464-Besitzer bitte den Wert &B7C3 in &B1C8 ändern!)

Wie Sie sehen ist dies nur eine kleine Variante des vorherigen Programmes!

Und nun noch ein weiterer Scherz, bei dem die 464-Besitzer allerdings erst alle POKE-Adressen anpassen müssen.

CPC 664/6128	CPC 464	Sinn
&B7C3	=&B1C8	= MODE-NUMMER
&B7C4 und &B7C5	=&B1C9 und &B1CA	= SCR OFFSET
&B7C6	=&B1CB	= SCR BASE HIGH

```

100 MEMORY &3FFF:POKE &B7C6,&40:MODE 0
110 POKE &B7C3,1:POKE &B7C4,1:POKE &B7C5,2
130 GOSUB 170
140 POKE &B7C4,1:POKE &B7C5,255
150 GOSUB 170
160 RUN
170 FOR I = 1 TO 255
180 IF LOWER$(INKEY$) = "e" THEN 230
190 PRINT I;
200 IF PEEK(&B7C6) = &C0 THEN POKE &B7C6,&40 ELSE POKE &B7C6,&C0
210 NEXT I:RETURN
220 '
230 POKE &B7C6,&C0:MODE 2

```

Versuchen Sie nun doch einmal herauszufinden, was bei diesem Programm alles an Unsinn getrieben wird!

SCR BASE HIGH ist die Speicherstelle, in welcher das Highbyte des Bildschirmstartes abgelegt ist!

Mit etwas Phantasie können Sie zwar lesen, was ausgegeben wird, aber mit den sonst üblichen Zeichen hat die Ausgabe eigentlich nur noch sehr wenig zu tun. Brechen Sie das Programm durch Eingabe von "e" ab.

Nun wieder ernsthaft

Wie Sie sehen konnten, ist die Codierung der Bytes in den verschiedenen Modis unterschiedlich und dies wollen wir nun näher betrachten.

Im Mode 2 ist alles sehr einfach zu überblicken, jedes Bit entspricht einem Pixel. Ein gesetztes Bit erscheint in der Farbe des Farbstiftes und ein nicht gesetztes Bit ist auf dem Bildschirm in der Farbe des Hintergrundes zu sehen. In diesem Mode kann der CPC nur zwei Farben darstellen.

Im Mode 1 beträgt die Auflösung nur noch 320 mal 200 Punkte und der CPC ist in der Lage vier Farben gleichzeitig darzustellen.



Nun muß die Codierung in den Schirmspeicher-Bytes anders erfolgen, denn sonst wäre eine Darstellung mit vier Farben, wenn der erforderliche Bildschirm-Speicherplatz nicht vergrößert werden würde, nicht möglich. Gleiches gilt dann auch für den Mode 0.

Prinzipiell werden in einem Byte unabhängig vom Mode immer acht Pixel codiert. Dies bedeutet aber, daß dann im Mode 0 insgesamt vier Bytes für eine Zeichenbreite benötigt werden. Auch für diese Betrachtungen gleich ein Programm.

## 6.6 Listing: PIXCODE12.BAS

```

100 'pixcode12
110 '
120 INPUT "MODE";m:IF m=0 THEN m=1
130 MODE m:INK 1,24:INK 2,13:INK 3,23,6:t$="CS":laenge=2:z=z+1:GOTO 200
140 LOCATE 1,1:PRINT t$
150 x1=PEEK(&C000):y1=PEEK(&C001):x=x1:y=y1
160 IF z=1 THEN LOCATE 1,2 ELSE LOCATE 1,10
170 IF z=3 THEN LOCATE 1,18
180 z=z+1:GOSUB 450
190 '
200 IF z<2 THEN PEN z ELSE IF z>3 THEN PEN 1:GOTO 250
210 FOR j = 48 TO 50:POKE &C2D0+j,&F0:POKE &C550+j,&F0:NEXT j
220 PEN z:LOCATE 15,1:PRINT "Pen "z
230 GOTO 140
240 '
250 LOCATE 26,7:PRINT "Steuerung mit":LOCATE 26,8:PRINT "Cursor-Tasten
260 LOCATE 26,11:PRINT "E = Ende
270 x=PEEK(&C000):y=PEEK(&C001)
280 LOCATE 26,1:PRINT "Pixelreihe "pr+1
290 LOCATE 26,3:PRINT "links = "HEX$(x,2)
300 LOCATE 26,4:PRINT "rechts= "HEX$(y,2)
310 x$=LOWER$(INKEY$):IF x$="" THEN 320
320 IF x$="e" THEN PEN 1:MODE 2:END
330 IF ASC(x$)=224 THEN 540
340 IF ASC(x$)=240 THEN pr=pr+1:IF pr>7 THEN pr=0
350 IF ASC(x$)=241 THEN pr=pr-1:IF pr<0 THEN pr=7
360 IF ASC(x$)=242 THEN x=x+1:IF x>255 THEN x=0
370 IF ASC(x$)=243 THEN y=y+1:IF y>255 THEN y=0
380 IF ASC(x$)=244 THEN x=x-1:IF x<0 THEN x=255
390 IF ASC(x$)=245 THEN y=y-1:IF y<0 THEN y=255
400 '
410 LOCATE 26,1:PRINT "W A R T E N !
420 LOCATE 1,18:GOSUB 450:GOTO 290
430 '
440 POKE &C000+pr*800,x:POKE &C001+pr*800,y
450 FOR pz = &C000 TO &FFFF STEP &800
460 FOR i=0 TO laenge-1
470 w=PEEK(pz+i)
480 a$=BIN$(w,8)
490 PEN 1:PRINT a$;" ";:PRINT HEX$(w,2);" ";:PEN z
500 NEXT i:PRINT:NEXT pz
510 RETURN
520 '
530 z=z+1:IF z>3 THEN z=1
540 '
120

```

```

550 PEN z
560 LOCATE 1,1:PRINT t$:LOCATE 15,1:PRINT"Pen "z
565 x=PEEK(&C000):y=PEEK(&C001)
570 PEN z
580 GOTO 420

```

Wenn Sie dieses Programm eingegeben haben, dann speichern Sie es bitte zuerst ab (wie üblich) und starten es erst dann.

Beim ersten Programmstart sollten Sie bei der Frage nach dem MODE mit 2 antworten!

Nach dieser Eingabe erscheint links oben das Buchstabenpaar "CS". Daneben wird während der nachfolgenden Ausgaben der aktive Stift (PEN) angezeigt.

CS		Pen	2	Pixelreihe	1
00110000	30	11000000	C0	links = 03	
01100000	60	01100000	60	rechts= 0C	
11000000	C0	00000000	00		
11000000	C0	00000000	00		
01100000	60	01100000	60		
00110000	30	11000000	C0		
00000000	00	00000000	00		
00000011	03	00001100	0C		
00000110	06	00000110	06		
00001100	0C	00000000	00		
00001100	0C	00000000	00		
00001100	0C	00000000	00		
00000110	06	00000110	06		
00000011	03	00001100	0C		
00000000	00	00000000	00		
00000011	03	00001100	0C		
00000110	06	00000110	06		
00001100	0C	00000000	00		
00001100	0C	00000000	00		
00001100	0C	00000000	00		
00000110	06	00000110	06		
00000011	03	00001100	0C		
00000000	00	00000000	00		

Steuerung mit  
Cursor-Tasten

E = Ende

Abbildung 10: Diese Hardcopy zeigt was Sie im MODE 1 zu sehen bekommen.

Darunter erfolgt die Ausgabe der binären Werte für die einzelnen Pixelzeilen der beiden links oben stehenden Buchstaben "CS".

Jeweils immer acht untereinanderstehende Wertezeilen sind für ein (MODE 1) oder zwei (MODE 2) Zeichen zuständig. (Jede Zeile zeigt die ersten Zeichen einer Pixelzeile an.)

Rechts neben den Binärzahlen steht jeweils der hexadezimale Wert. Während der Ausgabe wird von Pen 1 auf Pen 2 und dann Pen 3 umgeschaltet und jeweils hierfür die Binärwerte ausgegeben.

Da wir beim ersten Mal im 80-Zeichen-Mode arbeiten, sind nur zwei Farben möglich und aus diesem Grund wird beim automatischen Um-

schalten auf PEN 2 kurzzeitig alles was mit diesem Stift geschrieben wurde, unsichtbar.

Das heißt, es werden zwar für jeden der beiden Buchstaben insgesamt  $3 * 8 = 24$  Pixelzeilen angezeigt, aber der mittlere Teil enthält - in diesem Mode - nur Nullen.

Sind die Ausgaben für die drei PENS erfolgt, dann erscheint in der Mitte oben die Angabe der Pixelzeile. Wenn Sie noch keine weiteren Eingaben gemacht haben, dann steht dort: Pixelzeile 1.

Darunter sehen sie die Angabe links = 3C und rechts = 3C. Diese Werte sind die Hexadezimalwerte der momentan "eingeschalteten" Pixelzeile. Links gilt dabei für den ersten Buchstaben (C) und rechts für den zweiten Buchstaben (S).

Jedes gesetzte Bit wird durch eine "1" im Binärfeld angezeigt. Wenn Sie die Bildschirmdarstellung aus einer größeren Entfernung betrachten, sehen Sie im oberen und unteren Teil des Binärfeldes die beiden Buchstaben "CS". Denn während des Programmlaufes wurden die beiden Speicherstellen &C000 und &C001 decodiert.

Das Programm kann nun mit den Cursor-Steuertasten weiter bedient werden.

Betätigen Sie nun einmal die COPY-Taste. Dadurch wird wieder auf PEN 1 geschaltet! Wenn Sie nun die Cursorsteuertaste links ohne gedrückte Shift-Taste betätigen, dann wird der Wert bei "links" erhöht, mit gedrückter Shift-Taste erniedrigt. Gleichzeitig können Sie beobachten, wie sich im unteren Teil des Binärfeldes in der zuständigen Zeile der Wert ändert und auch das Zeichen "C" links oben unterliegt diesen Änderungen. Das heißt, Sie können nun diese oberste Pixelzeile des Zeichens "C" ändern.

Sinngemäß gilt das gleiche für die Taste Cursor nach rechts, nur daß sich dann der zweite Buchstabe, das "S" ändert.

Mit den Tasten "Cursor nach oben" und "Cursor nach unten" können Sie sich dann auch die anderen Pixelzeilen "vornehmen".

Sie können mit diesem Programm nun sehen, daß im 80 Zeichen-Mode tatsächlich in jedem Bildspeicherbyte acht Pixel-Zeichen codiert sind und diese auch nach Belieben verändern.

Da die obere Pixelanzeige des Binärfeldes stehen bleibt, können Sie, wenn Sie Ihre Zeichen völlig "zerstört" haben, mit diesen Werten Ihre beiden Zeichen wieder "restaurieren".

Für ganz "Eilige" gibt es aber auch die Möglichkeit mittels der COPY-Taste die Rückstellung durchzuführen. Sie müssen diese Taste nur so oft drücken bis wieder PEN 1 oder PEN 3 erscheint.

Wenn Sie sich dies alles dann angesehen haben, dann sollten Sie in den MODE 1 umschalten. Dazu geben Sie bitte "e" ein und starten das Programm neu.

Die Frage nach dem Mode müssen Sie dann mit "1" beantworten!

Der Programmablauf ist der gleiche, nur wie Sie sofort erkennen können, blinkt ein Teil der Buchstaben. Es sind die Buchstaben, die mit PEN 3 geschrieben wurden. Außerdem, und das ist nun das wichtige an der Binärausgabe, sehen Sie nun nicht mehr die beiden Buchstaben CS, sondern nur noch den Buchstaben "C" und selbst dieser ist nicht mehr in einem Feld enthalten, sondern auf die beiden Felder aufgeteilt. Da im Mode 1 vier Farben möglich sind, werden insgesamt zwei Bytes für ein Zeichen benötigt.

Auch jetzt sollten sie mit dem Programm etwas arbeiten, damit Sie die Veränderungen sehen. Die Bedienung des Programmes ist sinngemäß die gleiche wie im MODE 2.

Leider kommen nur Farbmonitorbesitzer in den Genuß, alle kleinen Detail-Veränderungen am Zeichen "C" zu erkennen, denn manche Unterschiede wirken sich am Grünmonitor kaum aus, da die Helligkeitsdifferenzen verschiedener Farben zu gering sind.

Was Sie aber sehr gut erkennen können ist die Tatsache, daß je nach gewähltem PEN das Zeichen "C" anders codiert wird. Auf die Dekodierung im MODE 1 wollen wir aber gar nicht näher eingehen, sondern gleich die nächste Schwierigkeitsstufe wählen und alles von MODE 0 aus betrachten, denn dann können Sie auch die Verschlüsselung des MODE 1 verstehen, und wir brauchen es nur einmal zu erklären.

Auch hierfür haben wir für Sie verschiedene Programme geschrieben, mit denen Sie sich Ihr Wissen erarbeiten können.

## 6.7 Codierung der Zeichen im MODE 0

Während für die binäre Darstellung von 2 Bytes der MODE 1 noch ausreicht, ist es bei vier Bytes im MODE 0 nicht mehr sinnvoll darstellbar. Aus diesem Grunde schaltet das nachfolgende Programm zur Anzeige der Auswertungen immer in den Mode 2 um. Außerdem wird auch nur die oberste Pixelreihe eines Zeichens decodiert, denn für das Verständnis reicht es aus.

Vorab möchten wir aber darauf hinweisen, daß in einigen Büchern zu den CPCs die Codierung der einzelnen Pixel falsch dargestellt ist! Deshalb sollte derjenige, der sich näher mit der Verschlüsselung beschäftigen will, alles bisher gelesene gegebenenfalls vergessen und sich anhand der nachfolgenden Programme alles neu ansehen! Scheinbar stimmen zwar die Angaben in anderen Büchern, aber nur scheinbar!

Denn dort wird es größtenteils so dargestellt, als wenn die Codierung mit den Farben zusammenhängen würde. Dies stimmt aber nicht!

Zwar kann man bei oberflächlicher Betrachtung zu diesem Schluß kommen, aber die richtige Codierung erkennt man sehr leicht, wenn man mittels des Plotbefehles eine Pixelgruppe zeichnen läßt!

Hierzu können Sie das nachfolgende Programm einmal derart abändern, daß durch Zeile 180 nicht nach 450, sondern nach 460 gesprungen wird.

### 6.7.1 Listing: MODE0COD.BAS

```

100 'mode0cod - Decodierung MODE 0
110 MODE 1:PRINT"Im Mode 0 werden -anstelle von einem-
120 PRINT"immer vier Punkte angesprochen !":PRINT
130 p$="Pixel element"
140 INPUT "Zeichen";a$
150 INPUT "Pen";p:IF p=0 THEN p = 1
160 MODE 0:PEN p
170 '
180 GOSUB 450:FOR i = 1 TO 500:NEXT i
190 '
200 f1$=BIN$(PEEK(&C000),8):f2$=BIN$(PEEK(&C001),8)
210 f3$=BIN$(PEEK(&C002),8):f4$=BIN$(PEEK(&C003),8)
220 '
230 MODE 2:PEN 1
240 p0$=MID$(f1$,7,1)+MID$(f1$,3,1)+MID$(f1$,5,1)+MID$(f1$,1,1)
250 p1$=MID$(f1$,8,1)+MID$(f1$,4,1)+MID$(f1$,6,1)+MID$(f1$,2,1)
260 p2$=MID$(f2$,7,1)+MID$(f2$,3,1)+MID$(f2$,5,1)+MID$(f2$,1,1)
270 p3$=MID$(f2$,8,1)+MID$(f2$,4,1)+MID$(f2$,6,1)+MID$(f2$,2,1)
280 p4$=MID$(f3$,7,1)+MID$(f3$,3,1)+MID$(f3$,5,1)+MID$(f3$,1,1)
290 p5$=MID$(f3$,8,1)+MID$(f3$,4,1)+MID$(f3$,6,1)+MID$(f3$,2,1)
300 p6$=MID$(f4$,7,1)+MID$(f4$,3,1)+MID$(f4$,5,1)+MID$(f4$,1,1)
310 p7$=MID$(f4$,8,1)+MID$(f4$,4,1)+MID$(f4$,6,1)+MID$(f4$,2,1)
320 '
330 MODE 2:PEN 1:PRINT"Das gewuenschte Zeichen war : "a$
340 '
350 LOCATE 1,5:PRINT"Die Speicherinhalte:":PRINT
360 PRINT"&c000      &c001      &c002      &c003"
370 PRINT f1$,f2$,f3$,f4$:PRINT
380 '
390 PRINT:PRINT"Der Code der obersten Pixelreihe:":PRINT
400 PRINT p0$,"1. "p$;" (links)":PRINT p1$,"2. "p$;PRINT p2$,"3. "p$
410 PRINT p3$,"4. "p$;PRINT p4$,"5. "p$;PRINT p5$,"6. "p$
420 PRINT p6$,"7. "p$;PRINT p7$,"8. "p$;" (rechts)"
430 END
440 '
450 PRINT a$;:RETURN
460 PLOT 0,390,p:RETURN

```

Im MODE 2 war in einem Byte der Code für 8 waagerechte und nebeneinander liegende Pixel enthalten. Dabei war es nur möglich mittels zweier Farben, nämlich der des Hintergrundes und der des Schreibstiftes darzustellen, d. h. eine dieser Farben zu wählen.

Das bedeutete, war ein Pixel gesetzt (hatte binär betrachtet also eine "1"), dann erschien der Bildpunkt in der Farbe des gewählten PENS ansonsten war die Hintergrund- (PAPER-) Farbe zu sehen.

Wie Sie bestimmt schon festgestellt haben, ist die Breite der Pixel in den verschiedenen Betriebsarten unterschiedlich. Um dies

besser betrachten und verstehen zu können, können Sie nachfolgendes Programm in den drei Betriebsarten laufen lassen.

### 6.7.2 Listing: AUFLOES.BAS

```

100 'AUFLOES.BAS
110 '
120 INPUT "MODE ";m:MODE m;z%=CHR$(143)
130 IF m=0 THEN anz=20 ELSE IF m=1 THEN anz=40 ELSE IF m=2 THEN anz=80
140 LOCATE 1,3:PRINT z$:LOCATE 19,13:PRINT z$:LOCATE anz,14:PRINT z$
150 ORIGIN 0,206:TAG:PRINT"Pixelaufloesung";:TAGOFF:ORIGIN 0,0
160 '
170 DRAW 639,400 'Diagonale zeichnen
180 '
190 PLOT 300,100:PLOT 302,99:PLOT 302,98 'Punkte plotten
200 '
210 'horizontal plotten, nur jeder 2. Wert wird geplottet !
220 FOR i = 0 TO 8*anz STEP 10:PLOT i,0:NEXT
230 FOR i = 0 TO 8*anz:PLOT i,1:NEXT:FOR i = 0 TO 8*anz:PLOT i,2:NEXT
240 FOR i = 0 TO 8*anz:PLOT i,3:NEXT
250 '
260 'horizontal Mitte plotten
270 FOR i = 0 TO 639:PLOT i,206:NEXT i:FOR i = 0 TO 638:PLOT i,204:NEXT i
280 FOR i = 0 TO 637:PLOT i,202:NEXT i:FOR i = 0 TO 636:PLOT i,200:NEXT i
290 FOR i = 0 TO 635:PLOT i,198:NEXT i:FOR i = 0 TO 634:PLOT i,196:NEXT i
300 FOR i = 0 TO 633:PLOT i,194:NEXT i:FOR i = 0 TO 632:PLOT i,192:NEXT i
310 '
320 'vertikal links
330 FOR i = 0 TO 400 STEP 10:PLOT 0,i:NEXT i
340 FOR i = 0 TO 398:PLOT 1,i:NEXT i:FOR i = 0 TO 396:PLOT 2,i:NEXT i
350 FOR i = 0 TO 394:PLOT 3,i:NEXT i:FOR i = 0 TO 392:PLOT 4,i:NEXT i
360 FOR i = 0 TO 390:PLOT 5,i:NEXT i:FOR i = 0 TO 388:PLOT 6,i:NEXT i
370 FOR i = 0 TO 386:PLOT 7,i:NEXT i
380 '
390 ORIGIN 0,206:TAG:PRINT"Pixelaufloesung";:TAGOFF:ORIGIN 0,0
400 '
410 'Ausgabe blinkendes Cursorsymbol
420 LOCATE 1,3:PRINT " ":LOCATE 1,3:PRINT CHR$(143)
430 LOCATE 19,13:PRINT " ":LOCATE 19,13:PRINT CHR$(143)
440 IF LOWER$(INKEY$)="e" THEN LOCATE 1,5:END ELSE 420

```

Durch dieses Programm können Sie sehr gut erkennen, daß zum Beispiel das Cursor-Symbol (=chr\$(143)) acht Pixel breit und auch ebensoviel hoch ist. Weiterhin sehen Sie auch, daß der Vertikalabstand der Pixelzeilen im 2er-Rhythmus erfolgt. Das heißt, daß Sie zwar maximal nur 200 Zeilen haben, aber der höchste Wert eben 400 ist, usw.

Achten Sie deshalb auch auf die erste punktierte Linie, die horizontal ausgegeben wird. Obwohl in der nachfolgenden Programmzeile für die zweite Horizontalinie (für die Y-Achse) ein anderer Wert angegeben ist, wird die erste punktierte Linie überschrieben.

Auch die Form der Abschrägungen am Ende des Vertikal- und des mittleren Horizontalbalkens (nach dem Plotvorgang) gibt Ihnen

weitere Hinweise, die im Zusammenhang mit der Dekodierung wichtig sind. Das Programmende erfolgt durch Eingabe des Buchstabens "e".

Um sich nun die Codierung im Mode 0 anzusehen, lassen Sie bitte das Programm MODE0COD.BAS laufen.

Als erstes wird Ihnen mitgeteilt, daß im MODE 0 - im Gegensatz zum MODE 2 - immer 4 Punkte angesprochen werden. Dieser Hinweis bedeutet, daß im MODE 0 ein "Bildpunkt" viermal so breit ist wie im MODE 2. Danach sollen Sie irgendein Zeichen eingeben. Dies sollte aber ein Zeichen sein, welches in der obersten Horizontalpixelzeile auch gesetzte Punkte hat, denn sonst sehen Sie nichts!

Wenn Sie dieses Zeichen eingegeben haben, werden Sie nach dem PEN gefragt, mit dem dieses Zeichen geschrieben werden soll. Haben Sie auch diesen bestimmt, dann wird das Zeichen im MODE 0 ausgegeben und die Speicherstellen &C000 bis &C003 dekodiert.

Danach schaltet das Programm in den MODE 2 um, sagt Ihnen nochmals welches Zeichen dargestellt werden sollte und zeigt Ihnen die Binärwerte der abgefragten Speicherstellen.

Darunter erfolgt dann die Ausgabe der richtig zusammengehörigen Bits der obersten Pixelzeile und deren Code.

Dadurch, daß die einzelnen Bits durch das Programm richtig zusammengefügt werden, erhalten sie die Information, die Ihnen aufzeigt, mit welchem PEN der Punkt gesetzt wurde. Wenn Sie nämlich den Code eines einzelnen Punktes in dessen Dezimalwert umrechnen, dann erhalten Sie die Zahl für die PEN-Nummer!

Alles was Sie also zu tun haben ist, den binären Wert in dessen dezimales Äquivalent umzurechnen.

Nun können Sie sich davon überzeugen, daß die Codierung nur von der PEN-Nummer abhängt. Unabhängig davon, welche INKs Sie festgelegt haben, die PEN-Nummer weist dem CPC den Code zu. Aufgrund dieser Angaben, sind Sie nun wahrscheinlich auch in der Lage das Programm so umzuschreiben, daß Sie auch die Codierung und Decodierung in MODE 1 durchführen können.

Falls nicht, dann können Sie auch nachfolgendes Programm abtippen und einsetzen.

### 6.7.3 Listing: MODE1COD

```
100 'mode1cod - Decodierung der Pixel im MODE 1
110 '
120 MODE 1:p$="Pixel element":DIM f$(17):z0$=CHR$(88):z1$=CHR$(143)
130 PRINT"Bitte Zeichen eingeben: ";
140 x$=INKEY$:IF x$="" THEN 140 ELSE PRINT x$
150 INPUT "Pen";p:IF p=0 THEN p = 1
160 MODE 1:PEN p:PRINT x$
170 'POKE &C000,192:POKE &C001,96
180 '
190 FOR i = 1 TO 500:NEXT
200 '
```

```

210 'f$(n) und f$(n+1) = Wert in den zusammengehörigen Bytes
220 'PZ=Pixelzeile
230 '-----
240 f$(1)=BIN$(PEEK(&C000),8):f$(2)=BIN$(PEEK(&C001),8) 'PZ 1
250 f$(3)=BIN$(PEEK(&C800),8):f$(4)=BIN$(PEEK(&C801),8) 'PZ 2
260 f$(5)=BIN$(PEEK(&D000),8):f$(6)=BIN$(PEEK(&D001),8) 'PZ 3
270 f$(7)=BIN$(PEEK(&D800),8):f$(8)=BIN$(PEEK(&D801),8) 'PZ 4
280 f$(9)=BIN$(PEEK(&E000),8):f$(10)=BIN$(PEEK(&E001),8) 'PZ 5
290 f$(11)=BIN$(PEEK(&E800),8):f$(12)=BIN$(PEEK(&E801),8) 'PZ 6
300 f$(13)=BIN$(PEEK(&F000),8):f$(14)=BIN$(PEEK(&F001),8) 'PZ 7
310 f$(15)=BIN$(PEEK(&F800),8):f$(16)=BIN$(PEEK(&F801),8) 'PZ 8
320 '
330 LOCATE 1,1:PRINT"Ich berechne die Codes":y=1:x=1
340 '
350 'pn$(y)=Code fuer das jeweilige Pixel
360 '-----
370 p1$(y)=MID$(f$(x),5,1)+MID$(f$(x),1,1)
380 p2$(y)=MID$(f$(x),6,1)+MID$(f$(x),2,1)
390 p3$(y)=MID$(f$(x),7,1)+MID$(f$(x),3,1)
400 p4$(y)=MID$(f$(x),8,1)+MID$(f$(x),4,1)
410 p5$(y)=MID$(f$(x+1),5,1)+MID$(f$(x+1),1,1)
420 p6$(y)=MID$(f$(x+1),6,1)+MID$(f$(x+1),2,1)
430 p7$(y)=MID$(f$(x+1),7,1)+MID$(f$(x+1),3,1)
440 p8$(y)=MID$(f$(x+1),8,1)+MID$(f$(x+1),4,1)
450 IF y<8 THEN y=y+1:x=x+2:GOTO 370
460 '
470 MODE 2:PEN 1:PRINT"Die Speicherinhalte:"
480 '
490 LOCATE 60,1:PRINT"Das Zeichen war: "x$:PRINT
500 '
510 PRINT"&c000      &c001      &c800      &c801"
520 PRINT f$(1),f$(2),f$(3),f$(4):PRINT
530 PRINT"&d000      &d001      &d800      &d801"
540 PRINT f$(5),f$(6),f$(7),f$(8):PRINT
550 PRINT"&e000      &e001      &e800      &e801"
560 PRINT f$(9),f$(10),f$(11),f$(12):PRINT
570 PRINT"&f000      &f001      &f800      &f801"
580 PRINT f$(13),f$(14),f$(15),f$(16):PRINT
590 '
600 PRINT"PZ-Nr.: Die Pen-Codes der Pixel:"
610 FOR y = 1 TO 8:PRINT y;TAB(10);p1$(y)" "p2$(y)" "p3$(y)" "p4$(y);
620 PRINT " "p5$(y)" "p6$(y)" "p7$(y)" "p8$(y)TAB(65);
630 '
640 IF p1$(y)<>"00" THEN PRINT z1$; ELSE PRINT z0$;
650 IF p2$(y)<>"00" THEN PRINT z1$; ELSE PRINT z0$;
660 IF p3$(y)<>"00" THEN PRINT z1$; ELSE PRINT z0$;
670 IF p4$(y)<>"00" THEN PRINT z1$; ELSE PRINT z0$;
680 IF p5$(y)<>"00" THEN PRINT z1$; ELSE PRINT z0$;
690 IF p6$(y)<>"00" THEN PRINT z1$; ELSE PRINT z0$;
700 IF p7$(y)<>"00" THEN PRINT z1$; ELSE PRINT z0$;
710 IF p8$(y)<>"00" THEN PRINT z1$; ELSE PRINT z0$;
720 PRINT:NEXT y

```

Dieses Programm haben wir sehr "umständlich" geschrieben, damit Sie sich bei Analysen leichter tun. Aktivieren Sie bei einem der Programmläufe auch einmal die Zeile 170!



Wenn Sie all das, was mit diesem Programm zu sehen ist, und auch das Programm selbst verstanden haben, dann sind Sie wahrscheinlich auch schon in der Lage, ein Programm zu schreiben, mit dem Sie eigene Zeichen erzeugen und in die entsprechenden Codes umsetzen lassen können.

Da aber Programmentwicklungen wesentlich länger dauern, als einfach ein fertiges abzutippen, folgt nachher unser Vorschlag für einen sehr komfortablen Zeicheneditor im MODE 1.

Im Gegensatz zu einem Editor, der nur für die Erzeugung von Symbolwerten für einen eigenen Zeichensatz dient und im nächsten Abschnitt folgt, können Sie mit dem nachfolgenden auch die Farben bestimmen.

Die Speicherinhalte:

Das Zeichen war: f

&c000	&c001	&c800	&c801
00010000	11000000	00110000	01100000
&d000	&d001	&d800	&d801
00110000	00000000	01110000	10000000
&e000	&e001	&e800	&e801
00110000	00000000	00110000	00000000
&f000	&f001	&f800	&f801
01110000	10000000	00000000	00000000

PZ-Nr.: Die Pen-Codes der Pixel:

1	00	00	00	01	01	01	00	00
2	00	00	01	01	00	01	01	00
3	00	00	01	01	00	00	00	00
4	00	01	01	01	01	00	00	00
5	00	00	01	01	00	00	00	00
6	00	00	01	01	00	00	00	00
7	00	01	01	01	01	00	00	00
8	00	00	00	00	00	00	00	00

```

XXX  XX
XX  X
XX  XXXX
XX  XXXX
XX  XXXX
XX  XXXX
XX  XXXX
XX  XXXX
XXXXXXX

```

Abbildung 11: Die Ausgabe der Werte erfolgt im MODE 2.  
Das komplette Zeichen wird decodiert!

## 6.8 Sprites auf den CPCs

Da alle möglichen Bildpunkte, die der MODE 1 zulässt, mit jedem möglichen PEN gesetzt werden können, haben Sie mit diesem Programm eigentlich einen sogenannten Sprite-Editor. Als Sprites bezeichnet man im Normalfall Multicolor-"Bildchen", die vor allem bei Spielen als bewegte Figuren eingesetzt werden. Es handelt sich also um etwas ähnliches, wie die sogenannten benutzerdefinierten Zeichen. Manche anderen Computer unterstützen die Erzeugung und Bewegung von derartigen Objekten und stellen auch die entsprechenden Routinen hierfür zur Verfügung.

Bei den CPCs sind leider keine Routinen für Sprites enthalten. Deshalb muß man sich also mit zusätzlichen Programmen behelfen.

Das nachfolgend abgedruckte Programm haben wir SPRITE genannt. Dies als Hinweis, daß es ein kleiner Sprite-Editor ist. Mit diesem Sprite-Generator können allerdings nur sehr kleine Objekte in der Größenordnung von 8 auf 8 Pixeln erstellt werden und reicht deshalb nicht direkt für die Erzeugung von Spielfiguren aus. Erweitert man aber dieses Programm, dann ist es auch möglich größere Bilder zu erzeugen.

### 6.8.1 Listing: SPRITE

```

100 'Sprite-Editor
110 MEMORY &9FFF:DIM p*(8,2)
120 '
130 'off=offset fuer untereinanderstehende Pixelzeilen
140 off=&800:pz1= &C000:pz2=&C001
150 '
160 'Pixelzeilenanfaenge fuer beide Inhaltsbytes berechnen
170 FOR i = 0 TO 7
180 pz(i+1,1)=pz1+i*off:pz(i+1,2)=pz2+i*off
190 NEXT i
200 '
210 'Maschinencodebytes lesen und poken
220 FOR i = 1 TO 8:READ a:IF a<>0 THEN POKE HIMEM+i,a
230 NEXT
240 '
250 'call &bb60:      ld (&a007),a:ret
260 DATA &cd,&60,&bb,&32,&07,&a0,&c9,&00
270 '
280 'Farben und Fenster auswahlen
290 ' 0 = schwarz  26 = leuchtendweiss  21 = limonengruen  3 = rot
300 MODE 1:INK 0,0:INK 1,26:INK 2,21:INK 3,6:'WINDOW #1,25,40,2,23
310 '
320 'Maske aufbauen (Raster und Text ausgeben)
330 LOCATE 7,2:PRINT"Zeichen-Editor"
340 LOCATE 22,2:PRINT"1.Byte  2.Byte"
350 LOCATE 1,21:PRINT "Steuerung mit Cursortasten"
360 PRINT "Setzen  = Ziffern 1-3 (PEN-Nummer)"
370 PRINT "Loeschen = l      Ende      =e"
380 PRINT "Bewegen:  v=vorwaerts  r=rueckwaerts"
390 a=102:' Basis 102 von unten (Offset vertikal)
400 b= 56:' Basis von links (Offset horizontal)
410 x=312:' Breite der Linie
420 MOVE b,a:DRAW x,a:' horizontal
430 a=a+32:' 32 = Hoehe des Kaestchens
440 IF a<359 THEN 420:' maximal 9 waagerechte linien
450 ' alle Horizontallinien sind ausgegeben
460 '-----
470 y=102
480 MOVE b,356:' 356 = Maximale Hoehe der Vertikallinien
490 DRAW b,y
500 b=b+16*2:' Breite der Kaestchen
510 IF b<313 THEN 480:' 191 acht Kaestchen
520 '-----Raster ausgegeben-----
530 '
540 'Positionen und Zeichen festlegen
550 spl=5:hp=spl:zo=4:vp=zo:spr=20:zu=19:POKE &A007,32

```

```

560 y1$=CHR$(32):y2$=CHR$(143):y3$=CHR$(32)
570 '
580 '=====Hauptprogramm=====
590 'y1$, y2$, y3$ = Zeichen an der jeweiligen Position im Rasterfeld
600 'hp = Horizontalposition  vp = Vertikalposition
610 'x =Horizontalposition im Rasterfeld  y =Vertikalposition im Rasterfeld
620 y3$=CHR$(PEEK(&A007)):x=(hp-3)/2:y=(vp-2)/2
630 '
640 'Werte der Pixelzeile als Binaerwert
650 pw1$=BIN$(PEEK(pz(y,1)),8):pw2$=BIN$(PEEK(pz(y,2)),8)
660 '
670 'Binaere Darstellung in Werte umrechnen
680 pw1=VAL("&x"+pw1$):pw2=VAL("&x"+pw2$)
690 '
700 'Werte in zustaendige Bytes poken
710 IF x<5 THEN POKE pz(y,1),pw1
720 IF x>4 THEN POKE pz(y,2),pw2
730 '
740 LOCATE 22,vp:PRINT pw1$ "pw2$
750 '
760 'Cursor positionieren und Zeichen ausgeben
770 LOCATE hp,vp:PRINT y3$
780 '
790 'Eingabe
800 in$=LOWER$(INKEY$)
810 IF in$="" THEN LOCATE hp,vp:PRINT y2$:LOCATE hp,vp:PRINT y1$:GOTO 770
820 '
830 IF in$="e" THEN 1050
840 IF in$="i" THEN y3$=CHR$(32):ein$="00":GOSUB 1000:GOTO 680
850 IF in$="1" THEN y3$="1":ein$="10":GOSUB 1000:GOTO 680
860 IF in$="2" THEN y3$="2":ein$="01":GOSUB 1000:GOTO 680
870 IF in$="3" THEN y3$="3":ein$="11":GOSUB 1000:GOTO 680
880 '
890 IF in$="v" THEN GOSUB 1180
900 IF in$="r" THEN GOSUB 1250
910 x=ASC(in$):GOTO 920
920 '-----
930 'Abfrage Cursor-Steuertasten
940 IF x=241 THEN vp=vp+2:IF vp>zu THEN x=240
950 IF x=240 THEN vp=vp-2:IF vp<z0 THEN vp=vp+2
960 IF x=242 THEN hp=hp-2:IF hp<sp1 THEN hp=hp+2
970 IF x=243 THEN hp=hp+2:IF hp>spr THEN hp=hp-2
980 LOCATE hp,vp:CALL &A000:GOTO 560
990 '-----
1000 'Binaere Werte in Poke-Werte einsetzen
1010 IF x<5 THEN MID$(pw1$,x,1)=LEFT$(ein$,1):MID$(pw1$,x+4,1)=RIGHT$(ein$,1)
1020 IF x>4 THEN MID$(pw2$,x-4,1)=LEFT$(ein$,1):MID$(pw2$,x,1)=RIGHT$(ein$,1)
1030 RETURN
1040 '
1050 FOR i = 1 TO 8
1060 piw$(i,1)= HEX$(PEEK(pz(i,1)),2):piw$(i,2)=HEX$(PEEK(pz(i,2)),2)
1070 NEXT
1080 '
1090 MODE 1
1100 PRINT"Die Poke-Werte fuer MODE 1 (hexadezimal)":PRINT
1110 PRINT"PZ", "Byte 1      Byte 2
1120 PRINT STRING$(40,"-")
130

```

```

1130 FOR i = 1 TO 8:PRINT i,piw$(i,1),piw$(i,2):NEXT i
1140 PRINT STRING$(40,"-")
1150 END
1160 '
1170 'Sprite nach rechts
1180 IF pz(1,1)>&C03F THEN RETURN
1190 FOR i = 1 TO 8: pz(i,1)=pz(i,1)+2:pz(i,2)=pz(i,2)+2
1200 POKE pz(i,1),PEEK(pz(i,1)-2):POKE pz(i,1)+2,0
1210 POKE pz(i,2),PEEK(pz(i,2)-2):POKE pz(i,2)+2,0
1220 NEXT i:RETURN
1230 '
1240 'Sprite nach links
1250 IF pz(1,1)<&C001 THEN RETURN
1260 FOR i = 1 TO 8: pz(i,1)=pz(i,1)-2:pz(i,2)=pz(i,2)-2
1270 'Sprite nach links
1280 POKE pz(i,1),PEEK(pz(i,1)+2):POKE pz(i,1)+2,0
1290 POKE pz(i,2),PEEK(pz(i,2)+2):POKE pz(i,2)+2,0
1300 NEXT i
1310 RETURN

```

### 6.8.2 Bedienung des Programmes SPRITE.

Nach dem Programmstart sehen Sie neben den allgemeinen Menü-Informationen ein Rasterfeld mit 8 mal 8 Kästchen. Im linken oberen Kästchen sehen Sie den blinkenden Cursor, der mittels der Cursor-steuertasten innerhalb dieses Feldes bewegt werden kann.

Da dieser Editor für den MODE 1 gedacht ist, sind zur Verschlüsselung der 64 Pixel zwei mal 8, also 16 Bytes nötig. Die binären Werte hierfür sehen Sie rechts unter den Bezeichnungen 1. und 2. Byte.

Da noch keine Eingaben erfolgten, ist nur die erste Pixelzeile mit nicht gesetzten Bits zu sehen.

Wenn Sie den Cursor nach unten bewegen, erfolgt auch die Ausgabe der weiteren Bytes.

Wollen Sie nun einen Bildpunkt setzen, dann brauchen Sie nur eine der Zifferntasten (1 bis 3) für einen der drei möglichen PENS zu wählen. Zur Erinnerung, der MODE 1 läßt vier Farben zu.

Eine Farbe wird dabei für den Hintergrund verwandt, so daß Sie drei PEN-Farben zur Verfügung haben. Damit Sie eine bessere Vorstellung vom jeweiligen Zeichenstand haben, wird das von Ihnen erzeugte Zeichen mit dem aktuellen Stand links oben eingeblendet. Farbmonitorbesitzer sehen deshalb auch den Stand der Arbeiten in Farbe. Auf einem Grünmonitor erscheint dieses Zeichen je nach gewählten INKS mit unterschiedlichen Helligkeiten.

In diesem Zusammenhang eine Information, die zur Zeichenerzeugung wichtig sein kann. Zwar kann ein Farbmonitorbesitzer vermuten, daß die Farbnummern von 0 bis 26 völlig ohne tieferen Sinn gewählt wurden, die Wahrheit ist aber, daß diese in der Reihenfolge der Helligkeit (Grauabstufungen auf einem monochromen Bildschirm) geordnet sind. Schwarz (Farbnummer 0) ist dabei der dunkelste und

Leuchtendweiß (26) der hellste Grauwert.

Rechts sehen Sie auch dauernd den aktuellen Stand der Codes.

Sind Sie mit Ihrer Arbeit zufrieden, dann können Sie durch Eingabe des Buchstabens "e" das Programm beenden, wodurch Ihnen dann die Codes für die 16 Bytes als Hexwerte ausgegeben werden. Diese Werte können Sie dann zur Erzeugung Ihres Zeichens in einem Programm verwenden.

Als kleine Anregung haben wir auch noch eine Möglichkeit, Ihr Sprite auf dem Bildschirm zu bewegen, vorgesehen. Die horizontale Bewegung erfolgt durch Betätigung der Tasten "v" (vorwärts) und "r" (rückwärts), andere Möglichkeiten haben wir nicht vorgesehen, da Basic mit dem Bildaufbau zu langsam ist. Außerdem erfolgt die Bewegung immer nur in Sprüngen von 8 Bildpunkten, denn sonst wäre das Programm zu umfangreich geworden.



Zeichen-Editor								1.Byte	2.Byte
1	1	1	1	1	1	1	1	11110000	11110000
			2		1			00000001	01000000
			2		2		3	00000001	00010101
3	3		2		1		3	11001101	01010001
3	3		2		2		3	11001101	00010101
			2		1			00000001	01000000
3	3	3	3	3	3	3	3	11111111	11111111
3	3	3	3	3	3	3	3	11111111	11111111

**Steuerung mit Cursortasten**  
**Setzen** = Ziffern 1-3 (PEN-Nummer)  
**Loeschen** = 1 Ende = e  
**Bewegen:** v=vorwaerts r=rueckwaerts

Abbildung 12: Der Zeichen-Editor "SPRITE" in Aktion.

Wie Sie aber sehen können, haben Sie mit diesem Programm wirklich einen kleinen Sprite-Editor und können Ihr Sprite sogar noch nach links und rechts bewegen.

Wollen Sie einen Bildpunkt verändern, dann brauchen Sie nur mit einer anderen PEN-Nummer überschreiben. Wollen Sie dagegen einen gesetzten Punkt löschen (also mit der Farbe des Hintergrundes schreiben), dann betätigen Sie bitte die Taste "1", also den ersten Buchstaben des Wortes löschen.

Wenn Sie dieses Programmlisting einmal mit dem Programm MODE1COD vergleichen, dann werden Sie feststellen können, daß in diesem Editor-Programm die Bildpunktcodierung und -decodierung sehr

stark komprimiert und auch nicht mehr so umständlich programmiert ist. Trotzdem aber hoffen wir, daß Ihnen die Kommentare im Programm-Listing ausreichen, auch dieses Programm zu verstehen. Die kleine Maschinenroutine dient nur dazu, die Zeichen im Rasterfeld zu erkennen. Der Aufruf von &BB60 bewirkt, daß an der Cursorposition ein Zeichen gelesen wird. Dieses Zeichen wird dann in die Speicherstelle &A007 geschrieben, und vom Basicteil ausgelesen und decodiert, also in seinen ASCII-Wert umgesetzt. Die benutzte Firmwareroutine kann zwar ASCII-Zeichen erkennen und wird deshalb auch bei einem später folgenden Programm zum Umkopieren von derartigen Zeichen eingesetzt, aber leider kann das Sprite nicht "behandelt" werden, sonst hätten wir eine einfachere Möglichkeit, dieses Zeichen zu bewegen.

Deshalb noch einmal zu Bewegung des Sprites. Wie Sie sehen konnten, geschieht die Bewegung durch Basic sehr langsam. Ein Maschinenprogramm kann dies viel besser und vor allem auch schneller. Wie die Grundprinzipien für ein derartiges Programm aussehen können und wie sich dies dann darstellt, können Sie mit nachfolgenden Programm erkennen.

### 6.8.3 Listing: SPDEMO.BAS

```

100 ' Spritedemo
110 '
120 MEMORY &9FFF:MODE 1
130 READ w$:IF w$="-1" THEN 160
140 POKE HIMEM+1+1,VAL("&"+w$):i=i+1:GOTO 130
150 '
160 a=&C000:b=&C001
170 READ w$:IF w$="-2" THEN 210
180 READ w1$:POKE a,VAL("&"+w$):POKE b,VAL("&"+w1$)
190 a=a+&800:b=b+&800:GOTO 170
200 '
210 POKE &A044,x:CALL &A000
220 FOR i = 1 TO 10:NEXT
230 IF x < 250 THEN x=x+2:GOTO 210
240 '
250 'Daten Maschinenprogramm
260 DATA ED,5B,44,A0,2A,44,A0,22,46,A0,EB,7E,32,4A,A0,23,7E,32,4C,A0
270 DATA 13,13,ED,53,48,A0,2A,48,A0,3A,4A,A0,77,23,3A,4C,A0,77,3E,0,2A
280 DATA 46,A0,77,23,77,2A,44,A0,7C,C6,8,FE,F9,67,22,44,A0,FA,0,A0,21
290 DATA 0,C0,22,44,A0,C9,0,C0,-1
300 '
310 'Daten Sprite
320 DATA F7,FF,D7,DF,C7,9F,D7,5F,D7,DF,D7,DF,D2,D7,DF,DF,-2

```

Die Daten für das Maschinenprogramm finden Sie in den Zeilen 250 bis 290. Die Daten für das Sprite sind in Zeile 320 zu lesen. Dort können Sie aber auch die Daten für ein von Ihnen kreiertes Zeichen ablegen. Die Geschwindigkeit der Bewegung kann in Zeile 220 variiert werden. Obwohl dieses Programm nur aufzeigen soll, wie ein selbstdefiniertes Zeichen bewegt werden kann, trotzdem noch das Assemblerlisting, dieser sehr spartanisch gehaltenen Routine.

## 6.8.4 Assemblerlisting: SPMOVE.EDI

```

;*****
;* SPMOVE.edi *
;*****
;bewegt Zeichen nach rechts

org &a000

pzstart
ld de,(startpz)      ;alte Adresse
ld hl,(startpz)      ;alte Adresse
ld (merkeralt),hl    ;merken
ex hl,de             ;alte Adresse

ld a,(hl)            ;Zeichen 1 holen
ld (zeichen1),a      ;und merken
inc hl              ;Adresse erhoehen
ld a,(hl)            ;Zeichen 2 holen
ld (zeichen2),a      ;und merken
inc de:inc de        ;neue adresse
ld (merkerneu),de    ;merken

ld hl,(merkerneu)    ;hl = neue adresse
ld a,(zeichen1)      ;erstes Zeichen holen
ld (hl),a            ;Zeichen1 schreiben
inc hl              ;Adresse erhoehen
ld a,(zeichen2)      ;Zeichen 2 holen
ld (hl),a            ;und schreiben

ld a,0
ld hl,(merkeralt)    ;alte Zeichenpositionen
ld (hl),a            ;loeschen
inc hl              ;Adresse erhoehen
ld (hl),a            ;loeschen
ld hl,(startpz)      ;alte Adresse holen
ld a,h:add a,&08      ;High Byte zwischenspeichern
cp a,&f9             ;Vergleichen ob Ende erreicht
ld h,a              ;H erhoehen (naechste PZ)
ld (startpz),hl      ;merken
jp m,pzstart
ld hl,&c000
ld (startpz),hl
ret

startpz defw &c000 : merkeralt defw &0000 : merkerneu defw &0000
zeichen1 defw &00 : zeichen2 defw &00

```

Zum Schluß über die Codierung der einzelnen Pixel nun noch eine Aufstellung, die Ihnen helfen soll, die Verschlüsselung besser zu verstehen, um Ihnen falls Sie dies vorhaben, bei der Erstellung eines Editors für den MODE 0 Anregungen zu geben. Alle Informationen, die hierzu erforderlich sind haben wir Ihnen gegeben.

Wenn Sie sich eine derartige Aufstellung für den MODE 0 erstellen, kann eigentlich nichts mehr schief gehen.

## 6.8.5 Übersicht: Codierung einzelner Pixel im MODE 1

Byte 1		Byte 2		Farbstift	Wert	Pixel-Stelle
1		1				
2631		2631				
84268421		84268421				
&cxxg 10000000	&cxxu 00000000	PEN 1	128			
&cxxg 00001000	&cxxu 00000000	PEN 2	8			1. Pixel gesetzt
&cxxg 10001000	&cxxu 00000000	PEN 3	136			
&cxxg 01000000	&cxxu 00000000	PEN 1	64			
&cxxg 00000100	&cxxu 00000000	PEN 2	4			2. Pixel gesetzt
&cxxg 01000100	&cxxu 00000000	PEN 3	68			
&cxxg 00100000	&cxxu 00000000	PEN 1	32			
&cxxg 00000010	&cxxu 00000000	PEN 2	2			3. Pixel gesetzt
&cxxg 00100010	&cxxu 00000000	PEN 3	34			
&cxxg 00010000	&cxxu 00000000	PEN 1	16			
&cxxg 00000001	&cxxu 00000000	PEN 2	1			4. Pixel gesetzt
&cxxg 00010001	&cxxu 00000000	PEN 3	17			
&cxxg 00000000	&cxxu 10000000	PEN 1	128			
&cxxg 00000000	&cxxu 00001000	PEN 2	8			5. Pixel gesetzt
&cxxg 00000000	&cxxu 10001000	PEN 3	136			

usw.

## Erläuterung:

Die Bezeichnung &cxxg weist auf Bytes mit geraden Endziffern, &cxxu auf solche mit ungeraden hin. Über der binären Darstellung sehen Sie den Wert für die jeweilige Bitstelle. Die Codierung selbst können Sie für die verschiedenen PENS an den ebenfalls verschiedenen Stellen in den entsprechenden Bytes erkennen.

Was Sie außerdem leicht erkennen können, ist die Tatsache, daß die ersten vier Pixel im 1. Byte und die Pixel 5 bis 8 im zweiten Byte gespeichert werden.

Das heißt, daß für die Pixel 5 bis 8 die gleiche Verschlüsselung wie bei den Pixeln 1 bis 4 gilt. Nur wird dann nicht im ersten, sondern im zweiten Byte abgespeichert.

## 6.9 Umkopieren von Bildschirmzeilen

Nun am Ende dieses Abschnittes - wie versprochen - ein Programm zum Umkopieren von ASCII-Zeichen für alle CPCs.



## 6.9.1 Assembler-Listing: LINECOPY.EDI

```

write "linecopy"
;*****
;*          Linecopy.edi          *
;*-----*
;* dieses Programm "kopiert"      *
;* Schirmzeilen von Quelle nach   *
;* Ziel um. Die Anzahl der Zeichen *
;* wird in's Register B ueber-    *
;* nommen um 'DJNZ' einsetzen zu  *
;* koennen.                      *
;*****

setcurs equ &bb75:readchar equ &bb60:writechar equ &bb5a

org &a000      ;Startadresse

ld bc,(anzahl) ;Zeichenanzahl holen
push bc       ;und bis Schluss merken

ld hl,(quelle) ;Startposition holen
push hl       ;und ebenfalls merken
ld hl,(quelle+2);Zielposition holen
push hl       ;und auch merken

weiter
ld hl,(quelle) ;Startposition
call check    ;pruefen ob rechter Rand erreicht
inc h        ;zum speichern erhoehen
ld (quelle),hl ;speichern
dec h        ;wieder korrigieren
call curset   ;Cursor setzen
push bc      ;Zaehler (b) retten
call readchar ;Zeichen lesen
pop bc       ;Zaehler wieder holen
ld c,a       ;gelesenes Zeichen retten

ld hl,(quelle+2);Zielposition holen
call check    ;pruefen ob rechter Rand erreicht
inc h        ;zum speichern erhoehen
ld (quelle+2),hl ;und speichern
dec h        ;korrigieren
call curset   ;Cursor an Ziel positionieren
ld a,c       ;gelesenes Zeichen wieder holen
call writechar ;und ausgeben
djnz weiter   ;Falls Zaehler >0 weitermachen

pop hl        ;Urspruenglichen Wert
ld (quelle+2),hl ;wieder abspeichern (Ziel)

pop hl        ;Urspruenglichen Wert
ld (quelle),hl ;wieder rueckschreiben (Quelle)

pop bc        ;Zaehler wieder
ld (anzahl),bc ;zurueckschreiben
ret           ;ab nach BASIC

```

```

curset  push hi      ;Position retten da durch Positionierung
        call setcurs ;hi veraendert wird
        pop hi
ret

check   ld a,h        ;hat h
        cp 80         ;rechten Rand erreicht ?
        ret c         ;nein
        inc i         ;ja,eine Zeile tiefer
        ld h,01       ;und in erste Spalte
ret

quelle defb &01,&01:zie|  defb &0c,&05:anzahl defb &00,255

```

Nun folgt, wie in diesem Buche Ublich, wieder der entsprechende Hexlader.

### 6.9.2 Listing:LINECOPY.HEX

```

870 REM linecopy.hex
880 a= 40960:e= 41047:zb=2000
890 FOR i =a TO e:READ d$:IF LEFT$(d$,1)<>"&" THEN 920
900 IF ps<>VAL(d$) THEN PRINT"Fehler Zeile "zb:END
910 ps=0:d$="":IF i=e THEN 3000:ELSE i=i-1:zb=zb+1:GOTO 950
920 d$="&"+d$
930 POKE i,VAL(d$):ps=ps+VAL(d$)
940 PRINT VAL(d$)
950 IF i < e THEN NEXT i
2000 DATA ED,4B,55,A0,C5,2A,51,A0,E5,2A,53,A0,E5,2A,51,A0,&080F
2001 DATA CD,49,A0,24,22,51,A0,25,CD,43,A0,C5,CD,60,BB,C1,&0830
2002 DATA 4F,2A,53,A0,CD,49,A0,24,22,53,A0,25,CD,43,A0,79,&06A9
2003 DATA CD,5A,BB,10,D8,E1,22,53,A0,E1,22,51,A0,C1,ED,43,&08A5
2004 DATA 55,A0,C9,E5,CD,75,BB,E1,C9,7C,FE,50,DB,2C,26,01,&093F
2005 DATA C9,01,01,0C,05,00,FF,&01DB
3000 SAVE"linecopy.bin",b,40960,87

```

Damit Sie nun die Arbeitsweise näher untersuchen können und auch diese Routine für eigene Programme in den Griff bekommen, hier auch noch ein entsprechendes Demonstrationsprogramm.

### 6.9.3 Listing:LINECOPY.BAS

```

100 MEMORY &9FFF:MODE 2:PRINT"Demoprogramm fuer Maschinenspracheroutine
110 PRINT"LINECOPY":PRINT:PRINT"Bitte eine Taste druecken!"
120 '
130 modus =2:CALL &BB10:zeilend =80:CLS
140 REM Quelle =&a051+&a052 (Zeile + Spalte)
150 REM Ziel  =&a053+&a054 (Zeile + Spalte)
160 REM Anzahl =&a056
170 '
180 IF PEEK(&A000)<>&ED THEN LOAD"linecopy.bin",&A000
190 '
200 MODE modus:POKE &A04B,zeilend
210 '
220 PRINT"Diese Zeichen werden umkopiert! 1234567890";

```

```
230 PRINT"abcdefghijklmnopqrstuvwyz";
240 PRINT"12345678901234567890123456789012345678901234567890";
250 PRINT"12345678901234567890123456789012345678901234567890";
260 PRINT"12345678901234567890123456789012345678901234567890";
270 '
280 CALL &A000:modus =modus - 1:zeilend=PEEK(&A04B)/2:PRINT
290 IF modus <0 THEN POKE &A04B,00:MODE 2:END
300 PRINT"Bitte eine Taste druecken":CALL &BB10
310 GOTO 200
```

Im nächsten Abschnitt wenden wir uns nun den Farben selbst zu.

### 6.10 Die Hardware-Nummern der Farben

Wie Sie den vorherigen Abschnitten entnehmen konnten, ist die Codierung der einzelnen Pixel bzw. deren Farben vom gewählten PEN abhängig. Die PEN-Nummer ist also die Zahl, die angibt in welches "Tintenfaß" eingetaucht wird. Da der CPC nur 16 Farben gleichzeitig zuläßt, insgesamt aber 26 Farben zur Verfügung stehen, hat er auch Speicherstellen, in denen er sich die Farben (INKs) merkt.

Wichtig ist in diesem Zusammenhang aber zu wissen, daß für die Farben zwei Farbtabellen im RAM abgelegt sind.

Der Grund für zwei Farbtabellen ist in der Blinkmöglichkeit für die Farben zu sehen, denn eigentlich geschieht im CPC folgendes: Alle Farben blinken ununterbrochen in den beiden - einander zugehörigen - ausgewählten Farbwerten. Sind zwei zugehörige Werte aber gleich, dann bemerkt der Benutzer es gar nicht, daß laufend von dem einen auf den anderen Farbsatz umgeschaltet wird.

Wie viele andere System-Speicherstellen liegen auch diese Tabellen bei den verschiedenen CPC-Typen unterschiedlich im RAM.

Hier deshalb wieder die Basisadressen:

CPC 464	&B1D9
CPC 664/6128	&B7D4

Die Reihenfolge ist dabei folgende:

```
BORDER Farbsatz 1
PAPER (INK 0) Farbsatz 1
INK 1 bis 15 Farbsatz 1
```

```
BORDER Farbsatz 2
PAPER (INK 0) Farbsatz 2
INK 1 bis 15 Farbsatz 2
```

Allerdings können die Farben nun nicht einfach durch Poken des entsprechenden "Basic-Farbwertes" korrekt beeinflußt werden. Vielmehr darf nur die Hardware-Nummer der Farbe in die jeweilige Speicherstelle geschrieben werden.

Der CPC legt nämlich beim Befehl BORDER 0 nicht die "Null" in die entsprechenden Speicherstellen ab, sondern die Hardware-Nummer der Farbe. Damit Sie aber auch direkt Farben poken können, folgen anschließend zwei Tabellen. Die erste Tabelle ist nach den Normalwerten sortiert, also nach den Werten, die Sie mit dem entsprechenden Farbbefehl "mitgeben". Die zweite Tabelle ist nach den Hardware-Nummern gegliedert. Außerdem ist in dieser Aufstellung auch noch der Binärwert angegeben.

## 6.10.1 Farbwerte sortiert nach Normalwerten

Farb-Nr.	Hardw-Nr.	Farbe
00	20	Schwarz
01	04	Blau
02	21	Hellblau
03	28	Rot
04	24	Magenta
05	29	Hellviolett
06	12	Hellrot
07	05	Purpur
08	13	helles Magenta
09	22	Gruen
10	06	Blaugruen
11	23	Himmelblau
12	30	Gelb
13	00	Weiss
14	31	Pastellblau
15	14	Orange
16	07	Rosa
17	15	Pastell-Magenta
18	18	Hellgruen
19	02	Seegruen
20	19	helles Blaugruen
21	26	Limonengruen
22	25	Pastellgruen
23	27	Pastell-Blaugruen
24	10	Hellgelb
25	03	Pastellgelb
26	11	Leuchtendweiss

## 6.10.2 Farbwerte sortiert nach Pokewerten (Hardware-Werten)

Hardw.-Nr.	Farb-Nr.	Farbe	Binaerwert
00	13	Weiss	00000
02	19	Seegruen	00010
03	25	Pastellgelb	00011
04	01	Blau	00100
05	07	Purpur	00101
06	10	Blaugruen	00110
07	16	Rosa	00111
10	24	Hellgelb	01010
11	26	Leuchtendweiss	01011
12	06	Hellrot	01100
13	08	helles Magenta	01101
14	15	Orange	01110
15	17	Pastell-Magenta	01111
18	18	Hellgruen	10010
19	20	helles Blaugruen	10011
20	00	Schwarz	10100
21	02	Hellblau	10101
22	09	Gruen	10110
23	11	Himmelblau	10111
24	04	Magenta	11000

25	22	Pastellgruen	11001
26	21	Limonengruen	11010
27	23	Pastell-Blaugruen	11011
28	03	Rot	11100
29	05	Hellviolett	11101
30	12	Gelb	11110
31	14	Pastellblau	11111

### 6.11 Ein eigener Zeichensatz

Oft besteht der Wunsch, anstelle des im CPC vorhandenen Zeichensatzes mit einem eigenen zu arbeiten. Dies trifft vor allem dann zu, wenn man Text verarbeiten will.

Für viele ist die Umrechnung in die entsprechenden Symbol-Werte sehr mühselig und deswegen wird darauf verzichtet. Dies muß aber nicht sein, denn unser Zeicheneditor erlaubt es, sehr komfortabel eigene Zeichen zu definieren und gibt Ihnen auch die Symbol-Werte aus.

Alles was wir zur Erstellung eines derartigen Programmes benötigen, wurde in den vorangegangenen Abschnitten erarbeitet. Da eigene Zeichen vor allem zur Textverarbeitung erforderlich sind und diese meist im 80-Zeichen-MODE durchgeführt wird, haben wir diesen Editor so programmiert, daß Ihnen nach der Zeichenerstellung das von Ihnen eingegebene Zeichen am oberen linken Bildrand dargestellt wird. Allerdings geschieht dies nur im 80-Zeichenmode. Im MODE 1 sind die Pixelzeilen ja über 2 Bytes verteilt und deswegen erfolgt bei 40-Zeichen-Darstellung keine Ausgabe des erarbeiteten Zeichens.

Trotzdem haben wir das Programm so geschrieben, daß die Editierung auch in diesem Mode erfolgen kann.

Wenn Sie sich diesen Editor näher betrachten, können Sie feststellen, daß er nur eine Variante des Sprite-Editors ist. Da aber zur Umdefinition von "Schreibzeichen" keine Farben erforderlich und auch nicht möglich sind, entfällt dieser Teil in diesem Listing.

Wie Sie sehen, kann mit den Grundprinzipien unserer Programme eine ganze Menge angefangen werden. Deshalb ist es durchaus möglich, z. B. aus diesen Grundprogrammen einen Zeicheneditor für den MODE 0 zu erstellen. Falls Sie bisher noch nicht die zündende Idee hatten, wie denn die Farbauswahl im Rasterfeld dargestellt werden kann, hier noch eine kleine Anregung.

Hierzu sollten Sie zur Farbauswahl am besten die Eingabe und auch die Ausgabe im Rasterfeld als Hexadezimalwert nehmen, denn dann sind diese Werte ohne Probleme im Rasterfeld unterzubringen. Auf die Anzeige der vier Bytes müssen Sie aus Platzgründen allerdings verzichten. Es gibt aber auch die Möglichkeit, das Zeichen in MODE 1 oder 2 zu entwickeln um dann wenn gewünscht, auf die 20-Zeichen-Darstellung umzuschalten und das Zeichen darstellen zu lassen.

Uns aber ging es bei nachfolgenden Programm vor allem um eine komfortable Möglichkeit, Zeichen neu zu erstellen um verschiedenen Tasten neue Zeichen zuzuordnen, wie dies beispielsweise für eine DIN-Tastatur erforderlich ist.

### 6.11.1 Listing: Zeicheneditor

```

100 'Zeicheneditor fuer Symbole
110 '
120 MEMORY &9FFF:CALL &BB00
130 FOR i = 1 TO 8:READ a:IF a<>0 THEN POKE HIMEM+i,a
140 NEXT i:POKE &A007,32
150 '
160 'Code der Maschinenroutine
170 ' call &bb00: ld (&a007),a:ret
180 DATA &cd,&60,&bb,&32,&07,&a0,&c9,&00
190 '
200 INPUT "MODE 2 oder 1";m
210 IF m >2 THEN 200 ELSE IF m<1 THEN 200
220 MODE m
230 IF m=2 THEN a=102:b=26:x=156:k=172 ELSE a=102:b=56:x=312:k=313
240 ' a= Basis von unten (Offset vertikal)
250 ' b= Basis von links (Offset horizontal)
260 ' x= Breite der Linie
270 '
280 INK 0,0:INK 1,26:INK 2,0:INK 3,15
290 WINDOW #1,25,40,2,23
300 '
310 MOVE b,a:DRAW x,a: horizontal
320 a=a+32:' 32 = Hoehe des Kaestchens
330 IF a<359 THEN 310:' maximal 9 waagerechte Linien
340 ' alle Horizontallinien sind ausgegeben
350 '-----
360 y=102
370 MOVE b,356:' 356 = Maximale Hoehe der Vertikallinien
380 DRAW b,y
390 b=b+(32/m):' Breite der Kaestchen
400 IF b<k THEN 370: acht Kaestchen
410 '
420 LOCATE 7,1:PRINT"Zeichen-Editor
430 PRINT#1:PRINT#1:PRINT #1,"Steuerung
440 PRINT #1:PRINT #1,"mit Cursortasten
450 PRINT#1:PRINT#1,"Setzen =COPY":PRINT#1
460 PRINT#1,"Loeschen =|":PRINT#1:PRINT#1,"Ende =e
470 FOR i = 1 TO 10:PRINT #1:NEXT i
480 '
490 '-----Raster ausgegeben-----
500 spl=5:hp=spl:zo=4:vp=zo:spr=20:zu=19
510 y1$=CHR$(32):y2$=CHR$(143)
520 FOR i = 1 TO 8:z(i)=0:r(i)="" :NEXT i:q$=""
530 LOCATE 1,23:PRINT"lhre Eingaben bitte
540 '
550 LOCATE hp,vp:CALL &A000:y3$=CHR$(PEEK(&A007))
560 LOCATE hp,vp:PRINT y3$
570 x$=INKEY$
580 IF LOWER$(x$)="e" THEN PEN 1:GOTO 740

```

```

590 IF LOWER$(x$)="1" THEN y3$=CHR$(32):GOTO 560
600 IF x$<>" " THEN x=ASC(x$):GOTO 660
610 '
620 LOCATE hp, vp:PRINT y2$
630 LOCATE hp, vp:PRINT y1$
640 GOTO 560
650 '-----
660 IF x=224 THEN y3$=y2$:GOTO 560
670 IF x=241 THEN vp=vp+2:IF vp>zu THEN x=240
680 IF x=240 THEN vp=vp-2:IF vp<zo THEN vp=vp+2
690 IF x=242 THEN hp=hp-2:IF hp<spl THEN hp=hp+2
700 IF x=243 THEN hp=hp+2:IF hp>spr THEN hp=hp-2
710 LOCATE hp, vp:CALL &A000
720 GOTO 510
730 '-----
740 CLS #1:hp=spl:vp=zo:x=0:PRINT#1, "& Binaer":PRINT#1
750 LOCATE hp, vp:CALL &A000
760 IF altvp <>vp THEN x=x+1:altvp=vp:PRINT CHR$(7)
770 IF PEEK(&A007)=143 THEN r$(x)=r$(x)+"1" ELSE r$(x)=r$(x)+"0"
780 hp=hp+2:IF hp <20 THEN 750 ELSE hp=spl
790 vp=vp+2:IF vp <20 THEN 750
800 '
810 FOR i =1 TO 8:x=128
820 FOR y = 1 TO 8:a$=MID$(r$(i), y, 1)
830 IF a$="1" THEN z(i)=z(i)+x
840 x=x/2
850 NEXT y:PRINT#1, HEX$(z(i), 2); " "; r$(i):PRINT#1
860 IF i<8 THEN q$=q$+"&" +HEX$(z(i), 2)+", " ELSE q$=q$+"&" +HEX$(z(i), 2)
870 NEXT i
880 '
890 'Zeichen poken
900 IF m= 1 THEN 940
910 POKE &C000, z(1):POKE &C800, z(2):POKE &D000, z(3):POKE &D800, z(4)
920 POKE &E000, z(5):POKE &E800, z(6):POKE &F000, z(7):POKE &F800, z(8)
930 '
940 LOCATE 1, 23:PRINT"Einverstanden (j/n) ?"
950 x$=LOWER$(INKEY$):IF x$="" THEN 950
960 IF x$="n" THEN 420
970 '
980 LOCATE 1, 23:PRINT"Soll der Code auf KEY 0 gelegt werden ?"
990 x$=LOWER$(INKEY$):IF x$="" THEN 990
1000 IF x$="j" THEN KEY 0, q$:KEY 139, "call &bb00"+CHR$(13)

```

Die Bedienung dieses Editors ist völlig unproblematisch und durch die Cursor-Steuerung auch kinderleicht. Fahren Sie durch Betätigung der Steuertasten also beliebig im Rasterfeld umeinander und drücken Sie immer dann, wenn Sie einen Punkt setzen wollen, die COPY-Taste.

Haben Sie einen Punkt versehentlich gesetzt, dann läßt sich dieser durch Betätigung der Taste "L" wieder ausradieren. Sind Sie mit dem editieren fertig, dann drücken Sie die Taste "e", daraufhin werden die hexadezimalen und binären Werte jeder Pixelzeile ausgegeben. Gleichzeitig werden Sie gefragt, ob Sie mit dem bisherigen einverstanden sind. Beantworten Sie diese



Frage mit "N" dann können Sie an Ihrem Zeichen weiterarbeiten.

Antworten Sie mit "J" dann folgt die Frage, ob Sie den gesamten Code für dieses Zeichen auf die Taste "0" des Zehnerblockes legen wollen.

Dies dient dem Zweck, daß Sie durch Voranstellen einer Zeilennummer und dem Wort DATA nach einfacher Betätigung der Zifferntaste 0 das eben erzeugte Zeichen in Datastatements übernehmen können, um dann das nächste Zeichen zu erzeugen usw.

Sind Sie mit all den Arbeiten fertig, dann sollten Sie durch gleichzeitiges Betätigen der SHIFT und der kleinen ENTER-Taste die Tastaturverwaltung wieder in den Originalzustand rückstellen. Durch löschen der Programmzeilen verbleibt Ihnen dann ein Datenfeld mit den neuen Zeichen, die Sie nun nur noch zuordnen müssen.

Wie dies geschehen kann, soll Ihnen nachfolgendes Listing zeigen, mit dem Sie auf dem Bildschirm dann eine "deutsche" Tastatur zur Verfügung haben.

#### 6.11.2 Listing: DEUTAST

```

100 'Deutast - Tastatur nach DIN
110 MODE 2
120 PRINT "Tastaturumbelegung nach DIN"
130 PRINT "ASCII (alt)                                Umdefinition
140 FOR i = 3 TO 13:READ a$:IF i = 3 THEN READ b$:a$=a$+CHR$(34)+b$
150 LOCATE 1,1:PRINT a$:NEXT
160 SYMBOL AFTER 64
170 '
180 SYMBOL 64,&3C,&60,&3C,&66,&3C,&6,&3C,&0'      Paragraph
190 SYMBOL 91,&C6,&10,&38,&6C,&C6,&FE,&C6,&0'
200 SYMBOL 92,&C6,&0,&7C,&C6,&C6,&C6,&7C,&0'      Schraegstrich
210 SYMBOL 93,&66,&0,&66,&66,&66,&66,&3C,&0'      #
220 SYMBOL 123,&66,&0,&78,&C,&7C,&CC,&78,&0'      ae
230 SYMBOL 124,&66,&0,&3C,&66,&66,&66,&3C,&0'      oe
240 SYMBOL 125,&66,&0,&66,&66,&66,&66,&3E,&0'      ue
250 SYMBOL 126,&78,&C6,&C6,&FC,&C6,&C6,&DB,&0'      scharfes s
260 '
270 'Reihenfolge: Tastennummer
280 'Dauerfunktion,Normal,Shift,CTRL      neu      alt
290 '
300 KEY DEF 17,1,43,42,27'      + * &1B      [ ( &1B
310 KEY DEF 19,1,35,39,29'      # ' &1D      ] ) &1D
320 KEY DEF 22,1,80,62,28'      < , >      \ ' &1C
330 KEY DEF 24,1,94,96,30'      ^ ` &1E      ^ # &1E
340 KEY DEF 25,1,126,63'      scharfes s ?      - =
350 KEY DEF 26,1,125,93'      ue UE      0 ; &0
360 KEY DEF 28,1,123,91'      ae AE      ; +
370 KEY DEF 29,1,124,92'      oe OE      : *
380 KEY DEF 30,1,45,95'      - _      / ?
390 KEY DEF 31,1,46,58'      . ,      . ,
400 KEY DEF 32,1,48,61,31'      0 = &1F      0 _ &1F
410 KEY DEF 39,1,44,59'      , ;      , <
420 KEY DEF 41,1,55,47'      7 /      7 `

```

```

430 KEY DEF 43,1,122,90,26'          z Z &1A          y Y &19
440 KEY DEF 56,1,52,36,163'          4 $ &A3          4 $
450 KEY DEF 57,1,51,64,&0'          3 Par. &00          3 #
460 KEY DEF 71,1,121,89,25'          y Y &19          z Z &1A
470 '
480 DATA "! ", " # $ % & ' ( ) _ = # "
490 DATA "1 2 3 4 5 6 7 8 9 0 - ^ "
500 DATA " "
510 DATA "Q W E R T Y U I O P ! { "
520 DATA "q w e r t y u i o p o [ "
530 DATA " "
540 DATA "A S D F G H J K L * + }
550 DATA "a s d f g h j k l : ; | "
560 DATA " "
570 DATA "Z X C V B N M < > ? ` "
580 DATA "z x c v b n m , . / \
590 '
600 RESTORE:FOR i = 3 TO 13:READ a$:IF i = 3 THEN READ b$:a$=a$+CHR$(34)+b$
610 LOCATE 40,1:PRINT a$:NEXT
620 PRINT:PRINT"Wie Sie sehen, wurden Tasten anders belegt !
630 PRINT"Ausserdem wurden neue Zeichen festgelegt, wie Sie sich durch
640 PRINT"Betaetigung der entsprechenden Tasten ueberzeugen koennen.
650 PRINT"Die Belegung entspricht nun DIN.
660 PRINT
670 PRINT"Leider wirkt sich beim CPC 464 die Umdefinition nur auf dem
680 PRINT"Bildschirm aus, da dieser CPC keine Druckerumsetztabelle hat.
690 PRINT"Beim 664 und 6128 werden diese Zeichen auch auf dem Drucker
700 PRINT"korrekt ausgegeben.

```

Wenn Ihnen das Abtippen des obigen Programmes zu mühsam ist, dann spielen Sie doch einfach Lotto, vielleicht haben Sie Glück und gewinnen soviel, um sich einen neuen "Superrechner" leisten zu können, der eine deutsche Tastatur hat.

### 6.11.3 Listing: Lottovorschlag

```

100 'LOTTOVORSCHLAG
110 MODE 1:BORDER 5:INK 1,9:INK 3,9:SPEED INK 1,1
120 PEN 2:LOCATE 11,2:PRINT"Lotto-Vorschlag":PEN 1
130 '
140 'Gitter zeichnen
150 a=344:b=118
160 MOVE b,a:DRAW 455,a:a=a-32:IF a>119 THEN 160:ELSE a=342
170 MOVE b,a:DRAW b,120:b=b+48:IF b<455 THEN 170
180 '
190 'ZIEHUNG
200 CLEAR:FOR i=1 TO 6:gez=INT(RND(1)*48+1)
210 FOR y=1 TO i:IF gez(y)=gez THEN 200
220 NEXT y:gez(i)=gez:NEXT i
230 '
240 PEN 1:z=z+1:z$=MID$(STR$(z),2):FOR i=1 TO 6:IF z=gez(i) THEN PEN 3
250 NEXT i:IF LEN(z$)<2 THEN z$="0"+z$
260 '
270 LOCATE 9+h,5+v:PRINT z$:PEN 1:h=h+3
280 IF h<19 THEN z1$=STR$(z):GOTO 240

```

```

290 v=v+2:IF v<13 THEN h=0:GOTO 280
300 '
310 LOCATE 30,8:PRINT"Viel Glueck":LOCATE 32,10:PRINT"wuenscht
330 LOCATE 33,13:PRINT"Ihnen":LOCATE 32,16:PRINT"Ihr CPC
350 LOCATE 3,21:PEN 3: PRINT"Bitte eine Taste druecken ! E = Ende "
360 INK 1,9:INK 3,23,9:PEN 1
370 '
380 x*=LOWER$(INKEY$):IF x$="" THEN 360 ELSE IF x$="e" THEN 440
400 '
410 LOCATE 3,21:PEN 2:PRINT"Ich erarbeite einen neuen Vorschlag !
420 INK 1,9:INK 2,23:INK 3,9:INK 4,0:CLEAR:GOTO 200
430 '
440 INK 0,1:INK 1,24:INK 2,20:INK 3,6:INK 4,26:PEN 1
450 IF x$<>"e" THEN RETURN

```

Falls Sie aber mit obigem Programm kein Glück haben oder aber falls Sie doch mit einem Schneider weiterarbeiten wollen, dann sollten Sie, falls Sie Probleme mit der Belegung von Funktions-tasten haben sollten, das nachfolgende Programm in Ihren CPC ein-geben, denn dieses Programm zeigt, wie Tasten als Funktionstasten in allen drei Ebenen belegt werden können.

#### 6.11.4 Listing: TEBENE.BAS

```

100 'T-Ebenen
110 '
120 CALL &BB00:CLS
130 PRINT"Tastenbelegung in drei Ebenen":PRINT:PRINT
140 '
150 PRINT"Dieses Programm zeigt, wie Funktions-
160 PRINT"tasten in allen drei Tastatur-Ebenen
170 PRINT"belegt werden koennen.
180 PRINT:PRINT"Ausgewaehlt wurden:
190 PRINT:PRINT"Die Zifferntaste 4 =Tastaturcode 20
200 PRINT"Cursor nach links =Tastaturcode 08
210 PRINT"Die Taste v =Tastaturcode 55
220 PRINT:PRINT"Wie diese Belegung zu erfolgen hat,
230 PRINT"koennen Sie dem Listing entnehmen!
240 PRINT:PRINT"Druecken Sie die entsprechende Taste
250 PRINT"alleine, mit Shift oder mit CTRL !
260 PRINT:PRINT"Rueckstellung mit CTRL v !
270 '
280 'Belegung der Zifferntaste 4 im Ziffernblock
290 KEY 141,"4 normal"
300 KEY 142,"4 mit Shift"
310 KEY 143,"4 mit CTRL"
320 KEY DEF 20,0,141,142,143:' 0=keine Dauerfunktion
330 '
340 ' Belegung der Taste Cursor nach links
350 KEY 144,"Cursor links"
360 KEY 145,"mit SHIFT nach links"
370 KEY 146,"mit CTRL nach links"
380 KEY DEF 8,1,144,145,146
390 '
400 ' Belegung der Taste v auf CTRL-Ebene mit: Call &bb00

```

```

410 ' Reihenfolge KEY DEF Tastennummer, Dauerfunktion, Zeichen
420 ' Also: Taste 55,Dauerfunktion aus, Zeichen der drei Ebenen
430 KEY 147,"call &bb00"+CHR$(13)
440 KEY DEF 55,1,118,86,147:' Zeichen v,V,call &bb00+return

```

Wie Sie dem Programm entnehmen können, ist es nicht nur möglich, die Zifferntasten als Funktionstasten einzusetzen. Die Erklärungen über die Belegung sind im Programmlisting enthalten. Mit dem Verständnis dieses Programmes dürfte auch das Programm DEUTAST keine Geheimnisse mehr für Sie haben.

Sie haben nun bereits eine ganze Menge über den Bildschirm und seinen Speicher, über die Tastatur und deren Belegung erfahren. Trotzdem reicht dies aber in vielen Fällen noch nicht aus, Programme so zu schreiben, daß diese auch vom Bildschirmaufbau her einen guten Eindruck machen, deswegen nun noch einige Hinweise, die Ihnen bei Ihren Programmen nützlich sein können.

Zum Herausstellen von Textanzeigen können Sie "Herausstellungen" durch die Wahl verschiedener PENS erreichen. Es gibt aber auch die Möglichkeit, durch Austauschen der PEN und PAPER-Farbe auf Texte aufmerksam zu machen, oder Texte "hervorzuheben". Hierzu stellt der CPC sogar eine Maschinenroutine zur Verfügung, deren richtige Anwendung ganz einfach ist.

Wie bei fast allen Informationen, die wir Ihnen geben, auch hier wieder ein kleines Demonstrationsprogramm.

#### 6.11.5 Listing: Invertierung von PEN und PAPER

```

100 'TXT Inverse &bb9c
110 inv=&BB9C
120 '
130 FOR i = 1 TO 2
140 MODE i
150 PRINT"Dieses Programm zeigt, wie durch den ";
160 LOCATE 16,5:PRINT"Befehl";:PRINT
170 CALL inv:LOCATE 14,7:PRINT"CALL &bb9c":CALL inv
180 PRINT:PRINT:PRINT"die Farben fuer den Text-Stift und fuer Paper ";
190 PRINT"vertauscht werden koennen.":PRINT:PRINT
200 PRINT"Dies ";:CALL inv:PRINT"kann";:CALL inv:PRINT" auch ";:CALL inv
210 PRINT"in";:CALL inv:PRINT" einer ";:CALL inv:PRINT"Zeile";:CALL inv
220 PRINT" geschehen."
230 LOCATE 3,20
240 CALL inv:PRINT"Es geht in allen Bildschirm-Modes!":CALL inv:PRINT:PRINT
250 FOR k=0 TO 2000: NEXT k
260 NEXT i

```

(Die gleichen Möglichkeiten haben Sie aber auch durch den Befehl PRINT CHR\$(24)!)

Damit sind wir, wie Sie bemerken, bei Darstellungen in verschiedenen Farben angelangt. Derjenige, der mit den verschiedenen Farben und deren Auswahl aber noch Probleme hat, kann nachfolgendes Programm als Unteroutine in seinen Programmen einsetzen. Sie

bietet die Möglichkeit, BORDER, PEN und INK auszuwählen und diese Einstellung dann für das restliche Programm zu "halten".

#### 6.11.5 Listing: Farbwahl

```

100 'Farbwahl
110 MODE 2:DIM a$(26):FOR a=0 TO 26:READ a$(a):NEXT a:a=0:b=24:c=0:p=6
115 INK 0,a:INK 1,b:BORDER c
120 LOCATE 25,1:PRINT"SCHNEIDER CPC - FARBWAHL":PRINT:PRINT
130 PRINT"E = Ende      Farben:                Einstellungen:"
140 PRINT STRING$(79,"-")
150 LOCATE 1,p+4:PRINT STRING$(79,"-")
170 '
180 LOCATE 1,p+1:PRINT"(P) aper=    "a$(a):LOCATE 40,p+1:PRINT"INK    0,";a
190 LOCATE 1,p+2:PRINT"(S) tift =    "a$(b):LOCATE 40,p+2:PRINT"INK    1,";b
200 LOCATE 1,p+3:PRINT"(B) order=    "a$(c):LOCATE 40,p+3:PRINT"BORDER ";c
210 '
220 x$=LOWER$(INKEY$):IF x$="" THEN 220
230 IF x$="p" THEN a=a+1:x=a:GOSUB 350:a=x:INK 0,a:GOTO 180
240 IF x$="b" THEN c=c+1:x=c:GOSUB 350:c=x:BORDER x:GOTO 180
250 IF x$="s" THEN b=b+1:x=b:GOSUB 350:b=x:INK 1,b:GOTO 180
260 IF x$="e" THEN GOTO 380
270 GOTO 220
280 '
290 DATA SCHWARZ,BLAU,LEUCHTEND BLAU,ROT,MAGENTA,MAUVE,LEUCHTEND ROT
300 DATA PURPUR,LEUCHTEND MAGENTA,GRUEN,BLAUGRUEN,HIMMELBLAU,GELB,WEISS
310 DATA PASTELLBLAU,ORANGE,ROSA,PASTELLMAGENTA,LEUCHTEND GRUEN,SEEGRUEN
320 DATA LEUCHTEND BLAUGRUEN,LIMONENGRUEN,PASTELLGRUEN,PASTELL-BLAUGRUEN
330 DATA LEUCHTEND GELB,PASTELLGELB,LEUCHTEND WEISS
340 '
350 IF x>26 THEN x=0
360 FOR i = 1 TO 3:LOCATE 10,p+i:PRINT STRING$(22," "):NEXT i:RETURN
370 '
380 LOCATE 1,p+7:PRINT"Sollen die Farben so uebernommen werden ?"
390 x$=LOWER$(INKEY$):IF x$="" THEN 390 ELSE IF x$="j" THEN END
410 INK 0,1:INK 1,24:BORDER 1:END

```

Wer trotzdem noch Probleme für das Verständnis mit Farben und Stiften hat, dem sei nachfolgendes Programm angeraten. Dieses Programm ist ein Schnellkurs für die Farbprogrammierung der CPCs.

#### 6.11.7 Listing: Schnellkurs Farbprogrammierung

```

100 REM Schnellkurs Farbprogrammierung
110 REM Definitionen und Festlegungen
120 '
130 MODE 1:anz=14:DIM f$(anz):t1=10:t2=2000:t3=4000
140 INK 0,1:INK 1,24:INK 2,20:INK 3,6
150 '
160 nz$="Durch Eingabe von '^' faehrt das      Programm fort !
170 td$="      Bitte eine Taste druecken !"
180 k$="Nr. Farbe      Nr. Farbe
190 w1$=" Schnellkurs: Farbprogrammierung CPC  "

```

148

```

200 z$="      Zeitschleife - Moment bitte "
210 strich$=STRING$(37,CHR$(131))
220 '
230 f$(1)=" 0  Schwarz          14  Pastellblau
240 f$(2)=" 1  Blau             15  Orange
250 f$(3)=" 2  Hellblau        16  Rosa
260 f$(4)=" 3  Rot              17  Pastellmagenta
270 f$(5)=" 4  Magenta          18  Hellgruen
280 f$(6)=" 5  Hellviolett     19  Seegruen
290 f$(7)=" 6  Hellrot          20  helles Blaugruen
300 f$(8)=" 7  Purpur           21  Limonengruen
310 f$(9)=" 8  helles Magenta  22  Pastellgruen
320 f$(10)=" 9  Gruen           23  Pastellblaugruen
330 f$(11)="10  Blaugruen       24  Hellgelb
340 f$(12)="11  Himmelblau     25  Pastellgelb
350 f$(13)="12  Gelb           26  Leuchtendweiss
360 f$(14)="13  Weiss
370 '
380 REM Windows links,rechts,oben,unten
390 WINDOW #1,1,40,1,5:WINDOW #2,1,40,3,18
400 WINDOW #3,1,40,20,23:WINDOW #4,1,40,25,25
410 FOR i = 1 TO 4:PAPER #i,0:CLS #i:NEXT i
420 :
430 REM ----- eigentlicher Programm-Start -----
440 '
450 PAPER #1,2:PEN #1,3:PRINT#1,w1$
460 PRINT#2,"Ihr Schneider CPC hat zwar insgesamt
470 PRINT#2,"27 Farben, aber nur 16 Farbreister!":PRINT#2
480 PRINT#2,"Je nach Mode (0,1,2) sind 16, 4 oder 2
490 PRINT#2,"      dieser Register nutzbar.":PRINT#2: PRINT#2
500 PRINT#2,"      Es sind also auch im"
510 PEN #2,3:PRINT#2:PRINT#2,"      MODE 0":PEN #2,1:PRINT#2
520 PRINT#2,"      nur 16 der insgesamt 27 Farben":PRINT#2:PEN #2,2
530 PRINT#2,"      gleichzeitig":PEN #2,1:PRINT#2
540 PRINT#2,"      verfuegbar!"
550 '
560 PRINT #4,"Wollen Sie die Farbtabelle sehen ";PEN #4,3:PRINT#4,"j/n"
570 '
580 x$=INKEY$:IF x$<>"j" AND x$<>"n" THEN 580
590 IF x$="n" THEN 700:ELSE IF x$="j" THEN CLS #4:ft$=" Farbtabelle "
600 '
610 CLS #2:PEN #2,2:PRINT#2,k$:PEN #2,1
620 '
630 REM Farbdefinitionen ausgeben + Text: Farbtabelle ausgeben
640 FOR i = 1 TO anz: PRINT #2,f$(i):PRINT#4,MID$(ft$,i,1):NEXT i
650 '
660 PEN #3,2:PRINT#3,"      Dies sind die moeglichen Farben
670 PRINT#3,"      Ihres Schneider CPC "
680 PEN #2,1:CLS #4:PRINT#4,td$:GOSUB 2840
690 '
700 CLS #2:CLS #3:CLS #4
710 PRINT #2,"Die Grundeinstellung in jedem Mode ist:
720 PEN #2,3:PRINT#2:PRINT#2,"      BORDER 1:PAPER 0:PEN 1":PEN #2,1:PRINT#2
730 PRINT #2,"Beim Systemstart (Mode 1) sind folgende
740 PRINT #2,"Einstellungen automatisch erfolgt.":PRINT#2
750 PEN #2,2: PRINT #2,"      INK 0,1:INK 1,24:INK 2,20:INK 3,6":PEN #2,1
760 PRINT#2

```

```

770 PRINT #2,"Es sind also 4 Farbbregister (0 bis 3)";
780 PRINT #2," mit Werten vorbesetzt!
790 PRINT#2:PRINT#2,"Reg. 0 mit dem Wert 1 =Blau
800 PRINT#2,"Reg. 1 mit dem Wert 24=Heilgelb
810 PRINT#2,"Reg. 2 mit dem Wert 20=helles Blaugruen
820 PRINT#2,"Reg. 3 mit dem Wert 6 =Hellrot
830 '
840 PEN #3,2:PRINT #3,"Im Mode 1 sind die Register 4-16 nur
850 PRINT #3,"Wiederholungen der Register 0 bis 3 !":PEN#3,1
860 PRINT #4,td$:GOSUB 2840:CLS #4
870 '
880 CLS #3: text$=" Die Zahl hinter Border ist die Farb- Nummer !"
890 FOR s = 1 TO LEN (text$):PRINT#3,MID$(text$,s,1):GOSUB 2860:NEXT s
900 PRINT #4,td$:GOSUB 2840:CLS #3:CLS #4
910 '
920 PEN #3,2:PRINT#3," Dies gilt nicht fuer Paper !":PEN #3,1
930 PRINT#4,z$:GOSUB 2870:CLS #3
940 PRINT#3,"Die PAPER-Farbe wird durch das ausge- ";
950 PRINT#3,"waehlte Register bestimmt!!
960 PEN #3,2:PRINT #3," BORDER und auch PAPER
970 PRINT #3," erstrahlen deshalb in BLAU ":PEN #3,1
980 PRINT #4,td$:GOSUB 2840:CLS #4
990 '
1000 CLS #3:PEN #3,2:PRINT#3,"Die Zahl hinter PAPER gibt also das
1010 PRINT#3,"Register und nicht die Farbe an!":PEN #3,1
1020 '
1030 PRINT #4,td$:GOSUB 2840:CLS #2:CLS #3:CLS #4
1040 '
1050 PEN #2,3:PRINT #2,"D i e R e g i s t e r ":PEN #2,1
1060 PRINT #2:PRINT#2
1070 PRINT #2,"koennen mit Tintenfaessern verglichen
1080 PRINT#2,"werden. Die Register-Nummer bestimmt
1090 PRINT#2,"welches Fass betroffen ist.
1100 '
1110 PRINT#2,"Die Fuellung der Tintenfaesser erfolgt durch den INK-Befehl.
1120 PRINT#2:PEN #2,3:PRINT#2,"INK <Register>,<Farbe>":PRINT#2
1130 PEN #2,1:PRINT#2,"Der Befehl ";
1140 PEN #2,2: PRINT#2,"INK 0,1":PEN #2,1:
1150 PRINT#2," fueilt in das
1160 PRINT#2,"das Fass 'Null' die Farbe BLAU."
1170 PRINT#2,"Fueilen wir in das Fass 0 (=Register 0) ";
1180 PRINT#2,"einmal eine andere Farbe:
1190 PRINT#2,"Wir waehlen 'Schwarz', also den Wert 0.
1200 PRINT#2,"Deshalb: ":PEN #2,2:PRINT#2," INK 0,0":PEN #2,1
1210 '
1220 PRINT#4,td$:GOSUB 2840:CLS #4
1230 '
1240 INK 0,0:PRINT #4,z$
1250 GOSUB 2870:CLS #4
1260 INK 0,1:CLS #2
1270 PRINT#2,"Sie konnten nun sehen, was der Befehl!":PRINT #2
1280 PEN #2,3:PRINT#2," INK 0,0":PEN #2,1:PRINT #2
1290 PRINT#2,"bewirkte !":PRINT#2:PRINT#2:PRINT#2
1300 PRINT#2,"Durch diesen Befehl faerbte sich der
1310 PRINT#2,"Hintergrund (=PAPER) schwarz !
1320 '
1330 PRINT#2:PRINT#2

```

```

1340 PEN #2,2:PRINT#2,"Nun duerfen Sie es selbst versuchen.":PEN #2,1
1350 PRINT #4,td$:GOSUB 2840:CLS #2:CLS #4
1360 '
1370 FOR i= 1 TO anz:PRINT#2,f$(i):NEXT i
1380 '
1390 PRINT#2:PEN #2,2:PRINT#2,"Der Befehl INK 0,x ( Paper 0 )!":PEN #2,1
1400 CLS #4:CLS #3:PRINT#3,nz$
1410 '
1420 INPUT #4,"<INK 0,Farbwert> Farbwert ";fw$
1430 '
1440 fw$=LEFT$(fw$,2):IF LEFT$(fw$,1)="^" THEN 1540
1450 IF LEFT$(fw$,1) <"0" OR LEFT$(fw$,1) >"9" THEN fw=27:GOTO 1500
1460 '
1470 fw=VAL(fw$):CLS#3:IF fw<0 THEN 1540
1480 IF fw<27 THEN 1520
1490 '
1500 PRINT#3,"Das geht doch nicht !":GOSUB 2870:CLS #3:GOTO 1400
1510 '
1520 INK 0,fw:PEN #3,2:PRINT#3,z$:GOSUB 2870:INK 0,1:PEN #3,1:GOTO 1400
1530 '
1540 CLS #3:PRINT#3,"Wir machen mit dem Befehl BORDER weiter.":CLS#4
1550 '
1560 CLS#2:FOR i= 1 TO anz:PRINT#2,f$(i):NEXT i
1570 PRINT #2:PEN #2,2:PRINT#2,"Der Befehl BORDER":PEN #2,1
1580 '
1590 CLS #3:PRINT#3,nz$
1600 '
1610 CLS #4:INPUT #4,"Bitte die BORDER-Farbe waehlen ";fw$
1620 '
1630 IF LEFT$(fw$,1)="^" THEN 1740
1640 IF LEFT$(fw$,1) <"0" OR LEFT$(fw$,1) >"9" THEN fw=27:GOTO 1610
1650 '
1660 fw=VAL(fw$):CLS#3
1670 '
1680 IF fw>=27 THEN PEN #3,3: PRINT#3,"Was soll denn das?":PEN #3,1:GOTO 1720
1690 '
1700 BORDER VAL(fw$):CLS#3:PEN #3,2:PRINT #3,z$:PEN #3,1
1710 '
1720 GOSUB 2870:BORDER 1:GOTO 1590
1730 '
1740 REM Aenderung der Window-Paper
1750 CLS:CLS #2
1760 PRINT#2,"Wie Sie vielleicht schon feststellten,
1770 PRINT#2,"wird in diesem Programm mit der sog.
1780 PRINT#2,"WINDOW-Technik gearbeitet!":PRINT#2
1790 PEN #2,2:PRINT#2," Das heisst, in diesem Programm
1800 PRINT#2," sind 4 Textfenster definiert! ":PEN #2,1
1810 PRINT#2:PRINT#2," Damit Sie sehen, wo diese WINDOW's
1820 PRINT#2," liegen, zeigen wir sie Ihnen gleich.
1830 PRINT#4,td$:CALL &BB06:CLS#4
1840 REM Window's zeigen
1850 FOR i = 1 TO 4
1860 PAPER #i,3:PEN #1,1:CLS #i:PRINT #i,"Dies ist Window #":PEN #1,1
1870 GOSUB 2870:PAPER #i,0:CLS #i
1880 NEXT i
1890 PAPER #1,2:PEN #1,3:PRINT#1,w1$
1900 '

```



```

1910 PRINT#2,"Bei den naechsten Aufgaben wird nun
1920 PRINT#2,"im Moment nur noch Window 3 benutzt !":PRINT#2
1930 PRINT#2,"-Window 3 sehen Sie rot eingefaerbt-
1940 '
1950 PAPER #3,3:CLS #3:PRINT#4,td$:CALL &BB06:CLS#4
1960 '
1970 CLS#2:PRINT#2," Wir wollen uns nun mit dem Befehl!":PRINT#2
1980 PRINT #2:PEN #2,2:PRINT#2," PEN ";PEN #2,1
1990 PRINT#2:PRINT#2
2000 PRINT#2," beschaeftigen."
2010 PRINT#3," Im Augenblick hat der Pen fuer dieses
2020 PRINT#3," Fenster den Wert 1.
2030 PRINT#3," Der Befehl hierfuer lautet:
2040 PRINT#3," pen #3,1
2050 '
2060 PRINT#4,td$:CALL &BB06:CLS #4
2070 '
2080 PRINT#2:PRINT#2,"Wenn Sie nun eine Taste druecken, wird
2090 PRINT#2,"der Befehl PEN #3,0 ausgefuehrt !
2100 '
2110 PRINT#4,td$:CALL &BB06:CLS #2
2120 '
2130 PEN #3,0:PEN #2,3:PRINT#2,"Befehl PEN #3,0 ausgefuehrt":PEN #2,1
2140 PRINT#2:PRINT#2,"Wie Sie feststellen konnten, hat sich
2150 PRINT#2,"dadurch im Window #3 nichts geaendert!
2160 PEN #2,2:PRINT#2
2170 PRINT#2,"Dies wirkt sich erst aus, wenn neu
2180 PRINT#2,"geschrieben wird !"
2190 '
2200 PRINT#4,td$:CALL &BB06:CLS#4:CLS#2:PEN #2,1
2210 '
2220 PRINT#2,"Die Ausfuehrung erfolgt nach einem
2230 PRINT#2,"erneuten Tastendruck von Ihnen !
2240 '
2250 PRINT#4,td$:CALL &BB06:CLS #4
2260 '
2270 text$="Neuer Schreibvorgang im Window #3"
2280 FOR s=1 TO LEN(text$):
2290 PRINT#3,MID$(text$,s,1):GOSUB 2860:NEXT s
2300 '
2310 CLS #2:PRINT#2
2320 PRINT#2,"Sie koennen durch den Befehl CLS #3
2330 PRINT#2,"aber auch vorher das Fenster loeschen!
2340 PRINT#2
2350 PRINT#2,"Durch Tastendruck Ihrerseits wird dies
2360 PRINT#2,"durchgefuehrt und neu geschrieben.
2370 '
2380 PRINT#4,td$:CALL &BB06:CLS #3:CLS #4
2390 FOR s=1 TO LEN(text$):
2400 PRINT#3,MID$(text$,s,1):GOSUB 2860:NEXT s
2410 '
2420 CLS #2:PRINT#2:PEN #2,1
2430 PRINT#2,"Der Befehl PEN# x,y bestimmt in welches
2440 PRINT#2," der zur Verfuegung stehenden Register
2450 PRINT#2," der Schreibstift, eingetaucht wird.
2460 PRINT#2:PEN #2,3
2470 PRINT#2,CHR$(135);strich$:CHR$(139)

```

```

2480 PRINT#2,CHR$(133);" Denken Sie z.B. an einen Malkasten ";CHR$(138)
2490 PRINT#2,CHR$(133);" mit verschiedenen Farbschalen und ";CHR$(138)
2500 PRINT#2,CHR$(133);" verschiedenen Pinseln ";CHR$(138)
2510 PRINT#2,CHR$(141)}STRING$(37,CHR$(140));CHR$(142)
2520 PEN #2,2:PRINT#2
2530 '
2540 PRINT#2,"Sie malen also:
2550 PRINT#2,"an mehreren Bildern (Window's)
2560 PRINT#2,"und tauchen ihre Pinsel (PEN's)
2570 PRINT#2,"in Farbtoepfchen, die mit
2580 PRINT#2,"Farben (INK's) gefuehlt sind!
2590 '
2600 PRINT#4,td$:CALL &BB06:CLS#4:CLS#3
2610 '
2620 CLS#2:PRINT#2,"Im Mode 1 haben wir 4 Farbtoepfe
2630 PRINT#2,"die im Augenblick die folgenden
2640 PRINT#2,"Standardfarben beinhalten:
2650 PRINT#2
2660 PRINT #2,"INK 0,1 = BLAU
2670 PRINT #2,"INK 1,24 = Hellgelb
2680 PRINT #2,"INK 2,20 = helles Blaugruen
2690 PRINT #2,"INK 3,6 = Hellrot
2700 PRINT#4,td$:CALL &BB06:CLS#4:CLS#3
2710 CLS #2:PRINT#2
2720 PRINT#2:PRINT#2,"Durch den Befehl: PEN #3,x (x=1 bis 4)
2730 PRINT#2,"werden folgende Ausgaben erreicht:
2740 FOR i = 1 TO 4
2750 PEN #2,i:PRINT#2,"test
2760 PEN #3,i:PRINT#3,"test
2770 NEXT i:PRINT#2,"Sie sehen, dass bei jedem Ausgabeblock
2780 PRINT#2,"eine Farbe fehlt. Dies liegt daran,
2790 PRINT#2,"dass auch durch den Hintergrund eine
2800 PRINT#2,"der 4 Farben belegt ist!":PRINT#2
2810 PRINT#2,"Ende des Schnellkurses !
2820 '
2830 INK 0,1:INK 1,24:INK 2,20:INK 3,6:LOCATE 1,24:END
2840 CALL &BB06:RETURN
2850 END:REM *****
2860 FOR t = 1 TO t1:NEXT t:RETURN :REM kurze Schleife
2870 FOR t = 1 TO t2:NEXT t:RETURN :REM mittlere Schleife
2880 FOR t = 1 TO t3:NEXT t:RETURN :REM lange Schleife
2890 PRINT#2,CHR$(135):strich$:CHR$(139)

```

#### 6.11.8 Die Steuerzeichen der CPCs

Beim C64 von Commodore hatten sich in der Anfangszeit viele Besitzer immer beschwert, dass die Steuerzeichen in veröffentlichten Listings immer irgendwelche Herzchen, Kreise, Blätter usw. waren und deswegen äußerst schwierig zu identifizieren. Die Zeitschriften-Redaktionen fanden aber bald Möglichkeiten, dies abzustellen und beseitigten damit das Ärgernis.

Auch Sie als CPC-Besitzer können Ihre Mitmenschen mit ähnlichen Steuer-codes ärgern.

Sie brauchen bei einem Print-Befehl nämlich nur Zeichen einzuge-

ben, die Sie durch gleichzeitiges Betätigen der CTRL-Taste erreichen.

Diese Steuerzeichen können Sie unbedenklich in eigenen Programmen einsetzen. Wenn Sie aber Listings - zum Zwecke des Abdrucks in einer Zeitschrift - weitergeben wollen, verzichten Sie bitte auf derartige Scherze, denn diese Steuerzeichen können ohne größeren Programmieraufwand nicht ausgedruckt werden.

Der Sinn der nachfolgenden Aufstellung ist vor allem im Hinweis zu sehen, wie derartige "Hieroglyphen" zustande kommen und um anhand des Handbuches dann diese Codes zu verstehen, falls sie in fremden Programmen auftauchen.

Eingabe CTRL und	Effekt	ASCII-Wert	Name
a	gibt Enterpfell aus	1	SOH
b	schaltet den Textcursor ab	2	STX
c	schaltet den Textcursor an	3	ETX
d	setzt den Bildschirmmodus	4	EOT
e	sendet Parameterzeichen zum Grafikcursor	5	ENQ
f	Schaltet auf Textbildschirm um	6	ACK
g	Klingelzeichen ertönt	7	BEL
h	Cursor links (Backspace)	8	BS
i	Cursor rechts (horiz.Tab)	9	HT
j	Cursor down (Line-Feed)	10	LF
k	Cursor hoch (Vert.Tab)	11	VT
l	cls (löscht das Textfenster)	12	FF
s	Bildschirm löschen, Cursor Zeile 25	19	DC3
^	Cursor home	..	...
0	lässt Border blinken	..	...

Die weiteren Steuerzeichen können Sie sich nun selbst "erlesen", wenn Sie im Handbuch einmal bei den ASCII-Zeichen nachsehen.

Zum Abschluß der gesamten Informationen über den Bildschirm nun ein Zeichenprogramm, mit dem Sie in der durch den CPC vorgegebenen höchstmöglichen Auflösung Zeichnungen erstellen können.

Es handelt sich also dabei um ein kleines CAD-Programm, mit dem Sie Zeichnungen erstellen, abspeichern, einlesen, beschriften und auch ausdrucken können. Das Programm hätte für den CPC 664 und 6128 zwar in einigen Punkten "besser" programmiert werden können, wenn wir die neuen Grafik-Befehle eingesetzt hätten, wir wollten aber auch dieses Programm für alle CPCs schreiben und deswegen haben wir auf diese Befehle verzichtet.

Trotzdem haben wir zum Beispiel Automatik-Routinen zur Erstellung von Rechtecken und Kreisen eingebaut und neben dem Maschinenprogramm zur Erstellung von Hardcopien nur eine - lediglich ein paar Byte lange - zusätzliche Maschinenroutine programmiert.

## 6.11.9 Listing: MINIMAL.BAS

```

100 ' MiniMal-Programm mit Cursor-Steuerung
110 '
120 ' Speicher begrenzen
130 MEMORY &9FFF
140 IF PEEK(&A16C) <> &FE THEN LOAD "hc.bin",&A16C
150 '
160 ' Maschinenroutine fuer Zeichenprogramm lesen und poken
170 ' Einspruege a000,a006,a00c
180 DATA 3e,0,cd,59,bc,c9,3e,1,cd,59,bc,c9,3e,3,cd,59,bc,c9
190 FOR i = 0 TO 17:READ a$:a$="&" + a$:a=VAL(a$)
200 POKE &A000+i,a:NEXT:CALL &A006
210 '
220 ' Ausgabemeldungen definieren
230 m$(0)="bewegen":m$(1)="zeichnen":m$(2)="loeschen"
240 w3$="COPY =MODE    b =Box  k =Kreis  h =Hardcopy  r =Raster  t =Text"
250 w4$="g      =goto Position  l =laden  s =speichern  e =Ende    c =cls"
260 '
270 MODE 2:'                80-Zeichen-Darstellung
280 BORDER 0:'             Rand schwarz
290 ORIGIN 0,18:'          Startpunkt Grafikcursor auf 0,18
300 '
310 ' Fenster definieren
320 WINDOW #1,1,80,1,1:'    Kopfzeile
330 WINDOW #2,1,80,24,25:'  unterer Rand
340 '
350 ' Farben und Farbstifte bestimmen
360 INK 0,0:INK 1,26:PEN #1,1:PEN #0,1
370 '
380 ' Paper zuordnen
390 FOR i = 1 TO 2:PAPER #i,0:NEXT i
400 '
410 ' Anfangskoordinaten fuer Cursormittelposition
420 x=312:y=188
430 '
440 '-----Ende der Zuordnungen, Definitionen usw.-----
450 '
460 PRINT #1," Mini-Mal-Programm fuer alle CPC's ";
470 ' Warteschleife
480 FOR i = 1 TO 500:NEXT
490 CLS #1:CLS #2:PRINT#2,TAB(8)w3$:PRINT#2,TAB(8)w4$
500 '
510 '
520 CALL &A000
530 MOVE 0,365:DRAW 639,365,1:MOVE 0,15:DRAW 639,15,1
540 CALL &A006
550 '
560 '-----Hauptprogramm-----
570 '
580 ' Mode-String ausgeben
590 PRINT#1,"MODE: "m$(flag)
600 LOCATE #1,50,1:PRINT#1,"Position: x= "x" y= "INT((y-16)/2);
610 '
620 ' Eingabe
630 b$=LOWER$(INKEY$)
640 '

```

```

650 ' Blinkschleife Cursor
660 FOR i =1 TO 2:PLOT x,y,0:PLOT x,y,1
670 GOSUB 1180:'
680 NEXT i
690 '
700 IF b$="" THEN 630:' Keine Eingabe deshalb wieder zur Eingabezeile
710 '
720 IF b$="b" THEN GOSUB 1540:GOTO 460:' Box
730 IF b$="c" THEN CLG:GOTO 460:' Grafikschirm loeschen
740 IF b$="e" THEN PEN 1:CLS #2:LOCATE 1,24:END:' Programmende
750 IF b$="g" THEN GOSUB 1320:GOTO 460:' Cursor positionieren
760 IF b$="h" THEN GOSUB 1420:GOTO 460:' Hardcopy
770 IF b$="k" THEN GOSUB 1470:GOTO 460:' Kreis
780 IF b$="l" THEN GOSUB 1270:GOTO 420:' Bild laden
790 IF b$="r" THEN GOSUB 1640:GOTO 460:' Raster
800 IF b$="s" THEN GOSUB 1220:GOTO 420:' Bild abspeichern
810 IF b$="t" THEN GOSUB 1380:GOTO 460:' Texteingabe
820 b = ASC(b$)
830 '
840 'ueberpruefung ob Copy-Taste betaetigt wurde
850 IF b = 224 THEN flag=flag +1:IF flag =3 THEN flag =0:GOTO 590 ELSE 590
860 '-----
870 ' quickmove - Shift und Cursorsteuertaste gleichzeitig
880 IF b=244 THEN y = y+10
890 IF b=245 THEN y = y-10
900 IF b=246 THEN x = x-20
910 IF b=247 THEN x = x+20
920 '-----
930 ' bewegen
940 IF flag = 0 THEN PLOT x,y,0:GOSUB 1080
950 '
960 ' zeichnen
970 IF flag = 1 THEN CALL &A000:PLOT x,y,1:GOSUB 1080
980 '
990 ' loeschen
1000 IF flag = 2 THEN GOSUB 1080:MOVE x,y:m= TEST(x,y):PLOT x,y,m
1010 '
1020 ' Sprung zur Eingabezeile
1030 GOTO 630
1040 '
1050 '***** Unterprogramme *****
1060 '
1070 'Neue Position berechnen
1080 IF b = 240 THEN y=y+2:ELSE IF b=241 THEN y=y-2:' nach oben oder unten
1090 IF b = 242 THEN x=x-1:ELSE IF b=243 THEN x=x+1:' nach links oder rechts
1100 '
1110 ' Abfrage ob Endpunkte erreicht, ggf. korrigieren
1120 IF x<=0 THEN x=0:ELSE IF x>=639 THEN x=639:' X-Achse
1130 IF y<=16 THEN y=16:ELSE IF y>=363 THEN y=363:' Y-Achse
1140 '
1150 LOCATE #1,50,1:PRINT#1,"Position: x= "x" y= "INT((y-16)/2)
1160 CALL &A006:RETURN
1170 '-----
1180 ' Malpinsel ausgeben
1190 MOVE x-3,y+1:TAG:PRINT CHR$(160):RETURN
1200 '-----
1210 'abspeichern

```

```

1220 PRINT#1,"MODE:Speichern ";:INPUT #1,"Name (maximal 8 Zeichen !!) ";na$
1230 CLS #1:CLS #2:SAVE LEFT$(na$,8)+".pic",b,&C0000,&3FFF
1240 RETURN
1250 '-----
1260 'laden
1270 PRINT#1,"MODE:Laden ";:INPUT #1,"Name (maximal 8 Zeichen !!) ";na$
1280 CLS#1:CLS #2:LOAD LEFT$(na$,8)+".pic",&C0000
1290 RETURN
1300 '-----
1310 'Cursor positionieren
1320 PRINT#1,"MODE: Positionieren ";:
1330 PRINT #1,"goto ";:INPUT #1,"x,y ";x,y
1340 y=y*2+16:GOSUB 1120
1350 RETURN
1360 '-----
1370 'Text eingeben
1380 CALL &A006:PRINT #1,"MODE: Text";:INPUT #1,t$
1381 MOVE x,y:TAG:PRINT t$;:GOSUB 1120
1390 RETURN
1400 '-----
1410 'Hardcopy
1420 PRINT #1,"MODE: Hardcopy
1430 CALL &A16C,19,364
1440 RETURN
1450 '-----
1460 'Kreis
1470 PRINT #1:INPUT #1,"MODE: Kreis Durchmesser,Step ";d,st
1480 DEG:CALL &A00C
1490 FOR a=360 TO 0 STEP -st
1500 PLOT x+d*COS(a),y+d*SIN(a)
1510 NEXT
1520 RETURN
1530 '-----
1540 PRINT #1,"MODE: Box ";
1550 INPUT #1,"Breite/Hoehe";b,h
1560 CALL &A000:MOVE x,y
1570 DRAW x,h+y: ' aufwaerts
1580 DRAW x+b,y+h: ' rechts
1590 DRAW x+b,y: ' abwaerts
1600 DRAW x,y: ' links
1610 RETURN
1620 '-----
1630 'Raster
1640 CALL &A00C
1650 PRINT #1,"MODE: Raster ";
1660 INPUT #1,"Rastergroesse ";g
1670 IF g=0 THEN g=m
1680 m=g
1690 FOR j = 1 TO 3
1700 FOR i = 0 TO g
1710 b=INT(RND * g):c=-INT(RND * g)
1720 PLOT x+b,y+c:PLOT x-b,y+c:PLOT x+b,y-c:PLOT x-b,y-c
1730 NEXT:NEXT
1740 RETURN

```

Auch dieses Programm, stellt Ihnen - trotz seiner bereits sehr großen Leistungsfähigkeit - nur eine Grundroutine zur Verfügung und soll Sie animieren, das Programm noch zu erweitern, denn es sind noch viele zusätzliche Möglichkeiten programmierbar. Eine davon könnte zum Beispiel das Umkopieren von Bildschirmteilen sein. Wie etwas derartiges programmiert werden kann, haben wir bei der Bewegung der Sprites aufgezeigt.

Damit sind wir nun am Ende dessen angelangt, was wir Ihnen über den Bildschirm und die Tastatur mitteilen wollten.

Was aber noch fehlt, ist die Beschreibung zur Programmbedienung des MINIMAL-Programmes. Da das Programm aber durch das Menüfeld alle Hinweise enthält, die möglich sind, glauben wir, daß es nicht erforderlich ist! Lediglich ein Hinweis. Durch Betätigung der COPY-Taste schalten Sie den Betriebsmode um, das heißt Sie wählen dadurch bewegen, zeichnen und löschen aus.

(Falls Sie es noch nicht bemerkt haben sollten: Für Ausdrücke wird die Hardcopy-Routine des Monitors eingesetzt!)

## 7. Die Firmware-Pakete (Packs)

Alle CPCs haben eine gegliederte und in sogenannte Packs (frei übersetzt mit: Pakete) aufgeteilte Firmware. Dies sind Abschnitte in der Firmware, die in sich geschlossen sind und ganz bestimmte Aufgabenbereiche umfassen.

Deshalb lassen sich diese Pakete auch aufgrund bestimmter Zuständigkeiten in Gruppen einteilen.

Sinn dieser Einteilung ist es vor allem, gemeinsam benötigte Einsprünge sowohl von der eigenen Firmware, als auch von anderen Programmen aus, sinnvoll und effektiv nutzen zu können.

Unter diesem Gesichtspunkt sind auch die Sprungtabellen im RAM der CPCs zu sehen, die ebenfalls abgeschlossene Einheiten bilden.

Doch nun erst einmal zu den im Firmware-Handbuch und auch anderen technischen Unterlagen aufgeführten Paketen.

In der nachfolgenden Aufstellung sind sie in der Reihenfolge genannt, wie sie im Firmware-ROM abgelegt sind. Die unter dem jeweiligen CPC-Typ genannten Adressen sind die Startadressen, an denen das jeweilige Paket im ROM beginnt.

### 7.1 Übersicht der Firmware-Packs

Paket	464	664	6128
Kernal	0000	0000	0000
Machine Pack	0580	057B	0591
Screen-Pack	0AA0	0ABB	0ABF
Text-VDU	1078	1070	1071
Graphics-VDU	15B0	15A4	15A8
Key-Manager	19E0	1B5C	1B5C
Sound-Pack	1E68	1FE9	1FE9
Cassette-Manager	2370	24BC	24BC
Editor-Pack	2A98	2C02	2C02

Laut Firmware-Manual gibt es dabei die folgenden Zugehörigkeiten:

Tastatur	-	Key-Manager
Bildschirm	-	Text-VDU, Graphics-VDU, Screen-Pack
Cassette	-	Cassette-Manager
Sound	-	Sound-Manager
Operating-System	-	Kernel, Machine-Pack, Jumper



## 7.2 Die Aufgaben der einzelnen Pakete:

Der Keymanager hat die Aufgabe, die Tastatur zu verwalten. Das heißt, die Abfrage der Tastatur (einschließlich der Funktionstasten und der Joysticks), sowie Tests auf Abbruchbedingungen und auch das Erzeugen von Zeichen wird durch diesen Teil erledigt.

Das Paket Text-VDU sendet die Text-Zeichen an den Bildschirm und setzt auch die Steuercodes um.

Bei Grafiken erledigt diese Aufgaben das Paket Grafik-VDU. Also das Zeichnen von Punkten, Linien, Testen auf gesetzte Punkte u.ä.

Die Schnittstelle zwischen den beiden VDU- (=Video Display Unit) Paketen und der Bildschirm-Hardware wird durch das Screen-Pack gebildet.

Für das Lesen und Schreiben in Verbindung mit der Kassette ist der Cassette-Manager zuständig. Ebenso wird durch ihn der Kassetten-Motor ein- und ausgeschaltet.

Für guten Klang und Geräusche aller möglichen Art, ist in Verbindung mit dem Sound-Generator der Sound-Manager verantwortlich.

Zum Operating System, dem Betriebssystem-Kern (Kernel) ist zu sagen, daß dieser Teil alle Unterbrechungen und Ereignisse verwaltet und außerdem den Zugriff auf die ROMs regelt, sowie den Ablauf der Programme bestimmt.

Das Machine-Pack (Maschinen-Paket) verwaltet den Drucker und ist auch für die Ansteuerung der Hardware zuständig.

Die Jumper sind für den Aufbau der Sprungtabelle verantwortlich.

Alle Einsprünge in die Firmware sollten immer über die Sprungtabellen erfolgen! Nur so kann bei den Programmen eine Aufwärtskompatibilität erreicht werden.

Softwarehäuser, die diese Festlegung nicht berücksichtigt hatten, mußten nach dem Erscheinen der CPC 664- und 6128-Serie erstaunt feststellen, daß die 464-Programme auf diesen Nachfolgern dann nicht mehr liefen, wenn diese auf die Firmware direkt zugriffen und nicht über die Sprungtabelle die verschiedenen Routinen aufriefen.

Sinn dieser Tabelle war und ist es, daß auch bei Änderungen des Betriebssystems diese Einsprünge gleich bleiben und deshalb Programme auf jedem CPC-Typ laufen.

Eine weitere Besonderheit bei allen CPCs sind die Indirections, Speicherstellen, die vor allem dazu dienen, Firmwareroutinen ggf. abzufangen und zu modifizieren.

Verschiedene ROM-Routinen springen bei ihrer Abarbeitung diese Speicherstellen an, um dann von dort aus, weiter- oder umgeleitet zu werden. In diesen Speicherstellen stehen also die "Wegweiser", die dem ablaufenden Maschinenprogramm den weiteren Pfad zeigen.

Die Inhalte dieser Speicherbytes können verändert werden.

Beim CPC 464 sind unter anderem auch die Indirections im Bereich von &AC00 bis &AC19 interessant. Diese erlauben es nämlich, den "Wegweiser" auf ein eigenes Maschinenprogramm zu richten, das zum Beispiel ein geschütztes Basic-Programm wieder "enttarnt".

Indirections sind also Speicherstellen, in denen normalerweise drei Bytes für Sprungbefehle oder Returns anzutreffen sind. Durch Umlenken auf eigene Routinen, kann dadurch Einfluß auf einige Programmabläufe genommen werden.

Neben den bereits für den CPC 464 genannten, gibt es auch andere Indirections. Diese liegen von &BD0D bis &BDF1 im RAM.

Auch diese Indirections erlauben es, Änderungen an bestehenden Firmware-Routinen durchzuführen.

So ist es ohne große Probleme möglich, z.B. Zeichen, die eigentlich an den Drucker geschickt werden würden, auf die Diskettenstation oder auf den Bildschirm umzulenken.

Da alle diese Sprungtabellen im zentralen RAM liegen, kann dort hin ohne Rücksicht auf die ausgewählten (eingeschalteten) ROMs gesprungen werden.

Das zentrale RAM liegt im Bereich von &4000 bis &BFFF und wird deshalb als zentrales RAM bezeichnet, da es von keinem ROM überlagert wird und zentral in der Mitte des RAM-Bereiches liegt.

Die Hauptsprungtabelle ist von &BB00 bis &BD39 zu finden.

Jeder Eintrag belegt drei Bytes und ist für die Verwendung von LOW-JUMP-Restarts (RST 1,xxxx) initialisiert.

Diese Tabelle wird durch die Firmware nicht mehr verändert. Geänderte Sprünge bleiben deshalb bis zu einer weiteren Änderung oder bis zum Ausschalten des CPC erhalten.

Eine Neuinitialisierung - nach einer Änderung - kann durch Aufruf von JUMP RESTORE (&BD37) erfolgen. Der Befehl CALL &BD37 bringt also die Standard-Sprungtabelle wieder in den Originalzustand.

### 7.3 Die Unterteilung der Einträge:

Sprungtabelle des unteren Betriebssystem-Kerns

&0000 bis &003B

Sprungtabelle des oberen Betriebssystem-Kerns

&B900 bis &B921

Hauptsprungtabelle

&BB00 bis &BB4B	Tastaturverwaltung	(KM)
&BB4E bis &BBB7	Text-VDU	(TXT)

&BBBA bis &BBFC	Grafik-VDU	(GRA)
&BBFF bis &BC62	Bildschirmpaket	(SCR)
&BC65 bis &BCA4	Kassettenverwaltung	(CAS)
&BCA7 bis &BCC5	Tongeneratorverwaltung	(SOUND)
&BCC8 bis &BD10	Betriebssystemkern	(KL)
&BD13 bis &BD34	Maschinenpaket	(MC)
&BD37	Jumper (Jump Restore)	

Beim CPC 464 nicht enthalten sind die folgenden Einträge in den Sprungtabellen des CPC 664/6128:

```
&B92A KL SCAN NEEDED
&BD3A KM SET LOCKS
&BD3D KM FLUSH
&BD40 TXT ASK STATE
&BD43 GRA DEFAULT
&BD46 GRA SET BACK
&BD49 GRA SET FIRST
&BD4C GRA SET LINE MASK
&BD4F GRA FROM USER
&BD52 GRA FILL
&BD55 SCR SET POSITION
&BD58 MC PRINT TRANSLATION
&BDF4 KM SCAN KEYS
```

Beim CPC 6128 steht zusätzlich bei &BD5B die Routine KL Bank Switch.

Leider haben sich auch manche Bedingungen für die Ein- und Aussprünge bei verschiedenen Routinen geändert, sodaß das Firmware-Handbuch für den CPC 464 nicht immer für den 664 und 6128 "stimmt".

An einem Beispiel wollen wir dies aufzeigen.

Die Routine CAS OUT DIRECT hat lt. Firmware-Handbuch folgende Aussprung-Bedingungen (Auszugsweise!):

Wenn die Datei nicht wie erwartet eröffnet war:

```
CARRY-Flag "aus"
ZERO-FLAG "aus"
.....
```

Immer:

A,BC,DE,HL,IX und die anderen Flags zerstört, alle anderen Register unverändert.

Für den 664 gilt aber:

A enthält den Fehlercode!

Ähnliches gilt anscheinend auch für einige andere Ein-/Aussprungbedingungen und deshalb sollten Maschinenprogrammierer versuchen, die entsprechenden Unterlagen für "ihre" CPCs zu erhalten. Kurz

vor der Fertigstellung dieses Buches erreichte uns noch die Nachricht, daß die englische Version des Firmware-Handbuches für den CPC 6128 verfügbar ist.

Weitgehend unbekannt sind auch die Angaben für die im CPC 664 und 6128 enthaltenen Drucker-Übersetzungstabellen "Printer Translation Tabelle". Diese setzen die Character im Bereich von &A0 bis &AF in die jeweiligen Landeszeichen um. Standard-ASCII oder Grafik-Zeichen werden nicht umgesetzt. Wird die Maschinen-Routine MC PRINT CHAR aufgerufen, dann werden Zeichen im angegebenen Bereich mittels der Drucker-Übersetzungstabelle decodiert. Übersetzt wird in amerikanisch/englische, französische, deutsche und spanische Zeichen.

Wird das Zeichen nicht in dieser Tabelle gefunden, dann wird es normal ausgegeben, ansonsten wird das Übersetzungszeichen gesendet. Ist dieses aber &FF, dann unterbleibt die Ausgabe.

#### 7.4 Weitere Indirections und Speicherbereiche:

&BDCD bis &BDD9	Text-VDU-Indirections
&BDDC bis &BDE2	Grafik-VDU-Indirections
&BDE5 bis &BDEB	Bildschirm-Paket-Indirections
&BDEE	Tastatur-Verwaltungs-Indirection
&BDF1	Maschinen-Paket-Indirection

Jedes Paket hat auch noch eigene RAM-Bereiche, die aber bei den verschiedenen CPCs auch wieder unterschiedlich sind.

#### 7.5 Die RAM-Bereiche

CPC 464	CPC 664/CPC 6128
Kernal-RAM	
&B100 bis &B1C7	&B82D bis &B8FF
Machine-Pack	
nicht vorhanden	&B804 bis &B82C
Screen-Pack	
&B1C8 bis &B20B	&B7C3 bis &B803
Text-Screen-Pack	
&B20C bis &B327	&B6B5 bis &B7C2
Grafik-Screen-Pack	
&B328 bis &B34B	&B693 bis &B6A8 &B6A9 bis &B6B4
Keyboard-Manager	
&B34C bis &B54F	&B496 bis &B692
Sound-Manager	
&B550 bis &B7FF	&B1ED bis &B495

Cassette-Manager  
&B800 bis &B8DB

&B118 bis &B1EC

Editor-RAM  
&B8DC bis &B8E3

&B114 bis &B117

In diesen Speicherbereichen sind die Pakete also ebenfalls aktiv.

Aber auch durch das Basic sind einige Speicherstellen in Gebrauch. Dieser Bereich liegt von &AC00 bis &B0FF. Er ist zwar für alle drei CPC-Typen gleich, jedoch hat sich die Bedeutung für den CPC 464 und den 664/6128 geändert.

So ist beispielsweise der Zeiger für HINEM beim 464 in &AE7B und &AE7C, bei den beiden anderen CPCs aber in &AE5E und &AE5F zu finden.

Deshalb kommt es bei Programmen, die derartige Speicherstellen benutzen, zu Problemen wenn sie auf dem verkehrten CPC laufen. Hier helfen also nur zusätzliche Unterlagen (z.B.:ROM-Listing) weiter.

Bei Zugriffen auf diese Bytes ist die Kenntnis der genaueren Details wichtig, denn sonst wird schwer, Programme umzuschreiben. Eine Gegenüberstellung von System-Speicherstellen finden Sie im Anhang.

Für diejenigen, die sich nun mit den einzelnen Befehlen, bzw. den Routinen, die über die Sprungtabellen angesprungen werden weiter auseinandersetzen wollen, ist, um nähere Details zu erfahren, auch wichtig, wo denn diese Routinen stehen.

Dies kann mittels eines Disassemblers leicht festgestellt werden. So erfolgen die Sprünge ab &BB00 über den Restart-Befehl RST1,x. Der Wert x gibt dabei die Adresse an.

Allerdings nicht im "Klartext", sondern verknüpft mit dem "ROM-State". Das heißt, die entsprechende ROM-Nummer muß bei diesem Befehl mit enthalten sein. Diese Angabe wird im Bit 07 mitverschlüsselt. Der Befehl RST 1,&99E0 springt also bei &19E0 ein. Dort ist beispielsweise beim CPC 464 die Routine KM INITIALIZE die über Call &BB00 aufgerufen wird. (Dieser Restart-Befehl weist beim 664/6128 natürlich auf eine andere Einsprungsadresse hin.)

Interessenten können sich diese Adressen ihres Computers mittels eines Disassemblers ab &B900 "abholen" allerdings wird, um diese Routinen dann zu verstehen, ein kommentiertes ROM-Listing benötigt.

Während der Überarbeitung dieses Kapitels bekamen wir ein ROM-Listing des Markt & Technik-Verlages in die Hand, welches für viele CPC-Besitzer ganz bestimmt eine wertvolle Hilfe sein kann. Der Titel des Buches: "ROM-Listing CPC 464/664/6128".

Zwar sind in diesem nicht alle Unterschiede und auch nicht alle Tabellen der 664- und 6128-ROMs enthalten, aber dieses Buch ist

eine "wertvolle" Arbeitshilfe und derzeit wohl auch die ausführlichste Dokumentation der CPC-Firmware-ROMs.

Da in England zum Zeitpunkt der Niederschrift dieses Kapitels ein Firmware-Handbuch für den CPC 664/6128 bereits erhältlich sein soll, dauert es hoffentlich nicht mehr allzu lange, bis es dieses auch in deutsch gibt. Wahrscheinlich aber können auch schon vorab englische Ergänzungsblätter zu den 464-Firmware-Handbüchern bei Schneider angefordert werden.

Mit Sicherheit sind - nicht nur aufgrund der soeben erwähnten Punkte - derartige Unterlagen für "Maschinenprogrammierer" ein unbedingtes "Muß".

Im nächsten Abschnitt wollen wir uns etwas mit den parallel anschließbaren ROMs beschäftigen, denn auch dieses Wissen wird bei einigen Programmen dieses Buches ausgenutzt.

## 8. Die Sideways-ROMs

Die Schneider CPCs erlauben es, theoretisch bis zu 252 sogenannte Sideways-ROMs anzuschließen. Theoretisch deshalb, weil dies vermutlich bisher noch kein Anwender getan hat und wahrscheinlich auch nie jemand tun wird.

Diese Technik bietet allerdings hervorragende Möglichkeiten, die man auf den ersten Blick vielleicht gar noch nicht so zu schätzen weiß.

Doch immer der Reihe nach.

Der Z80 kann bekannterweise nur 64 KByte Adressraum verwalten. Aber bereits intern im CPC wird dieser Bereich schon überschritten. Warum sollten deshalb die Entwickler des CPCs nicht auch für externe Ansteckplatinen die Adressierung erlauben? Und die Realisierung dieser Möglichkeit "öffnet" vielen interessanten Zusätzen die Tür zum CPC.

So gut durchdacht, wie bei den CPCs, ist es bisher bei keinem anderen Computer dieser Preisklasse realisiert worden.

Falls Sie nun glauben, das wäre für Sie zu uninteressant, weil Sie ja doch nie eine derartige Erweiterung anstecken wollen, dann lesen Sie trotzdem weiter, denn vielleicht haben Sie in Ihrem CPC bereits ein Sideways-ROM und wissen es noch nicht. Vielleicht aber haben Sie auch einen CPC 464 und sind verärgert, daß 664-Programme nicht laufen. Möglicherweise aber sind Sie als Besitzer des 6128 darüber traurig, daß alte 464-Programme auf Ihrem Computer nicht laufen wollen. Für alle gibt es Abhilfemöglichkeiten, die wir unter anderem in diesem Kapitel aufzeigen.

Das Konzept bei den Sideways-ROMs ist so, daß parallel zum normalen Bildschirmbereich noch Zusatz-ROMs geschaltet werden können. (Normaler Bildschirmbereich deshalb, weil dieser auch z.B. auf die Adresse &4000 gelegt werden kann!)

Um es adressmäßig korrekt darzustellen, nun die Angabe, daß derartige Zusatzbausteine dem Adressbereich &C000 bis &FFFF parallel liegen.

Da die CPCs aber parallel zu den RAMs in diesem Bereich bereits einen Teil des Firmware-ROMs haben, kann dieses schon als erstes Sideway-ROM betrachtet werden.

Diese Sideway-ROMs werden - über eine entsprechende Adressdekodierung durch die Hardware - immer aktiv oder inaktiv geschaltet.

Besitzer von Diskettenstationen haben bereits ein weiteres zusätzliches Sideway-ROM, nämlich das Betriebssystem des Diskettencontrollers. Dabei spielt es keine Rolle, ob die Diskettenstation eingebaut ist oder von außen angeschlossen wurde.

Auch andere Erweiterungen gibt es, die Sideway-ROMs enthalten. So ist beispielsweise ein Entwicklungssystem (MAXAM), ein komplettes Textverarbeitungsprogramm (PROTEXT) und es sind auch Utilities (UTOPIA oder PROGRAMMERS TOOLKIT) als derartige ROMs erhältlich.

Von der Firma VORTEX werden für die Diskettenstationen ebenfalls Sideway-ROMs eingesetzt.

Wie weiß nun aber die Soft- und Hardware welches ROM aktiviert werden soll, welches als sogenanntes Vordergrund- oder Hintergrund-ROM tätig ist usw.?

Dies wird in den ROMs selbst festgelegt. Deshalb hier nun die Informationen wie dies erkannt wird.

Um einen reibungslosen Ablauf der gesamten Steuerung von ROMs im Bereich ab &C0000 zu gewährleisten, sind besondere Vereinbarungen zu berücksichtigen. Mit diesen "Vorschriften" wollen wir uns nun etwas beschäftigen.

Prinzipiell gilt, daß ein externes ROM auf Adresse 0 Vorrang hat. Wird der CPC eingeschaltet, so ist als erstes ROM 0 ausgewählt. Befindet sich kein Erweiterungsrom der Adresse 0 im Zugriffsbereich des CPC, dann wird das ROM auf der Platine (On Board Rom) ausgewählt und Basic gestartet.

Dadurch, daß wie schon erwähnt, ein Erweiterungs-ROM der Adresse Null Vorrang hat, kann beispielsweise das Disketten-ROM durch eine sehr geringfügige Hardware-Änderung als Vordergrund-ROM aktiviert werden und bei jedem Einschaltvorgang startet der CPC als reine CP/M-Maschine.

Diese Hardware-Änderung ist lediglich das Durchtrennen der Brücke LK 1 im Floppy-Controller der DDI 1-Diskettenstation des CPC 464. (Beim CPC 664 und 6128 heißen diese Brücken entweder LK 201 oder LK 7.)

Auf dem uns vorliegenden Platinenbild (Bestückungsplan) des CPC 6128 befindet sich diese Brücke vor den Anschlüssen 1 und 14 des IC 212. Da wir nicht wissen, ob es mehrere Platinenversionen bei den 664- und 6128-CPCs gibt, ist dies nur eine Angabe, um versierten "Technikern" weiterzuhelfen. In allen Fällen raten wir aber, falls Ihr CPC beim Einschalten immer als CP/M-Maschine starten soll, diesen Eingriff durch einen Fachmann durchführen zu lassen.

Durch das Auftrennen der genannten Brücke, die nach Masse führt, wird dem Disketten-ROM anstelle der Adresse 7 die Adresse 0 zugewiesen und da externe ROMs Vorrang haben, wird der Basic-Interpreter ausgeblendet und das Disketten-ROM tritt an dessen Stelle. Vordergrund-ROMs müssen beim CPC 464 die Adressen 0 bis 7 haben, beim 664 und 6128 können sie bis 15 gehen.



Vordergrund-ROMs müssen zusammenhängend von ROM-Adresse 1 bzw. 0 aufwärts installiert sein, denn durch Lücken in der Installation werden ROMs nach dieser Lücke nicht mehr "erkannt". Dies trifft für Hintergrund-ROMs nicht zu.

Alle ROMs werden durch einen OUT &DFxx-Befehl initialisiert. Der Wert "xx" ist die Nummer des ROMs, die hardwaremäßig festgelegt wird. Da vermutlich nur relativ wenige CPC-Besitzer in der Lage sind, Erweiterungsplatinen selbst zu bauen, unterlassen wir an dieser Stelle die genauere Beschreibung der Adressdekodierung.

Trotzdem sind die nun nachfolgenden Informationen aber für alle wieder interessant, denn unabhängig davon, ob Sie nun eine Erweiterungsplatine oder einen Schneider mit oder ohne Diskettenlaufwerk haben, gilt das folgende wieder für alle CPCs.

### 8.1 Die ersten fünf Bytes eines Erweiterungs-ROMs

Wie schon bekannt, liegen diese Speicherbausteine dem RAM-Bereich ab &C000 parallel und können maximal je eine Kapazität von 16 KB haben. Die wichtigsten Informationen für den CPC sind in den ersten Bytes enthalten. Hier nun die Details:

&C000	Der ROM-Typ
&C001	Die ROM-Markierungsnummer
&C002	Die ROM-Versions-Nummer
&C003	Die ROM-Modifikations-Nummer
&C004	Beginn der externen Befehlstabelle

Zur besseren Erläuterung hier nun eine Hardcopy der ersten Bytes des Basic-ROMs beim CPC 464.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Status : 253
&C000	80	01	00	00	4C	C0	31	00	C0	CD	CB	BC	CD	C4	F4	DA	...
&C001	00	00	21	00	AC	36	00	06	1B	23	36	C9	10	FB	21	3F	...
&C002	C0	CD	37	C3	AF	32	00	AC	CD	CB	DD	CD	84	CA	CD	97	...
&C003	ED	CD	D3	C0	CD	3E	C1	11	F0	00	CD	08	F7	18	25	20	...
&C004	42	41	53	49	43	20	31	2E	30	0A	0A	00	42	41	53	49	BASIC 1.0...
&C005	C3	00	CD	E1	C2	C0	31	00	C0	CD	9A	E7	CD	63	E1	CD	...
&C006	43	CA	38	54	CD	01	AC	31	00	C0	CD	62	C1	CD	D6	DD	...
&C007	DC	B6	BC	CD	48	BB	CD	86	C3	3A	45	AE	E7	C4	3E	C1	...
&C008	3A	AA	AD	D6	02	20	08	32	AA	AD	CD	DF	CA	EB	30	C6	...
&C009	21	CC	C0	CD	41	C3	CD	CB	DD	3A	1C	AC	B7	28	11	CD	...
&C00A	02	C1	30	C0	7E	B7	28	F1	CD	D2	E6	CD	7A	C1	18	E3	...
&C00B	CD	3B	CA	36	FB	CD	4E	C3	CD	BC	E6	30	05	C4	7A	C1	...
&C00C	18	D4	CD	BB	DE	CD	53	C4	2B	C3	74	DB	52	65	E1	64	...
&C00D	79	0A	00	AF	19	05	22	10	AC	3E	FF	32	1C	AC	C9	11	...
&C00E	0A	00	28	02	FE	2C	C4	E1	CE	D5	11	0A	00	CD	55	DD	...
&C00F	DC	E1	CE	CD	4A	DD	EB	22	1F	AC	E1	CD	D6	C0	C1	C3	...

Seite vor = Cursor rechts      Seite zurueck = Cursor links  
 Zeile vor = Cursor nach unten      Zeile zurueck = Cursor hoch  
 Leertaste = Aenderung der Adresse      \$ = Status      [E] = Ende

Abbildung 13: Speicherauszug des Basic-ROMs

Im ersten Byte erkennen Sie den ROM-Typ, dieser ist Null! Da es sich aber um ein ON-BOARD-ROM handelt, ist Bit 7 gesetzt und dadurch hat diese Speicherstelle den Inhalt &80.

Die ROM-Markierungs-Nummer ist 1. Die ROM-Versions-Nummer 0 und die ROM-Modifikationsnummer ebenfalls. Die nächsten beiden Bytes sind die Zeiger auf die Befehlstabelle dieses ROMs.

(Mit dem Monitor MON1 oder MON2 können Sie sich, wenn Sie wollen, alle Speicherstellen des Basic-ROM ansehen.)

Durch ein kleines Maschinenprogramm kann aufgrund der genauen Festlegungen in der Firmware und durch eine zur Verfügung stehende Betriebssystemroutine abgefragt werden, welcher CPC-Typ vorliegt.

## 8.2 Assemblerlisting: CPC-ART.EDI

```

;*****
;* CPC-ART.EDI *
;*****

org &a000

rsxbind equ &bcd1:textout equ &bb5a:klprom equ &b915

ld bc,rsxtabl ;Erweiterungsbefehl
ld hl,sysram ;4 Bytes RAM fuer System
jp rsxbind ;RSX einbinden

rsxtabl
defw rsxbef1
jp version ;Prg-Beginn Version
jp typaus ;Prg-Beginn Ausgabe
ret
;-----
rsxbef1
defb "VER";Befehlswort festlegen
defb "S"+&80 ;Endbuchst./8.Bit setzen
defb "TY":defb "P"+&80
defb &00 ;Ende der Befehlswoorte
;-----
version
ld c,0 ;Basic-Rom auswaehlen
call klprom ;Klasse und Version feststellen
ld (klasse),a ;Klasse merken
ld (cpcvers),hl ;Version merken
ld b,&e
ld hl,text0 ;Zeiger auf Text 'ROM-Version'
meldung
ld a,(hl) ;Zeichen holen
call textout ;und ausgeben
inc hl ;Zeiger erhoehen
djnz meldung ;weitermachen falls b nicht Null

ld hl,cpcvers ;Adresse in HL
inc hl
ld a,(hl) ;Typ holen
adc &30 ;in Zahl wandeln
call textout ;Zahl ausgeben
ret
;-----
typaus
ld b,5 ;Zeichenanzahl laden

```

```

ld hl, text1      ;HL auf Text 'CPC' stellen

weiter2
ld a, (hl)        ;Zeichen holen
call textout      ;und ausgeben
inc hl            ;Adresse erhoehen
djnz weiter2      ;Falls Zaehler nicht Null weitermachen

ld a, (cpcvers+1) ;0 = 464    1 = 664    2 = 6128
adc a             ;verdoppeln
adc a             ;und nochmal verdoppeln
ld e, a           ;Typzahl merken (Zeiger auf Typname)
ld d, 0           ;zur Addition 0 auf Null
ld hl, text3      ;Beginn der Typennamen
adc hl, de        ;Offset addieren
ld b, &4          ;Zaehler setzen

weiter
ld a, (hl)        ;Zeichen des Typs holen
call textout      ;ausgeben
inc hl            ;Adresse erhoehen
djnz weiter       ;Falls b nicht Null, dann weitermachen
ret              ;Ab -> Basic

```

```

text0 text "ROM-Version : " :text1 text " CPC ":text3 text " 464"
text4 text " 664":text5 text "6128":cpcvers defs 2:klasse defs 1
sysram defs 8
;-----

```

Und hier das zugehörige Hexlader-Listing.

### 8.2.1 Listing: CPC-ART.HEX

```

800 MEMORY &9FFF
870 REM cpc-art.hex
880 a= 40960:e= 41086:zb=2000
890 FOR i =a TO e:READ d$:IF LEFT$(d$,1)<>"&" THEN 920
900 IF ps<>VAL(d$) THEN PRINT"Fehler Zeile "zb:END
910 ps=d$:IF i=e THEN END:ELSE i=i+1:zb=zb+1:GOTO 950
920 d$="&"+d$
930 POKE i, VAL(d$):ps=ps+VAL(d$)
940 PRINT VAL(d$)
950 IF i < e THEN NEXT i
2000 DATA 01,09,A0,21,81,A0,C3,D1,BC,12,A0,C3,1A,A0,C3,3C,&076A
2001 DATA A0,C9,56,45,52,D3,54,59,D0,00,0E,00,CD,15,B9,32,&0681
2002 DATA 80,A0,22,7E,A0,06,0E,21,5F,A0,7E,CD,5A,BB,23,10,&0627
2003 DATA F9,21,7E,A0,23,7E,CE,30,CD,5A,BB,C9,06,05,21,6D,&071B
2004 DATA A0,7E,CD,5A,BB,23,10,F9,3A,7F,A0,8F,8F,5F,16,00,&071B
2005 DATA 21,72,A0,ED,5A,06,04,7E,CD,5A,BB,23,10,F9,C9,52,&072B
2006 DATA 4F,4D,2D,56,65,72,73,69,6F,6E,20,3A,20,20,43,50,&04DC
2007 DATA 43,20,20,34,36,34,20,36,36,34,36,31,32,38,&02B2

```

Das Programm wird durch CALL &A000 aufgerufen. Dadurch werden zwei RSX-Befehle eingebunden, die Ihnen sowohl die Versions-Nummer

des ROMs, als auch den CPC-Typ mitteilen. Die beiden Befehlswoorte sind:

VERS und TYP,

bei denen wie bei allen RSX-Befehlen der senkrechte Strich vorangestellt werden muß.

### 8.3 Machen Sie aus Ihrem CPC 464 einen CPC 664! (Neue Betriebssysteme für die CPC's von Schneider)

Für viele Besitzer des CPC 464 von Schneider ist es immer wieder ein Ärgernis, wenn sie lesen müssen, daß dieses oder jenes Programm nur auf dem CPC 664 oder 6128 läuft. Aber auch umgekehrte Fälle treten häufig auf. Schuld daran ist, daß die Firmware-ROMs aller drei CPC-Typen unterschiedlich sind.

Dabei sind - abgesehen von der höheren Speicherkapazität beim 6128 - die Unterschiede zwischen diesen Rechnern gar nicht so groß. Diese Tatsache kann man sehr leicht dazu ausnutzen, um z.B. auf dem CPC 464 einen 664 oder gar einen 6128 zu simulieren.

Bevor wir aber nun auf das eigentliche Thema eingehen, wollen wir erst einmal feststellen, was denn einen 464 oder einen der Nachfolger ausmacht.

Der CPC 464 hat genausoviel RAM-Speicherplatz wie der 664. Besitzt man zum 464 auch noch die Diskettenstation, so ist der 464 sogar noch "weiter ausgebaut" als der 664 da zusätzlich ja noch der eingebaute Kassettenrecorder vorhanden ist.

Von der Kompaktheit her sind beide ebenfalls identisch. Stellt man nämlich dem 664 noch einen Kassettenrecorder bei, wird in etwa vergleichbarer Platz benötigt wie beim 464 mit einer externen Diskettenstation.

Grundsätzlich kann man also feststellen, daß in der Konfiguration CPC 464 mit zusätzlichem Diskettenlaufwerk im Vergleich zum CPC 664 beide Geräte gleiche Leistungsmerkmale haben.

Auch ein Vergleich mit dem CPC 6128 zeigt, daß dieser lediglich mehr RAM-Speicherplatz (plus ein Ansteuer-IC) hat. Aber ansonsten sind - abgesehen von den Äußerlichkeiten - eigentlich keine großen Unterschiede vorhanden.

Das einzige Ärgernis an dieser Sache ist, daß durch die drei unterschiedlichen Firmware-ROMs keine volle Kompatibilität bei der Software mehr gegeben ist.

Dies soll nun wirklich kein Vorwurf in Richtung der englischen Firma Locomotive Software Ltd. (dem Ersteller der Firmware) oder Amstrad sein, geschweige denn gegenüber der Firma Schneider, nein ganz im Gegenteil.

Die Firmware des 464 enthielt Fehler (z.B.: DEC\$ u.ä.), die man bei den Nachfolgern ausgemerzt hat. Weiterhin wurde ja auch noch

der Befehlssatz erweitert, also durchaus positive Eigenschaften der 464-Nachfolger und auch positives Verhalten der "CPC-Eltern". Nur, die Kompatibilität mußte deshalb eben leiden.

Schön wäre es aber gewesen, wenn man den 464-Besitzern zumindest die Möglichkeit eröffnet hätte, ihren CPC umzurüsten, denn wie wir zeigen werden, ist dies überhaupt kein Problem!

### 8.4 Wie macht man aus einem 464 einen 664?

Da die wirklichen Unterschiede, wie wir eben schon angeführt haben, sehr gering sind, ist es theoretisch (und auch praktisch) überhaupt kein Aufwand, aus einem 464 einen 664 zu machen.

Alles was zu erfolgen hat, ist der Austausch des Firmware-ROMs. Allerdings vermuten wir, daß es sehr schwierig sein wird, dieses ROM von der Firma Schneider zu erhalten.

Besser ist es wahrscheinlich, zu versuchen, bei einem Händler das ROM als "Ersatzteil" zu bekommen. Der Überglückliche, dem dies gelingt, hat dann aber bereits wieder das Problem, daß manche Software für den 464 wieder nicht läuft.

Aber auch hier gibt es eine Lösung. Man kann die beiden Betriebssystem-ROMs "Huckepack" löten und dann die Chipauswahlsignale so über einen Schalter "umlegen", daß man zwischen den beiden CPC-Verhaltensweisen wählen kann. Der Umschaltvorgang darf aber nur bei ausgeschaltetem Computer erfolgen.

Nur ist dies ein Eingriff in das Gerät selbst, der nicht jedermanns Sache ist. Bitte haben Sie aber noch ein paar Zeilen Geduld, bis Sie eine andere mögliche Lösung präsentiert bekommen.

Mit Sicherheit wird ein Teil der Leser auch Interesse daran haben, das eingebaute Betriebssystem selbst zu ändern, eigene Routinen einzubauen, usw.

Dies gilt aber bestimmt nicht nur für die Besitzer eines 464, sondern auch für die anderen "Schneider", die ohne großen Aufwand diese Eingriffe in die Firmware durchführen möchten, ohne jedesmal gleich den Computer öffnen zu müssen. Wie dies geschehen kann soll nun aufgezeigt werden.

### 8.5 Wo liegen externe ROMs ?

Alle Schneider CPC's haben ein Firmware-ROM mit einer Kapazität von 32 KByte. Diese 32 KB werden in einen unteren (Betriebs-System) und oberen Teil (Basic-Interpreter) gesplittet.

Die ersten 16 KB liegen im Bereich von &0000 bis &3FFF dem RAM-Speicher parallel und die zweiten 16 KB ab &C000 bis &FFFF.

Letztere liegen also parallel zum - normalerweise - ausgewählten Bildschirm-RAM.

Alle wichtigen Leitungen der Schneider-Computer sind von außen her zugänglich. Dies betrifft nicht nur die Daten- und Adressleitungen, sondern auch Leitungen, die für externe "Geräte" zuständig sind. Als externe Geräte sind damit nicht nur der Drucker oder Diskettenlaufwerke gemeint, sondern auch andere nützliche Erweiterungen für die CPCs.

Wie wir Ihnen schon mitgeteilt haben, und wie Sie auch dem Handbuch entnehmen können, sind die CPC's in der Lage bis zu 252 externe ROMs anzusprechen.

Im Regelfall handelt es sich um 16 KB-ROMs, die dem Bereich ab &C000 parallel liegen.

So "arbeitet" dort (also parallel zum Basic-Interpreter-ROM) der Bildschirm, oder Erweiterungen wie MAXAM, PROTEXT, PROGRAMMERS TOOLKIT und all die anderen schönen Utilitie-ROMs. Aber auch das "Disketten-ROM" ist dort zu finden.

In keiner uns bis dato bekannten Veröffentlichung aber steht klar geschrieben, (warum eigentlich nicht ?) daß das gesamte Firmware-ROM von außen komplett ausgeblendet und ein externes ROM an dessen Stelle eingeblendet werden kann und vor allem nicht, wie dies geht. Dabei ist es ohne große Kunstgriffe möglich und bietet nun speziell für 464-Besitzer die Möglichkeit, von außen aus einem CPC 464 einen 664 zu machen. Die Zeitschrift MC unseres Verlages hat bereits in der Juni-Nummer 1986 diese Platine veröffentlicht, da wir einem möglichst großen Kreis diese wirklich sehr einfache Möglichkeit aufzeigen wollten.

## 8.6 Wir "blenden" ein und aus

Am Pin 42 des Expansions-Anschlusses (beim CPC 464 =Diskettenanschluß) ist das Signal ROMEN (negiert) und am Pin 43 das Signal ROMDIS herausgeführt.

Die beiden Signalpegel an diesen Anschlüssen bestimmen, welches ROM oder EPROM gerade immer aktiv ist. Das Signal ROMEN kommt vom Gate-Array und liegt intern auch am Pin 20 des Firmware-ROMs.

Dieser Anschluß ist der Chip-select (Auswahl) für das Firmware-ROM. Dieses Rom hat aber auch noch einen weiteren Anschluß: OE (negiert). Ein dort anliegendes Signal bestimmt, ob der Ausgang des ROMs hochohmig (Zustand "Z" = High Impedance) oder "niederohmig" ist.

Nur durch diese Eigenschaften, daß Ausgänge hochohmig werden können, ist das parallele Arbeiten mehrerer ROMs oder parallel liegender RAMs überhaupt möglich.

Diese Technik wird beispielsweise auch beim "Einklinken" der Diskettenstation benutzt.

Zurück zum Anschluß OE des Firmware-ROMs. Dieser liegt über einen 2,2-kOhm-Widerstand auf "Null". An diesen Eingang führt die Leitung ROMDIS vom herausgeführten Expansionsanschluß.

Ist eine Diskettenstation angeschlossen, dann wird von dieser, immer dann wenn die Diskettenstation "aktiv" ist, nicht nur der obere Teil, sondern das komplette Firmware-ROM "weggeschaltet".

Das heißt ganz simpel ausgedrückt und ohne nun wirklich in's Detail gehen zu müssen, daß die Diskettenstation die Firmware ein- und ausschalten kann. Gleiches kann natürlich auch künstlich herbeigeführt werden.

Legt man an den Pin 43 des Platinenanschlusses positiven Pegel, so ist das Firmware-ROM inaktiv. Aktiviert man nun gleichzeitig ein externes ROM oder EPROM, so kann es an dessen Stelle treten. Dabei ist nun aber zu beachten, daß dann, wenn der Anschluß an Pin 43 konstant auf Plus gelegt wird, und diese Leitung dann zu anderen Erweiterungen geführt wird, diese ebenfalls inaktiv werden.

Ein z. B. dem internen ROM parallel angeschlossenes 664-ROM würde deshalb zwar bewirken, daß sich ein CPC 464 als CPC 664 meldet, ein Diskettenbetrieb wäre aber nicht mehr möglich. Die Informationsleitung, daß das Disketten-ROM nun arbeiten soll und deswegen das Firmware-ROM "weg" muß, liegt ja nun auf Pluspegel und die Firmware "erkennt" die Diskettenstation oder andere Erweiterungen nicht mehr.

Deshalb darf diese Anschluß-Leitung nicht einfach zur Diskettenstation durchgeführt werden. Die Leitung an Anschluß 43 des Diskettencontrollers muß also an das externe ROM geführt werden und kann dieses dann "schalten".

Nun sind fast alle Punkte, die ein externes Anschließen von "neuer" Firmware ermöglichen, angesprochen. Wenn Sie also eine entsprechende "verschaltete" Platine an Ihren CPC anschließen, dann wird das interne ROM ignoriert. Um nun aber wahlweise mit dem einen oder anderen Betriebssystem arbeiten zu können, muß lediglich eine entsprechende Umschaltmöglichkeit vorgesehen werden.

### Einfacher gehts kaum

Die im Anhang 3 dieses Buches abgedruckte Platine bietet diese Möglichkeit. Wir möchten aber gleich darauf hinweisen, daß man zwar eine elektronisch "elegantere" Weise wählen kann, diese Platine aber ohne (mit Ausnahme des ROMs oder EPROMs) aktive Bauelemente arbeitet. Einfacher geht es ohne Geräteeingriff wahrscheinlich nicht mehr.

Diese Platine kann natürlich mittels eines Eins zu Eins-Adapters auch an den CPC 6128 angeschlossen werden, um den 464 zu simulieren, nur sollte, wenn die Diskettenstation betrieben werden soll, doch ein Eingriff in den CPC erfolgen, denn das Disketten-ROM ist ja intern "verdrahtet". Gleiches gilt auch für den CPC 664.

Vielleicht als interessante Anmerkung erwähnt: Das erste, noch "wilde, handgestrickte" Freiluftverdrahtungs-Muster dieser Erwei-

terungsplatine wurde auch am CPC 6128 betrieben. Auf der Platine befand sich als Firmware das Betriebssystem des 664.

Zwangsläufig wurde durch die Platine das interne Disketten-ROM abgeschaltet. An die Platine wurde dann die eigentlich zum CPC 464 gehörende Diskettenstation (mit Controller) des 464 angeschlossen und daraufhin, konnte mit dem im 6128 eingebauten Laufwerk gearbeitet werden! Dies hat zwar kaum einen praktischen Wert, sollte aber aufzeigen, was mit der Platine so möglich ist.

Besitzer eines 664 oder 6128 müssen also die "Leitung 43" der Diskettenstation innen (!) auftrennen und nach außen zum externen ROM auf der Erweiterungsplatine führen, dann kann auch mit diesen Rechnern ein anderer CPC-Typ voll simuliert werden. Ist eine "normale" ROM-Erweiterungsplatine vorhanden, geht es auch ohne Auftrennen, wenn in einem der freien Steckplätze ein Disketten-EPROM steckt. Auch Besitzer der VORTEX-X-Laufwerke können mit VDOS arbeiten (Amsdos geht nicht)!

Zum Abschluß noch einige Punkte die sehr wichtig sind. Da ein EPROM physikalisch anders aufgebaut ist als ein ROM und da diese wirklich sehr einfache Platine ohne große Beschaltung auskommt, können evtl. nicht gewünschte Effekte auftreten. Dies äußert sich dadurch, daß der Schneider-Computer zwar mit dem neuen Betriebssystem "durchstartet", dann aber irgendwelche undefinierbare Zeichen auf dem Bildschirm erscheinen, oder aber daß die Schriftzeichen selbst nicht sauber sind.

So zeigte sich während der mehrmonatigen Testphase, daß die Platine - mit einem EPROM bestückt - am 464 ohne angeschlossene Diskettenstation nicht korrekt arbeitete. War aber die Diskettenstation angeschlossen, war alles in Ordnung. Wurde die Platine mit einem ROM bestückt, trat dieser Effekt ebenfalls nicht mehr auf. Mit einem ROM konnte der 464 also auch ohne Diskettenstation als 664 betrieben werden.

Der gleiche Effekt zeigte sich auch, wenn die Platine am 6128 mit EPROM betrieben wurde und kein Disk-Controller angeschlossen war. Wurde dieser aber angeschlossen, oder die Platine mit einem ROM bestückt, dann waren diese Effekte auch beim 6128 nicht zu bemerken. Bei Versuchen mit anderen zusätzlichen Erweiterungsplatten war in allen Fällen ein einwandfreier Betrieb möglich. Da wir aber ja keine Zusatzhardware für die CPC's verkaufen wollen und auch den Aufwand so gering wie möglich halten wollten, haben wir gegen dieses Fehlverhalten nichts unternommen, denn es kann ja durch "die Diskettenstation" abgestellt werden. Und ein 664 oder 6128 hat ja eine Diskettenstation!

Daß ein CPC 464 oder 664 durch diese Platine nicht zu einem echten 6128 mit 128 KB-RAM-Speicher wird, leuchtet ja jedem ein, aber so manche 6128-Software kann darauf dann verarbeitet werden.

Ein letzter Satz noch zu den EPROMs. Diese sollten unbedingt 200 ns schnell sein, sonst könnte es ebenfalls Probleme geben!

Viel Spass mit Ihren "Multi-CPC's".



## 9. Rund um die Diskette

Bevor wir nun detaillierter auf die Diskette eingehen, sollten Sie den Monitor aus Kapitel 1 erweitern, denn wir wollen ja, daß Sie auch das folgende nun nachvollziehen können.

Grundlage der Erweiterung ist die Version MON2.BAS, also das um die Zeile 145 erweiterte und in der Zeile 100 geänderte Programm MON1.BAS!

Wir hoffen, daß Sie die Zeilennummern des Basic-Listings noch genauso haben, wie im Buch abgedruckt, denn ansonsten gibt es Schwierigkeiten mit dem MERGE-Vorgang.

Als erstes sollten Sie das nun folgende kleine Programm in den Basicteil von MON2.BAS "einmischen".

### 9.1 Listing: MONERW.BAS

```
100 'mon3.bas
110 MODE 2:MEMORY &8FFF
120 IF PEEK(&A000) <>&C3 THEN LOAD"mon1.bin",&A000
121 IF PEEK(&A16B) <>&FE THEN LOAD"HC.bin",&A16B
122 IF PEEK(&A1E8) <>&87 THEN LOAD"secread.bin",&A1E8
123 PRINT"MON3 + SECREAD
124 INPUT "Track ";track
125 INPUT "Formse";formse;formse=VAL("&"+formse)
127 drive=0:buffer=&9000:befehl=&84
130 status = 255
140 adresse=buffer
145 CALL &A1E8,drive,track,formse,buffer,befehl
150 CLS
265 PRINT" d = hardcopy";
270 PRINT TAB(68)CHR$(24);"e = Neustart";CHR$(24)
320 IF LOWER$(x$)="e" THEN RUN
340 IF LOWER$(x$)="d" THEN CALL &A16B,90,360:GOTO 300
```

Wie dieses "Einmischen" zu erfolgen hat, dürfte Ihnen zwar aufgrund anderer MERGE-Beispiele (in den vorausgegangenen Kapiteln) schon bekannt sein, aber trotzdem nochmals die Vorgehensweise, falls diese Zeilen von Ihnen nicht direkt in das Programm eingegeben werden.

- 1.) MONERW.BAS abtippen und als ASCII-FILE abspeichern (Save "MONERW.BAS",A)

- 2.) MON2.BAS laden  
(Load "MON2.BAS")
- 3.) MONERW.BAS hinzumischen  
(Merge "MONERW.BAS")
- 4.) MON3.BAS abspeichern  
(Save "MON3.BAS")

Dadurch haben Sie den Basisteil des erweiterten Monitors abgespeichert, nun fehlt noch der Maschinencodeteil, dessen Erklärung (kommentiertes Assembler-Listing) allerdings erst etwas später erfolgt und zwar bei der Beschreibung des Programmes SECREAD.EDI.

## 9.2 Listing: SECREAD.HEX

```

400 MEMORY &A000
410 GOSUB 450
420 SAVE "SECREAD.BIN",B,&A1E8,58
430 END
450 REM SECREAD.HEX
460 a= 41448:e= 41506:zb=2000
470 FOR i =a TO e:READ d$:IF LEFT$(d$,1)<>"&" THEN 500
480 IF ps<>VAL(d$) THEN PRINT"Fehler Zeile "zb:END
490 ps=0:d$="":IF i=e THEN RETURN:ELSE i=i-1:zb=zb+1:GOTO 530
500 d$="&"&d$
510 POKE i,VAL(d$):ps=ps+VAL(d$)
520 '
530 IF i < e THEN NEXT i
2000 DATA 87,32,22,A2,DD,22,24,A2,ED,4B,22,A2,2A,24,A2,11,&063F
2001 DATA 26,A2,ED,B0,3A,2E,A2,32,2E,A2,21,26,A2,CD,D4,BC,&07B7
2002 DATA 22,30,A2,79,32,32,A2,21,2E,A2,5E,21,2C,A2,56,21,&0528
2003 DATA 2A,A2,4E,2A,28,A2,DF,30,A2,C9,&0488

```

Haben Sie nun auch dieses Programm abgeschrieben und laufen lassen, dann steht Ihnen für den Monitor und auch für andere Programme dieses Binärfile zur Verfügung.

Heben Sie sich diesen Hexlader für später noch auf, dann brauchen Sie beim Programm SECREAD.BAS weniger zu schreiben!

Fall Sie die Maschinenspracheteile auf eine andere Diskette übertragen wollen, nennen wir Ihnen nun nochmals (zusammengefaßt) die Adressen und Längen der für den Monitor (MON3.BAS) erforderlichen Teile.

Programmname	Startadresse	Länge
MON1.BIN	&A000	363
HC.BIN	&A16B	124
SECREAD.BIN	&A1E8	58

Mit diesen Angaben können Sie nun, falls gewünscht, diese Binärfiles auch auf andere Disketten oder Kassetten abspeichern, ohne immer den jeweiligen Hexlader benutzen zu müssen.

Nehmen wir an, daß Sie das Maschinenprogramm HC.BIN von einer Diskette auf eine andere Übertragen wollen. Dann können Sie beispielsweise folgendermaßen vorgehen:

1. Speicher durch MEMORY ADRESSE auf einen erforderlichen Wert begrenzen.
2. Diskette mit diesem File einlegen.
3. Laden mit LOAD"HC.BIN",HIMEM+1
4. Diskette wechseln
5. SAVE "HC.BIN",B,HIMEM+1,LÄNGE
6. Fertig!

Vielleicht ist Ihnen nun aufgefallen, daß wir nicht die Startadresse mit den in der Aufstellung angegebenen Werten benutzt haben. Der Grund ist ganz einfach, Sie können diese Teile nämlich auch in andere Speicherbereiche einladen und wieder zurückschreiben. Solange das Programm nicht an diesem Platz arbeiten soll, macht dies nichts.

Genau aus diesem Grunde geben wir bei einem Großteil unserer Programme prinzipiell immer die Ladeadresse mit an, denn dadurch kommt ein Maschinenprogramm, das aus einem anderen Speicherbereich geschrieben wurde immer an die richtige Adresse.

Wenn Sie in obigen Beispiel MEMORY auf &5FFF setzen, dann wird zwar das Programm ab &60000 eingelesen und auch ab dieser Adresse wieder gespeichert, durch den Ladebefehl des zugehörigen Basicprogrammes aber an die korrekte Adresse gebracht!

Wir setzen nun voraus, daß Sie das Basic-Programm MON3.BAS und die Maschinencodeteile auf einer Diskette haben. Wenn Sie nun das Programm starten, sieht es in den jeweiligen Anfangsadressen der MC-Routinen nach, ob dort das erste Byte des Programmes vorhanden ist. Falls nicht, dann wird der entsprechende Teil nachgeladen. Andernfalls macht das Programm gleich weiter.

Was durch das Programm MON1 geschieht, wissen Sie ja bereits: Ein Speicherauszug wird ausgegeben. Hinzu kam durch die Modifikation zu MON2 dann die Möglichkeit der Hardcopy.

Durch die Ergänzungen zu MON3 können Sie nun einzelne Disketten-sektoren in den Speicher des CPC einlesen und diesen Bufferbereich als Dump ausgeben und falls von Ihnen gewünscht, auch Hardcopies drucken lassen.

Damit erfährt der Monitor eine weitere Leistungssteigerung und kann Ihnen bei Arbeiten mit einer Diskettenstation sehr viel weiterhelfen.

### 9.3 Die Diskette

Ihr Handbuch zur Diskettenstation bzw. Ihr CPC-Handbuch enthält die grundlegenden Informationen über dieses Speichermedium und seiner Handhabung. Wir gehen in diesem Kapitel wesentlich weiter und zeigen Ihnen Dinge auf, die Ihnen Möglichkeiten eröffnen, auf Ihren Disketten zu schalten und zu warten, wie Sie nur wollen.

Die Programme in diesem Kapitel zeigen und öffnen Ihnen die Wege, Ihre Disketten wirklich zu beherrschen. Aus diesem Grunde sollten die Grundkenntnisse bereits bekannt sein!

Grundsätzlich gelten die nun folgenden Ausführungen für das Diskettenbetriebssystem AMSDOS (Amstrad Disk Operating System) und gelten auch (oder sinngemäß) sowohl für Arbeiten unter CP/M als auch unter Basic. Von beiden wird die gleiche File-Struktur benutzt und deshalb ist es möglich, die gleichen Disketten sowohl unter AMSDOS, als auch unter CP/M zu benutzen.  
(Das CP/M-BIOS befindet sich im Disketten-ROM.)

Vermutlich werden auch zukünftige Programmiersprachen, die es vielleicht einmal für die Schneider CPCs geben wird, diesen "Diskettenstandard" in wohl gleicher Weise übernehmen, um kompatibel zu bleiben.

Ebenso werden sich wahrscheinlich die Hersteller von Zusatz-Disketten-Stationen (mit eigenem Controller) an diese Standards anlehnen.

Das Prinzip ist immer das gleiche und nur Details können unterschiedlich sein. Beispielsweise ist die Anzahl der Inhaltsverzeichnis-einträge (Directory) beim VDOS2 der Vortex-Laufwerke größer als das der "Original"-Stationen und umfaßt anstelle von 64 Eintragsmöglichkeiten 128. Auch die Sektornummern (Formatkennzeichen) sind andere, aber wie erwähnt, das Prinzip ist gleich!

Wenn sich die Hersteller von "Fremdlaufwerken" an diese Standards gehalten haben, ist es meist sehr einfach (falls überhaupt erforderlich) die in diesem Buch enthaltenen Programme so umzuschreiben, daß sie auch in Verbindung mit diesen Laufwerken arbeiten.

Die Maschinenprogramme für Diskettenzugriffe sind alle so geschrieben, daß die Adressen der Befehlsroutinen durch Aufruf der Betriebssystemroutine KL FIND COMMAND gesucht werden und dadurch auch z.B. mit den Vortex-Laufwerken benutzt werden können.

Die erforderlichen Anpassungen sind aufgrund der kommentierten Assemblerlistings leicht durchzuführen. Bei einem Formatierprogramm für ein 80-Spur-Laufwerk muß eben die Spüranzahl von 40 auf 80 erweitert und bei Doppelkopflaufwerken eben auch berücksichtigt werden, daß die zweite Diskettenseite "behandelt" wird.

Wie Sie wissen, muß eine Diskette, damit sie benutzt werden kann, erst einmal formatiert werden. Dieser Vorgang bewirkt, daß die Diskette in Spuren und Sektoren eingeteilt wird. Je nach Wunsch, erfolgt diese Einteilung im System-, Daten- oder IBM-Format.

Und nun können Sie sich mittels des Programmes MON3 alles selbst ansehen. Wir gehen davon aus, daß dieses Programm auf einer DATA-ONLY-Diskette steht!

Starten Sie dieses Programm und geben Sie bei der Frage nach der Spur (=Track) den Wert "0" ein.

Die Frage nach dem Format und dem einzulesenden Sektor (formse) beantworten Sie bitte mit "C1".

Nach einigen Augenblicken sehen Sie einen Speicherauszug, der in etwa folgendes Bild hat:

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Status : 255
9000	00	4C	48	45	53	45	20	20	20	42	41	53	00	00	00	06	•LIESE BAS....
9010	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
9020	00	41	53	43	2D	48	45	58	50	42	41	53	00	00	00	00	•ASC-HEXPBAS....
9030	03	04	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
9040	00	44	41	54	41	47	45	4E	20	42	41	53	00	00	00	13	•DATAGEN BAS....
9050	05	06	07	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
9060	00	56	4F	52	42	45	4C	45	20	42	41	53	00	00	00	07	•VORBELE BAS....
9070	08	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
9080	00	44	41	54	45	49	32	20	20	20	20	20	00	00	00	01	•DATEI2 ....
9090	09	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
90A0	00	49	4E	48	41	4C	54	20	20	42	41	53	00	00	00	11	•INHALT BAS....
90B0	0C	00	0E	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
90C0	00	4D	48	4E	49	4D	4F	4E	32	42	41	53	00	00	00	16	•MINIMON2BAS....
90D0	0F	10	11	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
90E0	05	44	41	54	45	49	31	20	20	42	41	4E	00	00	00	01	•DATEI1 BAK....
90F0	1E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Seite vor = Cursor rechts      Seite zurueck = Cursor links  
 Zeile vor = Cursor nach unten      Zeile zurueck = Cursor hoch  
 Leertaste = Aenderung der Adresse      S = Status    d = hardcopy      e = Neustart

Abbildung 14: Der erste Teil des eingelesenen Sektors

Was Sie nun erkennen können, ist ein Teil des Inhaltsverzeichnisses der eingelegten Diskette.

Bei den ASCII-Zeichen erkennen Sie die Programmnamen. Bedenken Sie hierbei nun bitte, daß nur die Hälfte des eingelesenen Sektors dargestellt wird. Den zweiten Teil erhalten Sie durch Betätigung der Taste CURSOR-rechts.

Die Disketten des "Schneider-Formates" werden mit insgesamt 40 Spuren formatiert. Diese Spuren (tracks) bestehen aus acht (IBM-Format) oder neun (Data-Only- und Systemformat) Sektoren (sectors). Wenn Sie nun nachrechnen, stellen Sie fest, eine Diskette im IBM-Format hat  $40 \times 8 = 320$  Sektoren, in den anderen Formaten  $40 \times 9 = 360$  Sektoren.

Ein Sektor kann 512 Bytes an Daten aufnehmen. Dies wiederum bedeutet, daß eine "Systemdiskette" eine Kapazität von  $40 \times 9 \times 512 (= 184320)$  Bytes besitzt.

Vergleichen Sie diesen Wert mit der ausgegebenen Kapazitätsangabe bei einer frisch formatierten Diskette im Systemformat, so fällt Ihnen auf, daß diese Werte nicht übereinstimmen. Durch den Befehl CAT erhalten Sie die Ausgabemeldung:

```
Drive A: user Ø
169K free
```

Dies kommt daher, daß bei einer Systemdiskette die ersten beiden Spuren durch CP/M verwendet werden.

Wenn Sie nun eine neue Berechnung durchführen, dann ist immer noch eine Differenz vorhanden, denn auch das Inhaltsverzeichnis (Directory) belegt Sektoren.

Wie Sie wissen, sind pro Diskettenseite maximal 64 Einträge im Inhaltsverzeichnis möglich.

Wenn Sie sich nun einmal die Abbildung 14 näher ansehen, können Sie feststellen, daß jeder "Namenseintrag" im Inhaltsverzeichnis 32 Bytes belegt und nun können Sie sich ausrechnen, wieviel Speicherplatz Ihnen durch das Inhaltsverzeichnis "verloren" geht.

Nämlich 64 (Einträge) mal 32 Bytes = 2048 Bytes = 2 KB.

Dieser Wert entspricht exakt 4 Sektoren. Da bei Systemdisketten die ersten beiden Spuren für CP/M reserviert sind, finden Sie das Inhaltsverzeichnis dieser Disketten in der Spur 2 und zwar in den Sektoren 1 bis 4.

Die Spuren werden von 0 bis 39 und die Sektoren von 1 bis 9 (1 bis 8 bei IBM-Format) gezählt.

Deshalb ergeben sich (wir lassen das IBM-Format nun bei allen weiteren Betrachtungen weg, weil kaum jemand damit arbeitet) bei neu formatierten Disketten folgende Kapazitätsangaben:

System	169 KB	=	$((38 \times 9 \times 512 - 4 \times 512) / 1024)$
Data-Only	178 KB	=	$(40 \times 9 \times 512 - 4 \times 512) / 1024$

Dies zeigt, daß bei einer Data-Only-Diskette mehr Platz für Programme zur Verfügung steht, deshalb sollte jeder (vor allem auch aus Kostengründen) nur die Disketten im Systemformat formatieren, die er auch wirklich benötigt.

Wie Sie sich erinnern, haben Sie bei der Frage nach der Spur die Zahl Null eingegeben, das war, wenn Sie eine Datendiskette eingelegt hatten auch richtig, denn dort beginnt das Inhaltsverzeichnis, das wir uns anschließend näher ansehen wollen. Doch zuvor noch eine Erklärung zur Eingabe bei der Frage nach dem Sektor. Hier hatten Sie mit C1 geantwortet.

Die Erklärung ist einfach. Die Sektoren werden je nach Format mit einem unterschiedlichen Kennzeichen versehen. Diese Kennzeichnung wird benötigt, damit die einzelnen Sektoren nicht nur gefunden, sondern auch bearbeitet werden können.

Bei Data-Only ist das Format "Cx" und bei Systemdisketten "4x". Der Wert x stellt die Sektornummer dar.

Üblicherweise haben Sektoren also Werte zwischen 41 und 49 oder C1 und C9. Diese Kennzeichnung ist jedem Sektor auf der Diskette in einem "Vorspann" (Header) vorangestellt.

Auf diesen Header wollen wir nicht näher eingehen, da Sie mit normalen Programmen keinerlei Zugriffsmöglichkeiten haben.

Wichtig ist nur, daß Sie wissen, daß vor jedem Sektor ein Header ist, der eine entsprechende Kennzeichnung aufweist. Warum dies für Sie wichtig ist, erfahren Sie beim Programm WRONGSEC!

Wenn Sie nun den zweiten Sektor einlesen wollen, dann geben Sie bitte bei der Frage nach der Sektornummer (Formse) den Wert "C2" ein, usw.

Sehen wir uns aber nun einmal die Abbildung 14 etwas genauer an. Das erste Byte enthält eine "Null". Dieses Byte ist die "Usernummer" (Falls Ihnen der Begriff "Usernummer" nicht geläufig ist, dann sehen Sie bitte im Handbuch nach.)

In den folgenden elf Bytes steht der Name des ersten Programmeintrages, nämlich der eigentliche Programmname und unmittelbar dahinter die sogenannte Extension, also das Programmartkennzeichen.

Das 13. Byte ist die Erweiterungsnummer. Jeder File-Eintrag im Inhaltsverzeichnis kann nämlich nur Dateien mit maximal 16 KB Länge behandeln.

Da also ein Eintrag nur 16 KB File-Länge "fassen" kann, muß bei längeren Programmen ein weiterer Eintrag erfolgen.

Beinhaltet dieses Byte also eine 0, dann ist dies der erste Eintrag mit diesem Namen. Steht dort eine 1, dann ist es der zweite Teil usw.

Die beiden folgenden Bytes sind DOS-Arbeitsvariable.

Das letzte Byte der ersten Zeile gibt an, wieviele Records der Eintrag hat. Ein Record hat eine Länge von 128 Bytes. Da CP/M bei der Entwicklung sehr gut durchdacht wurde, kann auf verschiedenen Systemen mit unterschiedlichen Sektorlängen gearbeitet werden. Die Basis des ganzen aber sind die Records. Jeder Produzent von Diskettenstationen mit eigenem Diskettencontroller muß beim DOS selbst dafür Sorge tragen, daß seine Sektorlängen aus vielfachen Records bestehen. Beim Schneider-Format (AMSDOS) besteht ein Sektor aus vier Records.

Wenn also mehrere Einträge für ein Datenfile existieren, dann ist beim ersten Eintrag in diesem Byte der Wert &80 zu finden. Nun können Sie nachrechnen ( $&80 * 128 = 16384 = 16 \text{ KB!}$ ) und das ist der Grund, weshalb ein Fileeintrag nur 16 KB "umfassen" kann.

Um Sie nun nicht zu sehr zu verwirren, noch ein paar Punkte zur Klarstellung. Die kleinste Einheit, die von AMSDOS und von auf

den auf den CPCs installierten CP/Ms direkt bearbeitet werden kann, ist 1 Sektor und dieser hat 512 Bytes.

Bei Speichervorgängen unter Basic wird aber immer ein sogenannter Block geschrieben und dieser besteht aus 2 Sektoren!

Daher kommt es daß ein Basic-Programm und mag es noch so kurz sein, auf einer Diskette immer mindestens 1 KB belegt!

Wir haben es also mit vier Einheiten zu tun:

Record	(128 Byte)	
Sektor	(4 Records	= 512 Bytes)
Block	(2 Sektoren	= 1024 Bytes)

Bei Zugriffen durch ein Maschinenprogramm, kann unter Normalbedingungen immer nur ein Sektor direkt bearbeitet werden. Durch Verändern von Diskettenparametern aber, sind auch andere "Einheiten" möglich.

#### 9.4 Das Formatieren

Bei der Formatierung wird auf die Diskette ein magnetisches Muster, das vom gewünschten Format abhängt, "aufgebracht".

Änderungen an diesem Format können meist nur derart erfolgen, daß die entsprechenden Inhalte der Spuren verlorengehen.

Normalerweise können durch die üblichen Formatier-Routinen entweder nur einzelne Spuren, oder aber die komplette Diskette formatiert werden.

Es gibt aber durch "direkte" Zugriffe auf die Diskette auch Möglichkeiten, einzelne Sektoren einer Spur derart zu verändern, daß diese auch unterschiedliche Formate haben.

Derartige Methoden werden zum Softwareschutz eingesetzt und deswegen wollen wir hier nicht näher darauf eingehen, denn Anleitungen für Raubkopierer wollen wir mit diesem Buch nicht geben. (Es ist schon schwer, Ihnen auf der einen Seite nützliche "Utilities" in die Hand geben zu wollen, andererseits aber auch zu versuchen nicht Raubkopierer zu unterstützen!)

Wir wollten nur darauf hinweisen, daß derartige Dinge möglich sind. Genauso ist es möglich, z. B. die Sektoren mit "falschen" Sektornummern zu versehen und in diesen Sektoren dann "Prüfmechanismen" zu verstecken. Darüber später noch mehr.

Leider kann - durch die mitgelieferte Software - eine Diskettenformatierung nur unter CP/M erfolgen.

Dies bedeutet in der Praxis, daß die zu verwendenden Disketten schon formatiert sein müssen, wenn Sie von einem Programm aus benutzt werden sollen, oder es muß erst CP/M gestartet, dann das Formatierprogramm aufgerufen und danach wieder das Basicprogramm neu gestartet werden usw.



Damit diese umständliche Prozedur aber für Basicprogrammierer etwas leichter wird und damit beispielsweise auch von einem Dateiverwaltungsprogramm aus, eine Formatierung durchgeführt werden kann, hier ein Formatierprogramm in Maschinensprache, das von Basic aus aufgerufen werden kann.

#### 9.4.1 Listing: FORMAT.HEX

```

800 MEMORY &9FFF
810 'Starttrack = &aia6           Endtrack = &aia7
820 'Datatable = &aia9           Systemtable = &aib3
830 'Laufwerk-Nummer = &aia8
840 '
870 REM FORMAT.HEX
880 a= 40960:e= 41405:zb=2000
890 FOR i = a TO e:READ d$:IF LEFT$(d$,1)<>"&" THEN 920
900 IF ps<>VAL(d$) THEN PRINT"Fehler Zeile "zb:END
910 ps=d$:d$="":IF i=e THEN END:ELSE i=i+1:zb=zb+1:GOTO 950
920 d$="&"&d$
930 POKE i,VAL(d$):ps=ps+VAL(d$)
940 PRINT VAL(d$)
950 IF i < e THEN NEXT i
2000 DATA 3A,A6,A1,32,73,A1,3A,A7,A1,32,91,A1,3A,A8,A1,32,&0762
2001 DATA 7C,A1,CD,D2,A0,3E,09,47,21,25,A0,3E,00,77,23,23,&05CB
2002 DATA 23,23,10,F9,C9,00,00,41,02,00,00,43,02,00,00,45,&02E5
2003 DATA 02,00,00,47,02,00,00,49,02,00,00,42,02,00,00,44,&011E
2004 DATA 02,00,00,46,02,00,00,48,02,20,2A,2A,20,20,46,6F,&01FD
2005 DATA 72,6D,61,74,69,65,72,70,72,6F,67,72,61,6D,6D,20,&0679
2006 DATA 66,75,65,72,20,43,50,43,20,34,36,34,20,2E,2E,2E,&0410
2007 DATA 2E,20,36,31,32,38,20,2A,2A,20,00,53,79,73,74,65,&03CB
2008 DATA 6D,2D,20,20,6F,64,65,72,20,20,20,44,61,74,61,2D,&048B
2009 DATA 6F,6E,6C,79,2D,46,6F,72,6D,61,74,20,3F,00,42,69,&0562
2010 DATA 74,74,65,20,44,69,73,6B,65,74,74,65,20,65,69,6E,&0606
2011 DATA 6C,65,67,65,6E,20,75,6E,64,20,65,69,6E,65,20,54,&05A7
2012 DATA 61,73,74,65,20,64,72,75,65,63,6B,65,6E,00,27,42,&0587
2013 DATA C0,07,21,A5,A1,CD,D4,BC,D0,22,CF,A0,79,32,D1,A0,&0908
2014 DATA 3E,02,CD,0E,BC,CD,9C,BB,21,49,A0,7E,FE,00,CA,F8,&0843
2015 DATA A0,CD,5A,BB,23,C3,EB,A0,CD,9C,BB,21,03,01,CD,75,&087E
2016 DATA BB,21,7B,A0,7E,FE,00,CA,11,A1,CD,5A,BB,23,C3,04,&07BB
2017 DATA A1,CD,06,BB,FE,73,CA,21,A1,FE,64,CA,36,A1,C3,11,&0903
2018 DATA A1,01,B3,A1,21,27,A0,0A,FE,FF,CA,4B,A1,77,23,23,&0758
2019 DATA 23,23,03,C3,27,A1,01,A9,A1,21,27,A0,0A,FE,FF,CA,&06D8
2020 DATA 4B,A1,77,23,23,23,23,03,C3,3C,A1,21,07,01,CD,75,&04FD
2021 DATA BB,CD,9C,BB,21,9E,A0,7E,FE,00,CA,64,A1,CD,5A,BB,&096B
2022 DATA 23,C3,57,A1,CD,9C,BB,CD,06,BB,21,09,09,CD,75,BB,&07C0
2023 DATA 06,00,3E,00,32,CE,A0,3A,CE,A0,57,1E,00,3A,27,A0,&0502
2024 DATA 4F,21,25,A0,DF,CF,A0,D0,3E,2A,CD,5A,BB,3A,CE,A0,&0845
2025 DATA FE,27,C8,3C,32,CE,A0,21,25,A0,06,09,77,23,23,23,&059E
2026 DATA 23,10,F9,18,D2,86,00,27,00,C1,C3,C5,C7,C9,C2,C4,&0822
2027 DATA C8,C8,FF,41,43,45,47,49,42,44,46,48,FF,&05F9

```

Der Aufruf dieses Programmes geschieht durch CALL &A000. Danach übernimmt das Programm die weiteren Abfragen.

Auch dieses Programm können Sie als Binärfile abspeichern. Die Anfangsadresse ist &A000 und die Länge können Sie dadurch berechnen, daß Sie den in Zeile 880 genannten Wert "a" (=Anfang) vom Wert "e" (=Ende) subtrahieren.

Damit haben Sie eine - gegenüber anderen "Formatierern" - zwar sehr umfangreiche, aber auch vielseitig einsetzbare Routine, denn fast alle Parameter können bei diesem Universalformatierer von Basic aus geändert werden.

Dieses Programm ist also Ihr "Programmnormal", wenn Sie auch einmal ein anderes Format schreiben wollen.

Damit die entsprechenden Anpassungen und Veränderungen aber von Ihnen durchgeführt werden können, ist das Assemblerlisting erforderlich. (Dort können Sie feststellen, was alles veränderbar ist!)

#### 9.4.2 Assemblerlisting: FORMAT.EDI

```

;*****
;* Formatierprogramm (System und Data-Only) *
;*      fuer CPC 464 bis CPC 6128      *
;*****
dump
;write "format.bin"      ;binaerfile speichern

org &a000      ;Start bei a000

output equ &bb5a:clearwi equ &bb6c:textinv equ &bb9c:setcurs equ &bb75
waitchar equ &bb06:setmode equ &bc0e:findco equ &bcd4
;-----
ld a,(starttrack)      ;erste Spur fuer Formatierung
ld (startspur+1),a      ;im Programm festlegen

ld a,(endtrack)      ;letzte Spur fuer Formatierung
ld (vergleich+1),a      ;im Programm festlegen

ld a,(unit)      ;Laufwerk
ld (laufwerk+1),a      ;auswaehlen

call start      ;Sprung zum eigentlichen Start

;*****
;* Tabelle wieder korrigieren *
;*****
ld a,&09      ;Hilfszaehler auf 9
ld b,a      ;ins Register b
ld hl,hlp      ;Zeiger Headerinfo-Puffer
ld a,&00      ;Akku mit 0 laden

rueckst
ld (hl),a      ;HIP-Position
inc hl:inc hl:inc hl
inc hl      ;viermal erhoehen
djnz rueckst      ;Hilfszaehler erniedrigen

```

```

ret                                ;Programmlauf beendet
;-----
hip                                ;Die Headerinfos (&ff) werden beim
                                ;Programmlauf auf das entsprechende
                                ;Format gesetzt. Das Ersatzbyte &FF
                                ;wird (normalerweise !!)
                                ;auf &Cx bzw. 4x gesetzt

defb &00,&00                      ;Folge der Headerinfos:
sektor
defb &ff,&02                      ;Track-#,Kopf-# Sector-#,Sectorgroesse-7
defb &00,&00,&ff,&02
defb &00,&00,&ff,&02
defb &00,&00,&ff,&02
defb &00,&00,&ff,&02
defb &00,&00,&ff,&02
defb &00,&00,&ff,&02
defb &00,&00,&ff,&02
defb &00,&00,&ff,&02

text1
defb " ** Formatierprogramm fuer CPC 464 .... 6128 ** ":defb &00

text2
defb "System- oder Data-only-Format ?"
defb &00

text3
defb "Bitte Diskette einlegen und eine Taste druecken"
defb &00

track
defb &00                          ;Trackmarker
faradr
defb &00,&00                      ;Far-Adresse im ROM
romstate
defb &00                          ;Disk-ROM ausgewaehlt

start
ld hl,befehl                      ;Formatierbefehl
call findco                       ;im ROM suchen
ret nc                            ;nicht gefunden
ld (faradr),hl                    ;Faradresse merken
ld a,c                            ;Romstate (normal 7)
ld (romstate),a                  ;merken

ld a,&2
call setmode                      ;MODE 2 setzen
call textinv                      ;Invers schalten
ld hl,text1                       ;Text "Format....."

wei
ld a,(hl)                        ;Textzeichen holen
cp &0                             ;ist der String zu Ende ?
jp z,aus1                        ;ja,fertig
call output                      ;Stringzeichen ausgeben
inc hl                           ;hl auf naechstes Zeichen

```

```

jp wei                ;alles wieder von vorne

aus1
call textinv          ;wieder normal
ld hl,&0103            ;Wert fuer Cursorposition
call setcurs          ;Cursor setzen
ld hl,text2           ;Auf "System ..." setzen

wei2
ld a,(hl)             ;Akkumulator mit Zeichen laden
cp &0                 ;Ende des Strings ?
jp z,warte            ;ja, weiter
call output           ;Zeichen ausgeben
inc hl                ;neu positionieren
jp wei2              ;und weiter

warte
call waitchar         ;auf Taste warten
cp a,"s"              ;s fuer System
jp z,system           ;System ausgewaehlt
cp a,"d"              ;d fuer Data-only
jp z,data             ;Data-only ausgewaehlt
jp warte              ;ungueltige Eingabe

system
ld bc,systable        ;die Systemdaten
ld hl,sektor          ;holen

m11
ld a,(bc)             ;Tabelleninhalt
cp &ff                ;Tabellenende ?
jp z,discein          ;Sprung Disk einlegen usw.
ld (hl),a             ;Daten in die Tabelle
inc hl:inc hl:inc hl  ;um 4 erhoehen
inc hl                ;Zeiger erhoehen
inc bc                ;
jp m11

data
ld bc,dattable        ;die Data-only-Daten
ld hl,sektor          ;holen

m12
ld a,(bc)             ;Tabelleninhalt
cp &ff                ;Tabellenende ?
jp z,discein          ;Sprung zu Disk einlegen usw.
ld (hl),a             ;Daten in die Tabelle
inc hl:inc hl:inc hl  ;um 4 erhoehen
inc hl                ;Zeiger erhoehen
inc bc                ;
jp m12

discein
ld hl,&0107            ;Cursorposition festlegen
call setcurs          ;und setzen
call textinv          ;Text invertieren
ld hl,text3           ;Zeiger auf Text 3

```

```

wei4
ld a,(hl)           ;Zeichen in Akku
cp &0              ;Ende erreicht ?
jp z,warte2         ;ja, weiter
call output         ;Zeichen ausgeben
inc hl             ;Zeiger erhoehen
jp wei4            ;weitermachen

warte2
call textinv        ;Textausgabe rueckstellen
call waitchar       ;auf Eingabe warten
;-----
ld hl,&0909         ;Cursor positionieren
call setcurs
ld b,0
startspur          ;Startspur
ld a,&0             ;0 in Akku
ld (track),a       ;und in Trackmarker

arbeite            ;naechste Spur
ld a,(track)        ;Track # holen
ld d,a             ;und in d ablegen

laufwerk           ;LW-Nummer
ld e,&00            ;e=Laufwerk
ld a,(sektor)       ;Kennzeichen fuer 1.Sektor
ld c,a             ;# 1. Sectors der Spur
ld hl,hip           ;Header-Puffer

rst 3,faradr       ;Far-call-Adr.+RDM-Select
ret nc             ;Fehler !!!!
;Rueckkehr nach RST 3
ld a,"*"           ;pro Track je
call output         ;ein * ausgeben
ld a,(track)        ;Track holen
vergleich cp &27    ;Endtrack erreicht?
ret z              ;ja deshalb wieder zurueck
inc a               ;nein, Tracknr erhoehen
ld (track),a       ;und merken
ld hl,hip           ;Trackwerte (Headerkopf)
ld b,&09            ;(Hilfszaehler fuer DJNZ)

lm
ld (hl),a           ;Track #
inc hl:inc hl       ;auf neuen Headerkopf
inc hl:inc hl       ;4 Adressen hoeher
djnz lm             ;B dec weiter falls > 0
jr arbeite          ;zurueck zum weitermachen

befehl defb &86
starttrack defb &00
endtrack defb &27
unit defb &00

dattable            ; Reihenfolge (data-only)
defb &c1,&c3,&c5,&c7,&c9,&c2,&c4,&c6,&c8,&ff

```

```

systable          ;Reihenfolge (system)
defb &41,&43,&45,&47,&49,&42,&44,&46,&48,&ff

```

Was Sie mit dieser Formaterroutine - außer einem normalen Formatieren im Laufwerk A - alles anstellen können, wenn Sie verschiedene Parameter ändern, wird Ihnen bei der Benutzung der nachfolgenden Programme sehr schnell klar werden.

Denn das nächste Programm ist ebenfalls eine Routine zum formatieren. Allerdings nicht für die komplette Diskette, sondern nur für eine Spur und diese wird auch noch fehlerbehaftet formatiert! Dieser Fehler wurde aber absichtlich eingebaut, denn wir wollen Ihnen ja zeigen, daß nicht nur mit den üblichen Formatierungen und Sektornummern gearbeitet werden kann.

## 9.5 Ein absichtlicher Sektor-Fehler

### 9.5.1 Listing: WRONGSEC.HEX

```

8000 MEMORY &9FFF
8700 REM WRONGSEC.HEX
8800 a= 40960:e= 41099:zb=2000
8900 FOR i =a TO e:READ d$:IF LEFT$(d$,1)<>"&" THEN 920
9000 IF ps<>VAL(d$) THEN PRINT"Fehler Zeile "zb:END
9100 ps=d$:d$="":IF i=e THEN END:ELSE i=i-1:zb=zb+1:GOTO 950
9200 d$="&"&d$
9300 POKE i,VAL(d$):ps=ps+VAL(d$)
9400 PRINT VAL(d$)
9500 IF i < e THEN NEXT i
2000 DATA 21,41,A0,CD,D4,BC,22,3E,A0,79,32,40,A0,1E,00,16,&061E
2001 DATA 02,0E,41,21,67,A0,DF,3E,A0,21,46,A0,11,8B,A0,01,&057A
2002 DATA 16,00,ED,B0,21,45,A0,CD,D4,BC,22,42,A0,79,32,44,&0709
2003 DATA A0,1E,00,16,02,0E,51,21,8B,A0,DF,42,A0,C9,42,C0,&060D
2004 DATA 07,86,3F,C0,07,85,54,65,73,74,74,65,78,74,20,76,&0613
2005 DATA 6F,6D,20,41,75,74,6F,72,20,64,65,73,20,50,72,6F,&05B4
2006 DATA 67,72,61,6D,6D,65,73,02,00,41,02,02,00,42,02,02,&0379
2007 DATA 00,43,02,02,00,44,02,02,00,45,02,02,00,51,02,02,&012D
2008 DATA 00,47,02,02,00,48,02,02,00,49,02,02,&00E2

```

Selbstverständlich auch zu diesem Programm die kommentierte Maschinensprache-Routine.

### 9.5.2 Assembler-Listing: WRONGSEC.EDI

```

;*****
;* WRONGSEC.EDI *
;*****

org &a000
findcom equ &bcd4:track equ &02:head equ &00

ld hl,befehl1      ;Befehlswort holen (Format Track)
call findcom       ;Befehl im ROM suchen

```

```
ld (form),hl      ;Adresse speichern
ld a,c            ;ROM-State
ld (form+2),a     ;merken
ld e,&00          ;Laufwerk Null
ld d,track        ;Spur 2
ld c,&41          ;1. Sektor-Nummer
ld hl,paramb      ;HL auf den Parameter-Block richten
rst &18           ;formatieren
```

defw form

```
ld hl,inhalt      ;Zeiger auf Textinhalt
ld de,buf         ;Zeiger auf Ziel
ld bc,33          ;Anzahl der Zeichen
ldir              ;und verschieben (kopieren)

ld hl,befehl2     ;ROM-Adresse holen (Write Sector)
call findcom      ;suchen und
ld (adr1),hl      ;ROM-Adresse merken
ld a,c            ;ROM-State
ld (adr1+2),a     ;merken
ld e,&00          ;Laufwerk 0
ld d,&02          ;Spur 2
ld c,&51          ;falsche Sektor-Nummer
ld hl,buf         ;Buffer fuer Aufruf uebergeben
rst &18           ;falschen Sektor schreiben
```

```
defw adr1
ret
```

```
form defs 3
befehl1 defb &86 ;Track formatieren
adr1 defs &3
befehl2 defb &85 ;Sektor schreiben
inhalt
text "Testtext vom Autor des Programmes"
paramb equ $ ;parameterblock
sect1 defb track:deff head:deff &41:deff 2
sect2 defb track:deff head:deff &42:deff 2
sect3 defb track:deff head:deff &43:deff 2
sect4 defb track:deff head:deff &44:deff 2
sect5 defb track:deff head:deff &45:deff 2
sect6 defb track:deff head:deff &51:deff 2 ;! falsch
sect7 defb track:deff head:deff &47:deff 2
sect8 defb track:deff head:deff &48:deff 2
sect9 defb track:deff head:deff &49:deff 2
```

buf defs 512

**Ein ganz wichtiger Hinweis!**

Nehmen Sie bitte für diese Experimente eine Test-Diskette, die wirklich nur für Testzwecke gedacht ist, denn wir garantieren Ihnen, daß Sie sonst Programme oder Daten "verlieren".

Die Testdiskette sollte für dieses "Experiment" im Systemformat formatiert sein! Der Aufruf des "hinterhältigen" Programmes geschieht mit CALL &A000.

Damit Sie aber auch Überprüfen können, was nun alles auf Ihrer Test-Diskette passiert ist, hier eine Routine, mit der Sie die Sektornummern einer Diskette herausfinden können.

Denn mit normalen "Zugriffen" bleibt Ihnen dieser Sektor und auch dessen Inhalt ansonsten verborgen. Vom normalen Basic aus, haben Sie überhaupt keine Möglichkeit etwas zu bemerken, geschweige denn diesen Sektor zu lesen.

Zuerst also ein Programm um die Diskette zu "checken", also um festzustellen, welche Sektornummern vorhanden sind. Diese Routine kann auch dazu dienen, die korrekte Formatierung zu überprüfen, denn nur dann wenn auf jeden Sektor mit den normalen Sektornummern zugegriffen werden kann, ist diese ordnungsgemäß formatiert.

Wir haben dieses Programm SNRCHECK "getauft". Dies ist die Abkürzung von Sektor-Nummern-Check.

#### 9.5.3 Listing: SNRCHECK.ED1

```
write "snrcheck"
;*****
;**      snrcheck      ***
;** Sektor-Nummern-Check ***
;*****

org &a200

findco equ &bcd4
anzahl equ &be66

ld a,(menge) ;Anzahl der Zugriffsversuche
ld (anzahl),a ;festlegen und in &be66 schreiben
ld a,&ff      ;keine Fehlermeldungen
ld (&be78),a ;mehr ausgeben

ld hl,bef     ;Befehl holen (Lesebefehl = 84)
call findco   ;Kommando suchen
ret nc        ;nicht gefunden

ld (adr),hl   ;Adresse im ROM merken
ld a,c        ;Select in Akku (Rom/Nummer)
ld (sel),a    ;und merken
ld a,(drive)  ;Drive-# holen
ld e,a        ;und nach E uebergeben
ld a,(track)  ;Spur-# holen
ld d,a        ;und nach D uebergeben
ld a,(secnr)  ;Sektor-# holen
ld c,a        ;und nach C uebergeben
ld b,0        ;Kopfnummer !

lesen
```



```

ld hl,buf      ;Bufferadresse fuer Sektor einlesen
rst 3,adr      ;Lese-Befehl durchfuehren

ld a,0         ;Fehlermeldungen wieder
ld (&be78),a   ;zulassen
ret            ;ab -> Basic

;-----
;Unterroutine um den Bufferspeicher mit 00 zu fuellen
;und dann an den Anfang das Wort "test" zu schreiben
;Aufruf mit call &a034
;Startadresse des Unterprogrammes
;
fuell
ld hl,schluss  ;hl mit Obergrenze des Bufferspeichers laden
ld a,0         ;Fuellzeichen in den Akku
ld b,&ff       ;Anzahl der zu "fuellenden" Speicherstellen
null ld (hl),a ;Akku-Inhalt in Adresse (hl) schreiben
dec hl        ;Adresse erniedrigen
djnz null     ;weitermachen falls der Zaehler (b) noch nicht null ist

ld b,4         ;vier Zeichen (Zaehler)
ld hl,testwort ;Adresse des Textes "test" holen
ld de,buf      ;Adresse des Buffers in DE

weiter
ld a,(hl)      ;Zeichen holen
ld (de),a      ;und schreiben
dec b         ;Zaehler erniedrigen
ld a,b        ;Zaehlerwert in den Akku
cp 0           ;und pruefen ob alle 4 Zeichen uebertragen sind
ret z         ;ja, fertig
inc hl        ;Quelladresse erhoehen
inc de        ;Zieladresse erniedrigen
jp weiter     ;und weitermachen
;-----

bef  defb &04   ;Lesebefehl
adr  defs 2    ;Adresse im Disk-ROM
sel  defs 1    ;Select (Nummer des Disk-ROMs)

track defb &00 ;Spur-Merker

drive defb &00 ;Drive-Merker

secnr defb &00 ;ohne Zuweisung

defb &00       ;Sektor-#
menge defb &02 ;Anzahl der Leseversuche
testwort text "test"

org  $
buf  defs 512
schluss

```

Dies war das kommentierte Assemblerlisting.

Hier nun das "komplette" Programm mit der Maschinenroutine als Datastatements.

#### 9.5.4 Listing: SNRCHECK.BAS

```

100 'SNRCHECK.BAS
110 MEMORY &9FFF:MODE 1:PRINT"SNRCHECK
120 PRINT:PRINT"Dieses Programm sucht Sector-Nummern
130 PRINT"auf einer Diskette!
140 IF PEEK (&A200)<> &3A THEN GOSUB 540
150 PRINT:PRINT"Bitte nun die zu untersuchende Diskette
160 PRINT"einlegen und eine Taste druecken!
170 CALL &BB18
180 MODE 2:PRINT"SEKTOR-CHECK (sucht Sektor-Nummern)
190 PRINT STRING$(79,"-")
200 WINDOW #1,1,80,4,23
210 LOCATE 3,3:PRINT"(          Hex          /dezimal )
220 LOCATE 40,3:PRINT"(          Hex          /dezimal )
230 mps=&A200
240 trackmerk =mps+&56:drivemerk =mps+&57:buffer =mps+&5F
250 sformmerk =mps+&58:anzmerk =mps+&5A
260 trackstart = 0:trackend=40:startform=&3F:endform = &4A:fuell =&A233
270 track = trackstart:anzahl=1:drive=0:secform=startform
280 POKE drivemerk,drive
290 lin=1:posi=0
300 '
310 LOCATE 50,1:PRINT"Spur :&HEX$(track,2)
320 LOCATE 62,1:PRINT "Form+Sektor :&HEX$(secform,2)
330 CALL fuell: pruef=0
340 secform = secform +1:GOSUB 450
350 POKE trackmerk,track:POKE sformmerk,secform
360 '
370 CALL mps:FOR i = buffer TO buffer+3:pruef=pruef+PEEK(i):NEXT i
380 IF pruef <>448 THEN posi=posi+3:GOTO 410
390 GOTO 300
400 '
410 LOCATE #1,posi,lin: PRINT #1,HEX$(secform,2),"":GOSUB 450
420 szaehl=szaehl+1:IF szaehl=9 THEN szaehl=0:secform=endform:GOSUB 450
430 GOTO 300
440 '
450 IF secform=endform THEN secform=startform:track=track +1:GOSUB 490
460 IF track =trackend THEN 520
470 RETURN
480 '
490 posi=posi+3:LOCATE #1,posi,lin:PRINT #1,"Spur: "track-1:posi=posi+8
500 IF posi >44 THEN posi= 0:lin =lin+1
510 GOTO 300
520 LOCATE 1,24:END
530 '-----
540 REM snrcheck.hex
550 a= 41472:e= 41568:zb=2000
560 FOR i =a TO e:READ d$:IF LEFT$(d$,1)<>"&" THEN 590
570 IF ps<>VAL(d$) THEN PRINT"Fehler Zeile "zb:END
580 ps=0:d$="":IF i=e THEN RETURN:ELSE i=i-1:zb=zb+1:GOTO 610
590 d$="&"&d$
600 POKE i,VAL(d$):ps=ps+VAL(d$)

```

```

610 IF i < e THEN NEXT i
2000 DATA 3A,5A,A2,32,66,BE,3E,FF,32,78,BE,21,52,A2,CD,D4,&07E7
2001 DATA BC,D0,22,53,A2,79,32,55,A2,3A,57,A2,5F,3A,56,A2,&0709
2002 DATA 57,3A,58,A2,4F,06,00,21,5F,A2,DF,53,A2,3E,00,32,&0546
2003 DATA 78,BE,C9,21,5F,A4,3E,00,06,FF,77,2B,10,FC,06,04,&061E
2004 DATA 21,5B,A2,11,5F,A2,7E,12,05,78,FE,00,C8,23,13,C3,&05FC
2005 DATA 46,A2,84,3C,C0,07,02,00,40,00,02,74,65,73,74,74,&04E7

```

Dieses Programm sucht also in jeder Spur nach Sektoren-Nummern von 0 bis 255. Aus diesem Grunde ist die Laufzeit dieser Routine auch sehr lang. Aber am Ende wissen Sie ganz genau, welche Sektornummern auf der untersuchten Diskette vorhanden sind.

## 9.6 Der direkte Diskettenzugriff

Maschinenprogramme zum direkten Zugriff auf Schneider-Diskettenstationen können durchaus kürzer und ohne die oft von Basic aus zu bestimmende Parameterübergabe erfolgen, aber aufgrund der Erfahrung mit einem bereits früher veröffentlichten Programm des Autors, daß Besitzer der Vortex-Diskettenstationen nicht damit zurecht kamen, wurde nachfolgendes Programm so geschrieben daß auch diese "bedient" werden können. Nur muß im Programm dann auch noch der Kopf ausgewählt werden (Übergabe im Register B). Außerdem haben die Vortex-Laufwerke 80 Spuren!

### 9.6.1 Assemblerlisting: SECREAD.EDI

```

;*****
;*      SECREAD.EDI      *
;*      sectorread fuer alle      *
;*      Diskettenstationen      *
;*****

org &a1e8

akdrive equ &a702:findcom equ &bcd4

add a,a      ;Anzahl verdoppeln
ld (anzahl),a ;und merken
ld (quelle),ix ;woher
ld bc,(anzahl) ;Anzahl
ld hl,(quelle) ;Quelladresse
ld de,befehl ;wohin verschieben
ldir        ;verschieben

ld a,(drive) ;gewuenshtes Laufwerk
ld (drive),a ;merken
ld hl,befehl ;Befehl aus Tabelle
call findcom ;Befehl im ROM suchen

ld (romadr),hl ;Romadresse merken
ld a,c      ;Romselect
ld (romsel),a ;merken
ld hl,drive ;Adr fuer Drive # holen

```

```

ld e,(hl)      ;nach E uebergeben
ld hl,track    ;Adr fuer Track # holen
ld d,(hl)      ;nach d uebergeben
ld hl,formse   ;Format + Sektor
ld c,(hl)      ;in Reg C
ld hl,(buffer) ;Bufferadr in HL
rst 3,romadr    ;weitermachen bei ROMADR
ret            ;und zurueck

```

```
anzahl defs 2:quelle defs 2
```

```

befehl defs 2:buffer defs 2:formse defs 2:track defs 2
drive  defs 2:romadr defs 2:romsel defs 1

```

Wie dieses Maschinenprogramm von Basic aus bedient wird, zeigt das nachfolgende kleine Basicprogramm. Dieses Programm greift nach Eingabe von Track und FORMSE (Format und Sektor) auf die Diskettenstation zu und holt den Inhalt des ausgewählten Sektors in den festgelegten Buffer.

#### 9.6.2 Listing: SECREAD.BAS

```

100 MEMORY &8FFF
110 MODE 2:PRINT"Programm SECREAD.BAS
120 PRINT"Dieses Programm liest den ausgewählten Sektor
121 INPUT "Bitte die Track-Nummer eingeben (dezimal)";track
130 PRINT"FORMSE STANDARD: 0x (IBM) 4x (SYSTEM) Cx (DATA-ONLY)
140 INPUT"Diskettenformat + Sektor als FORMSE eingeben";formse$
141 formse$="&" + formse$:formse=VAL(formse$)
150 '
160 IF PEEK(&A1E8)<> &87 THEN GOSUB 460
170 '
180 PRINT"Daten fuer Maschinenprogrammaufruf werden festgelegt!
190 drive=0:buffer=&9000:befehl=&84
200 FOR i = buffer TO buffer+512:POKE i,0:NEXT
210 '
220 PRINT"Maschinenprogramm wird aufgerufen!
230 CALL &A1E8,drive,track,formse,buffer,befehl
240 '
250 PRINT"Rueckkehr aus der Maschinenspracheroutine!
260 PRINT:PRINT"Folgende Parameter wurden im M-Programm verwendet (in hex):
270 ga=&A226
280 befehl = PEEK(ga):PRINT"Befehl = "HEX$(befehl)" ";
290 buffer = PEEK(ga+2)+PEEK(ga+3)*256:PRINT"Buffer = "HEX$(buffer,4)" ";
300 formse = PEEK(ga+4):PRINT"Format + Sektor = "HEX$(formse)
310 track = PEEK(ga+6):PRINT"Spur = "HEX$(track,2)" ";
320 drive = PEEK(ga+8):PRINT"Laufw. = "HEX$(drive,2)
330 romadr = PEEK(ga+10)+PEEK(ga+11)*256:PRINT"ROM-Adr. = "HEX$(romadr,4)" ";
340 romsel = PEEK(ga+12):PRINT"Romselect = "romsel
350 '
360 PRINT:PRINT"Nun erfolgt die teilweise Ausgabe des Bufferinhaltes:"
370 PRINT
380 '
390 FOR i = buffer TO buffer+256:w=PEEK(i):IF w<32 THEN w=32
400 IF w>128 THEN w=w-64:GOTO 400' gesetzte Bits ausblenden

```

```

410 PRINT CHR$(w);
420 'z=z+1:IF z=12 THEN z=0:i=i+20:PRINT " ";
430 NEXT
440 END
450 REM secread.hex
460 a= 41448:e= 41506:zb=2000
470 FOR i =a TO e:READ d$:IF LEFT$(d$,1)<>"&" THEN 500
480 IF ps<>VAL(d$) THEN PRINT"Fehler Zeile "zb:END
490 ps=0:d$="":IF i=e THEN RETURN:ELSE i=i-1:zb=zb+1:GOTO 530
500 d$="&"&d$
510 POKE i,VAL(d$):ps=ps+VAL(d$)
520 '
530 IF i < e THEN NEXT i
2000 DATA 87,32,22,A2,DD,22,24,A2,ED,4B,22,A2,2A,24,A2,11,&063F
2001 DATA 26,A2,ED,B0,3A,2E,A2,32,2E,A2,21,26,A2,CD,D4,BC,&07B7
2002 DATA 22,30,A2,79,32,32,A2,21,2E,A2,5E,21,2C,A2,56,21,&0528
2003 DATA 2A,A2,4E,2A,28,A2,DF,30,A2,C9,&0468

```

Wenn Sie das Semikolon in Zeile 420 entfernen und bei der Frage nach der Spur "Null", beim Sektor "C1" eingeben und eine Diskette eingelegt haben, dann wird Ihnen ein Teil des Inhaltsverzeichnis dieser Diskette ausgegeben. Ein kleines "e" vor dem Filenamen bedeutet "gelöscht".

Wollen Sie noch mehr über Ihre Disketten wissen? Wollen Sie sehen, welche Einträge auf welchen User-Ebenen sind? Mit diesem Programm erfahren Sie viele Details Ihrer Disketten.

### 9.6.3 Listing: DINFO.BAS

```

100 'DINFO - Gibt Informationen ueber die Files einer Diskette
110 '
120 MODE 2:MEMORY &9FFF:POKE &BE78,255
130 PRINT"Bitte die zu untersuchende Diskette einlegen"
140 PRINT"und eine Taste druecken ":CALL &BB18
150 WINDOW #0,1,80,1,20:WINDOW #3,1,80,24,24
160 '
170 GOSUB 850:PRINT"DINFO - Ein Disketteninformationsprogramm":GOSUB 850
180 WINDOW #0,1,80,6,20:WINDOW #1,1,80,3,20
190 FOR i = 1 TO 2000:NEXT
200 '
210 ON ERROR GOTO 600
220 anz=14:intr = 63:DIM x$(intr):catstart=HIMEM-2047
230 tr$=STRING$(79,"-"):LOCATE 1,20:PRINT tr$:LOCATE 1,1:PRINT tr$
240 '
250 PRINT #1,"Ich durchsuche die User-Ebenen"TAB(50)"User:
260 LOCATE #1,63,1:PRINT #1,"Anzahl:";
270 PRINT"Files sind auf folgenden User-Ebenen:"PRINT
280 '
290 FOR u = 0 TO 15:USER,u
300 MEMORY &9FFF
310 POKE &BB5A,&C9:CAT:POKE &BB5A,&CF:MEMORY HIMEM -&FFF
320 '
330 IF PEEK(catstart)<>0 THEN PRINT"User "u " ";
340 LOCATE #1,55,1:PRINT #1,u;

```

```

350 '
360 FOR k=0 TO eintr
370 IF PEEK(catstart+k*anz)=0 THEN k=eintr:GOTO 430
380 a=a+1:LOCATE #1,70,1:PRINT #1,a;
390 FOR i = 1 TO anz-3
400 x$(a-1)=x$(a-1)+CHR$(PEEK(catstart+i*k*anz))
410 NEXT i
420 x$(a-1)=HEX$(u,2)+x$(a-1)
430 NEXT k:NEXT u
440 '
450 !USER,0
460 CLS:CLS #1:PRINT #1,"Folgende File-Eintraege sind auf der Diskette:
470 u$=LEFT$(x$(0),2):PRINT"User "VAL("&"&u$):PRINT
480 '
490 FOR i = 0 TO a-1
500 IF LEFT$(x$(i),2)=u$ THEN 530
510 u$=LEFT$(x$(i),2):GOSUB 840
520 CLS #0:PRINT"User "VAL("&"&u$):PRINT
530 PRINT RIGHT$(x$(i),11)" ";:NEXT:PRINT:GOSUB 840
540 '
550 CLS #1:PRINT#1,"Nun die File-Analysen
560 FOR i = 0 TO a-1
570 x$=MID$(x$(i),3,8)+". "+RIGHT$(x$(i),3):u=VAL("&"&LEFT$(x$(i),2))
580 !USER,u:POKE &BE78,255
590 OPENIN x$
600 CLOSEIN:RESUME NEXT
610 CLS:PRINT"User: "u:PRINT
620 PRINT TAB(33)"Echter File-Name: "TAB(63);:GOSUB 850:PRINT x$:GOSUB 850
630 PRINT TAB(33)"File-Name lt. OPENIN-Eintrag: ";:GOSUB 850
640 FOR j = &A756 TO &A764:PRINT CHR$(PEEK(j));:NEXT j
650 PRINT" ":GOSUB 850:PRINT:PRINT
660 '
670 d1=PEEK(&A76A)+PEEK(&A76B)*256
680 l1=PEEK(&A76D)+PEEK(&A76E)*256
690 ea=PEEK(&A76F)+PEEK(&A770)*256
700 '
710 ft=PEEK(&A767):ft$="UNBEKANNT
720 IF ft = 0 THEN ft$="BASIC
730 IF ft = 1 THEN ft$="BASIC PROTECTED
740 IF ft = 2 THEN ft$="BINARER
750 IF ft = 22 THEN ft$="ASCII
760 PRINT:PRINT"Filetyp          : "ft$:PRINT
770 '
780 PRINT"Ursprung              : &HEX$(d1,4)
790 PRINT"logische Laenge      : &HEX$(l1,4)
800 PRINT"Einsprung bei MPG    : &HEX$(ea,4)
810 GOSUB 840
820 '
830 NEXT i:!USER,0:MODE 2:END
840 PRINT #3,"Bitte ein Taste druecken":CALL &BB10:CLS#3:RETURN
850 CALL &BB9C:RETURN

```

Haben Sie auch schon einmal aus Versehen ein eben erstelltes Basic-Programm als ASCII-File abspeichern wollen und dann ganz in Gedanken anstelle eines "a" ein "p" eingegeben?

Wenn nein, in Ordnung. Falls aber doch, wie haben Sie Ihr Programm wieder "entschützt"?

Hoffentlich, war Ihr Programm nicht für immer verloren!

Besitzer eines CPC 464 haben mit Folgeproblemen durch einen derartigen Fehler kaum Schwierigkeiten, es gibt genügend Programme, die eine der Indirections patchen und dann ein normales Einlesen eines geschützten Programmes ermöglichen.

Die anderen waren schlechter dran, aber nun auch nicht mehr, mit PCOPY ist das Problem gelöst.

#### 9.6.4 Listing: PCOPY.EDI

```
write "pcopy.bin"
;*****
;*      PCOPY.EDI      *
;*-----*
;* Liest mit SAVE"NAME",P *
;* abgespeicherte Programme *
;* ein und schreibt diese *
;* als neues File auf Disk *
;* Nach dem Programmlauf er-*
;* folgt ein Systemreset ! *
;*****
org &a000

laenge      equ &c:casinopen      equ &bc77:casoutopen  equ &bc8c
casinclose   equ &bc7a:casoutclose equ &bc8f:casindirect equ &bc83
casoutdirect equ &bc98:basstart   equ &170: block      equ &c000
waitchar     equ &bb06

.start
.beginn
ld b,laenge      ;Laenge des Namens (insgesamt 12 Zeichen)
ld de,block      ;Blockbuffer in DE
ld hl,name       ;Adresse des Namens laden
call casinopen   ;File oeffne
ld hl,basstart   ;Adresse des Basicstarts
call casindirect ;Programm direkt holen
call casinclose  ;File schliessen

;schreiben
ld b,laenge      ;Filenamenaenge
ld hl,name       ;Adresse des Namens
ld de,block      ;Blockbuffer fuer Schreiben
call casoutopen  ;File zum schreiben oeffnen
ld hl,basstart   ;Startadresse
ld de,(&a76d)     ;Laenge des Files
ld a,0           ;Fileart festlegen
call casoutdirect ;File schreiben
call casoutclose ;File schliessen
rst 0            ;Reset damit kein Unsinn passiert!
name
defb "dummyname.bas"
```

Das zugehörige Basic-Programm lädt den durch den Hexlader erzeugten Maschinencodeteil nach, und fordert Sie auf, den Programmnamen einzugeben. Nach dem Programmlauf führt der CPC einen Systemreset durch und das normale Basic-Programm steht wieder auf Ihrer Diskette, das geschützte Programm hat die Extension ".BAK".

#### Listing: PCOPY.BAS

```
100 'Basisteil PCOPY
110 '
120 MODE 2:MEMORY &9FFF
130 PRINT"PCOPY":PRINT
140 PRINT"Ein Programm zum Rueckwandeln geschuetzter Basicprogramme!":PRINT
150 PRINT"!! Dieses Programm darf nur fuer Software eingesetzt werden
160 PRINT"deren Kopierrechte Sie besitzen !!
170 '
180 LOAD"pcopy.bin",&A000:PRINT
190 INPUT "Filename (ohne Extension .BAS)";Filename$
200 IF LEN(filename$)>8 THEN PRINT"Eingabefehler":GOTO 190
210 IF LEN(filename$)=0 THEN filename$="Krasser Bedienungsfehler":GOTO 200
220 IF LEN(filename$)<8 THEN filename$=filename$+" ":GOTO 220
230 '
240 FOR i = 1 TO 8
250 a=ASC(MID$(filename$,i,1))
260 POKE &A02E+i,a
270 NEXT
280 '
290 PRINT:PRINT"Nicht erschrecken, am Ende des Programmlaufes
300 PRINT"erfolgt ein Systemreset!
310 CALL &A000
```

Damit das Basic-Programm aber das erforderliche Binärfile auch laden kann, muß es erst erzeugt werden. Dazu dient der nachfolgende Hexlader.

#### 9.6.6 Listing: PCOPY.HEX

```
850 REM pcopy.hex
860 MEMORY &9fff
880 a= 40960:e= 41019:zb=2000
890 FOR i =a TO e:READ d$:IF LEFT$(d$,1)<>"&" THEN 920
900 IF ps<>VAL(d$) THEN PRINT"Fehler Zeile "zb:END
910 ps=0:d$="":IF i=e THEN END:ELSE i=i-1:zb=zb+1:GOTO 950
920 d$="&"&d$
930 POKE i,VAL(d$):ps=ps+VAL(d$)
940 PRINT VAL(d$)
950 IF i < e THEN NEXT i
2000 DATA 06,0C,11,00,C0,21,2F,A0,CD,77,BC,21,70,01,CD,83,&05B5
2001 DATA BC,CD,7A,BC,06,0C,21,2F,A0,11,00,C0,CD,8C,BC,21,&06C8
2002 DATA 70,01,ED,5B,6D,A7,3E,00,CD,98,BC,CD,8F,BC,C7,64,&086F
2003 DATA 75,6D,6D,79,6E,61,6D,2E,62,61,73,&0468
```

Die Anfangs- und Endadresse für den Maschinencode-Teil finden Sie



in Zeile 880 und können sich dadurch dann - die für die Abspeicherung erforderliche - Längenangabe ausrechnen.

Nun zum Abschluß der doch sehr konzentrierten Disketteninformationen (Wir bringen an anderer Stelle noch mehr) ein "Retter in höchster Not".

Das Programm FRETTER gibt Ihnen die Chance, aus Versehen gelöschte Diskettenfiles wieder zu restaurieren. Dies gelingt aber nur dann mit Sicherheit, wenn Sie nach einer versehentlichen Löschung nicht wieder auf die Diskette geschrieben haben!!!

Bei einer Filelöschung wird auf der Diskette nämlich nicht wirklich gelöscht, sondern nur die USER-Nummer (im Directory) auf &E5 gesetzt und diese Kennzeichnung &E5 bewirkt, daß beim Speichern von Files, dieser Eintrag wieder beschrieben, also überschrieben, werden darf. Deshalb kann vor einem neuerlichen Schreibvorgang dieses Kennzeichen wieder auf eine "gültige" USER-Nummer gesetzt werden und das Programm ist wieder gerettet. Genau nach diesem Prinzip arbeitet das nachfolgende Programm. Da wir hoffen, daß Sie nun ausreichende Kenntnisse über Ihren CPC haben, schenken wir uns eine genaue Bedienungsanleitung für dieses Programm.

Denken Sie bitte daran:

Gültige User-Nummern sind von 1 bis 15 und ein Programm kann auch länger als 16 KB sein und deshalb mehrere Fileeinträge haben, die dann auf eine gültige User-Nummer gesetzt werden müssen.

Und vor allem: Üben Sie mit dem Programm schon vor dem eigentlichen Ernstfall!

Verzeichnis der File-Einträge (UN=Usernummer) Format: Data-on-D

Name	EXT	EN	UN	Name	EXT	EN	UN	Name	EXT	EN	UN	Name	EXT	EN	UN
ALISE	BAS	01	00	ASC-HEXP	BAS	02	00	DATAGEN	BAS	03	00	VORNAME	BAS	04	00
DATE12	BAS	05	00	INHALT	BAS	06	00	MINIMON2	BAS	07	00	DATE11	BAK	08	E5
TCODE	BAS	09	00	AUTO	BAS	0A	00	KONTOF	BAK	0E	E5	DATE25	BAK	0C	00
MORSEN	BAS	0D	00	MUSLIK	BAS	0E	00	MINI2	BAS	0F	00	DATEN	BAK	10	00
MINI1	BAS	11	00	TOKEFIND	BAS	12	00	MINI3	BAS	13	00	4-FIND	BAS	14	00
SUCHPROG	BAS	15	00	LOTTO1	BAS	16	00	MINIMAL	BAS	17	00	DATE12	BAK	18	E5
DATE11	BAS	19	00	MINIMORS	BAS	1A	00	TPMAN2	BAS	1B	00	KONTOF	BAK	1C	E5
CASAN	BAS	1D	00	DIXI	BAS	1E	00	TPUFFMAN	BAS	1F	00	KEYCODE	BAS	20	E5
DICHTER	BAS	21	00	KONTOF	BAS	22	00	TCEN	BAS	23	E5	STEUERZ	BAS	24	E5
LOTTO2	BAS	25	00	FARBKURS	BAS	26	00	STRINGS	BAS	27	00	DISKINHA	BAS	28	00
PAPERFAR	BAS	29	00	PENFARBE	BAS	2A	00	VORPEEK	BAS	2B	E5	4-DEMO	BAS	2C	00
CASLIST	BAS	2D	00	DIGUHR	BAS	2E	00	HLADAT	BAS	2F	00	BSTART	BAS	30	00
TPUFFER	BAS	31	00	FARBKOM	BAS	32	E5	SCREEN	BAS	33	00	RAROMON	BAK	34	E5
SPRUNG	BAS	35	00	PROGNOSE	BAS	36	00	DAYSON	BAS	37	E5	ESCONSCR	BIN	38	E5
SORTIER	BAS	39	E5	WLSORT	BAS	3A	00	ESX	BIN	3B	00	PEEKER	BAS	3C	E5
SORT	BIN	3D	00	KNACKER	BAS	3E	E5	KNACKER2	BAS	3F	00	INSTDRAN	BIN	40	00

Eintragsnummer (EN) des zu rettenden Files (0=ende): ? 32

Abbildung 15: Die gelöschten Einträge zeigen bei der USER-Nummer (UN) das Kennzeichen "E5"!

## 9.6.7 Listing: FRETTER

```

100 '*****
110 '* FRETTER = Disk-File-Retter fuer die CPC's (c) by LM *
120 '*****
130 '
140 MODE 2:MEMORY &9DFF:GOSUB 1270:PRINT"Programm: FILE-Retter":GOSUB 1270
150 '
160 'Adressen und Werte fuer Maschinenprogramm
170 aktdrivemc=&A702:befehlmc=&A622:buffermc=&A629:drivenmc=&A626
180 formsemc=&A628:trackmc=&A627:befehl=&84:mcstart=&A600
190 '
200 'Spaeter benoetigte Variable festlegen
210 mis$=" -":pl$=" +":tr$=STRING$(79,"-")
220 '
230 'Maschinenprogramm einlesen falls Pruefbyte nicht ok!
240 adr=mcstart
250 c=PEEK(adr):'Pruefbyte holen (Maschinenprg vorhanden ?)
260 IF c=58 THEN 300:'Sprung da Pruefbyte stimmt
270 LOAD"sector.bin",&A600
280 '
290 'Beginn des Bufferbereiches
300 buffer$="9E00"
310 '
320 PRINT:PRINT"Bitte entsprechende Diskette einlegen und Taste druecken"
330 CALL &BB06
340 '
350 OPENOUT "dummy":CLOSEOUT:CLS:' pruefen, welches Diskformat
360 drive=PEEK(aktdrivemc)
370 IF drive = 0 THEN form =PEEK(&A89F)-1:'Diskformat abholen und merken
380 IF drive = 1 THEN form =PEEK(&A8DF)-1:'Diskformat abholen und merken
390 IF form =&C0 THEN offset=0:fo$="Data-only"
400 IF form =&40 THEN offset=2:fo$="CP/M oder Vendor"
410 '
420 '*****
430 '* eigentlicher Start und Wiedereinsprung *
440 '*****
450 'Werte fuer Bufferadresse berechnen
460 bufflow=VAL("&"+RIGHT$(buffer$,2)):buffhigh=VAL("&"+LEFT$(buffer$,2))
470 sector=0:IF flag=1 THEN befehl=&85:flag=0
480 '
490 '*****
500 '* Komplettes Inhaltsverzeichnis einlesen oder schreiben *
510 '*****
520 sector = sector +1
530 track=offset
540 GOSUB 1210:' Sector einlesen oder schreiben
550 '
560 ' Falls sector = 4, dann ist das Inhaltsverzeichnis komplett
570 IF sector=4 THEN 650
580 '
590 buffhigh=buffhigh+2:GOTO 520
600 '
610 '*****
620 '* Nun Auswertung und Ausgabe *
630 '*****
640 '

```

```

650 lesezeiger = VAL("&" + buffer$):CLS
660 PRINT "Verzeichnis der File-Eintraege (UN=Usernummer) ";
670 PRINT "Format: ";:GOSUB 1270:PRINT fo$:GOSUB 1270:PRINT
680 PRINT "Name EXT EN UN Name EXT EN UN Name EXT EN UN";
690 PRINT "Name EXT EN UN":PRINT tr$
700 blcnt=0:filename$=""
710 usernummer=PEEK(lesezeiger)
720 IF usernummer=&E5 THEN file$=ni$: ' File geloescht
730 IF usernummer<&16 THEN file$=pl$: ' File nicht geloescht
740 '
750 ' File-Namen holen
760 FOR i = lesezeiger+1 TO lesezeiger+12
770 z$=CHR$(PEEK(i))
780 filename$=filename$+z$
790 NEXT i
800 lesezeiger = lesezeiger+32
810 '
820 'Pruefung ob letzter File-Eintrag
830 IF LEFT$(filename$,1)=CHR$(&E5) THEN fz=65:GOTO 910
840 fz=fz+1
850 IF file$=pl$ THEN GOSUB 1270: ' File gueltig deshalb Text invertieren
860 PRINT LEFT$(filename$,8);" ";:PRINT RIGHT$(filename$,4);
870 PRINT " "HEX$(fz,2);:PRINT " "HEX$(usernummer,2)" ";
880 IF file$=pl$ THEN GOSUB 1270: ' File gueltig deshalb Text invertieren
890 PRINT " ";:filename$=""
900 '
910 IF fz<64 THEN 710: 'noch keine 64 Directory-Eintraege
920 PRINT:PRINT tr$
930 '
940 '*****
950 '* Alle Eintraege ausgegeben *
960 '*****
970 '
980 PRINT:INPUT "Eintragsnummer (EN) des zu rettenden Files (0=ende): ";f$
990 f$=LOWER$(f$):IF f$="" THEN END
1000 f$="&" + f$:fz=VAL(f$):filename$=""
1010 '
1020 lesezeiger=VAL("&" + buffer$)+(fz-1)*32
1030 '
1040 usernummer=PEEK(lesezeiger): merker =lesezeiger
1050 '
1060 'Namen holen
1070 FOR i = lesezeiger+1 TO lesezeiger+12
1080 z$=CHR$(PEEK(i))
1090 IF ASC(z$)>128 THEN z$=CHR$(ASC(z$)-64):GOTO 1090: ' Gesetzte Bits raus
1100 filename$=filename$+z$
1110 NEXT i
1120 '
1130 CLS:PRINT "Gerettet werden soll : "filename$;
1140 PRINT "Eintrags-Nr. : "fz:PRINT
1150 INPUT "Usernummer";usernummer
1160 POKE merker,usernummer
1170 fz=0
1180 flag=1:GOTO 450
1190 '
1200 '*****
1210 secform =form+sector:POKE buffermc,bufflow:POKE (buffermc+1),buffhigh

```

```

1220 POKE drivenmc,drive:POKE trackmc,track:POKE formsemc,seform
1230 POKE befehlmc,befehl:CALL mcstart: RETURN
1240 '
1250 '*****
1260 ' Schrift invertieren
1270 CALL &BB9C:RETURN

```

Zu diesem Programm fehlt nun nur noch der Hexlader, der das erforderliche Maschinenprogramm als Binärfile erzeugt.

#### 9.6.8 Listing: SECTOR.HEX

```

100 'SECTOR.HEX
110 '
650 a=&A600:e=&A62A:zb=1000:e=e+1
660 FOR i =a TO e:READ d$:IF LEFT$(d$,1)="/" THEN flag =1
670 IF (flag AND ps<>VAL(d$)) THEN PRINT"Fehler in Zeile "zb+1:END
680 IF (flag AND i=e) THEN 750
690 IF flag THEN i=i-1:zb=zb+1:ps=0:d$="":flag = 0:GOTO 730
700 d$="/"&d$:POKE i,VAL(d$):ps=ps+VAL(d$):
730 IF i < e THEN NEXT i
750 SAVE"sector.bin",b,&A600,&2A
760 END
1001 DATA 3A,02,A7,32,26,A6,21,22,A6,CD,D4,BC,22,23,A6,79,&068B
1002 DATA 32,25,A6,21,26,A6,5E,23,56,23,4E,2A,29,A6,DF,23,&052D
1003 DATA A6,C9,84,00,00,00,00,00,C1,00,90,&0344

```

Das Maschinenprogramm - das durch diesen Hexlader erzeugt wird - ist so kurz, daß wir diesmal das Assemblerlisting nicht mit abdrucken. Aber im nächsten Kapitel finden Sie einen Re-Assembler, mit dem Sie versuchen sollten, das Programm in einen "Quellcode" zurückzuverwandeln.

Wir hoffen, daß wir Ihnen mit den Programmen Anregungen für Ihre eigene "Diskettenhandhabung" gegeben haben und schließen damit dieses Thema seitens der Grundlagen ab.

## 10. Hilfen zur Maschinensprache-Programmierung

In diesem Kapitel wollen wir uns etwas eingehender mit Hilfsmitteln zur Maschinensprache-Programmierung befassen.

Um in Maschinensprache zu programmieren, bedient man sich sinnvollerweise eines Assemblerprogrammes. Für die Schneider CPC's gibt es bereits einige leistungsfähige Programme dieser Art im Handel, und es sind auch schon einige - mehr oder minder leistungsfähige - Programme dieser Art (zum Beispiel in Fachzeitschriften) veröffentlicht worden.

Da die CPC's die Möglichkeit bieten, externe ROMs anzuschließen, ist der Einsatz eines Maschinensprache-Entwicklungskits in einer ROM-Version sehr vorteilhaft, da diese nur wenig, des manchmal doch so kostbaren Speicherplatzes, wegnehmen. Die Besitzer von "Disketten-CPC's" haben unter den CP/M-Utilities aber auch einige Programme, die zur Maschinenprogramm-Entwicklung eingesetzt werden können.

Von solchen Programmen soll hier aber nun nicht die Rede sein, sondern wie eingangs schon erwähnt, von weiteren Hilfsmitteln.

Um sinnvoll und effektiv auf Maschinenebene programmieren zu können, sind genauere Kenntnisse der Computer erforderlich. Die CPC 464-Besitzer sind hier eigentlich gar nicht schlecht bedient, denn Literatur über das Betriebssystem (Intern von Data Becker, CPC INSIDE-OUT vom Huslik-Verlag und das Firmware-Handbuch von Schneider) ist ausreichend und dadurch auch eine gute Vielfalt an Informationen.

Für den CPC 664 und 6128 sieht es noch etwas magerer aus. Teilweise treffen die in den angeführten Büchern enthaltenen Informationen zwar auch zu, aber andererseits sind auch gravierende Unterschiede vorhanden. Daß in der Spezialliteratur für die beiden letzteren Computertypen teilweise die Mnemonic fehlt, ist ein großes Handicap und nicht gerade erfreulich für die Besitzer dieser beiden Rechnertypen, die auch etwas mehr mit ihrem Computer anfangen wollen, als nur mit fertigen Programmen zu arbeiten. Für diese soll das nun folgende Programm vor allem gedacht sein.

Im Prinzip handelt es sich um einen normalen Disassembler, der aber die Eigenschaft hat, sein "Wissen" automatisch beim Disassemblerlauf einzufügen.

Es handelt sich um einen "analysierenden" Disassembler. Gleich vorweg sei bemerkt, daß mit einem Vorläufer dieses Programmes

viele Detail-Unterschiede bei den verschiedenen CPC-Typen herausgefunden wurde. Die Tabelle in Anhang 1 ist z.B. ein Ergebnis hiervon und dient gleichzeitig dem Programm als Grundlagenwissen.

#### 10.1 Was macht ein Disassembler?

Nun, er beginnt bei einer ihm mitgeteilten Anfangsadresse den Speicher Byte für Byte (bis zur Endadresse) durchzusehen und gibt die - den Inhalten entsprechenden - Maschinenbefehle als mnemonischen Code aus. Im Gegensatz zu einem Programm, welches nur die Speicherinhalte darstellt (z.B. MONI bis 3), teilt ein Disassembler also auch den Maschinencode mit.

Dies ist in etwa mit dem Basicbefehl LIST zu vergleichen. Wie Sie wissen, können Sie ein Basic-Programm durchaus auch mittels eines "Peekmonitors" ansehen, nur beim analysieren wird es etwas schwierig, den die nackten Speicherinhalte sind schlecht "lesbar".

Der Befehl LIST ist also normalerweise der bessere und bequemere Weg, sich ein Programm anzusehen, wenn man nicht gerade nach irgendwelchen "versteckten" Zeichen sucht oder sich mit der Ablage eines Basicprogrammes selbst beschäftigen will.

Zurück zum Disassembler.

Wird ein derartiges Programm gestartet, so gibt es im Regelfalle als erstes die Anfangsadresse aus. In den meisten Fällen auch noch die Hexwerte der Speicherstellen und die sogenannte Mnemonic, also eine für den Menschen verständlichere Form der Maschinenbefehle.

Oft erfolgt auch noch die Ausgabe der ASCII-Zeichen. Gerade die letztere Anzeige ist ein sehr positiver Punkt, denn dadurch erkennt man meist sehr schnell, ob man nun wirklich echten Maschinencode vor Augen hat, oder aber irgendwelche Tabellen, die ja ebenfalls in Programmen vorhanden sein können.

So, damit wäre prinzipiell geklärt, was ein normaler Disassembler ist. Was ist aber ein "analysierender" Disassembler?

Im Gegensatz zum vorher angesprochenen Assembler wird dieser mit "Wissen" versehen, also "intelligent" gemacht und kann mittels dieses zusätzlichen Wissens bestimmten Werten weitere Informationen zufügen. Das heißt, werden einem Disassembler beispielsweise die Einsprünge der Firmware-Routinen mitgeteilt, so kann er immer dann, wenn eine dieser Adressen angesprochen wird, bereits in einem kurzen Klartext weitere Hinweise geben. Dadurch wird ein Disassemblerlisting besser lesbar.

Weiter oben wurde schon angeführt, daß z.B. in einem Buch zwar die Kommentare zu den ROM-Routinen enthalten sind, aber die Mnemonic fehlt. Die Hintergründe hierfür hängen mit dem Firmware-Copyright zusammen.

Leider ist es sehr mühsam, mittels Buch und Disassembler am Bildschirm zu arbeiten. Sinnvoller ist es dann vielleicht schon, sich ein eigenes kommentiertes ROM-Listing zu erstellen. Und genau hierzu, soll der "analysierende" Disassembler dienen.

Nur ist es aber so, daß Sie diesen selbst erst einmal "schlau" machen müssen. Die "Grundintelligenz" können Sie diesem Buch entnehmen, aber leider nicht alles. Einfacher wäre es sicher, ein entsprechend kommentiertes Originallisting zu haben. Vielleicht gibt es dieses bis zum Erscheinen dieses Buches schon. Trotzdem haben Sie dann aber nicht "mit Zitronen gehandelt", wenn Sie das Programm abgetippt haben, denn es kann ja auch zur Analyse des Disketten-ROMs, oder anderer Sideway-ROMs dienen und soll Ihnen vor allem einen Weg aufzeigen, wie man sich, wenn Unterlagen fehlen, auch selbst weiterhelfen kann.

Da das nachfolgende Programm fast vollständig in Basic geschrieben wurde, können Ergänzungen und Änderungen leicht von Ihnen selbst durchgeführt werden.

Auch dieses Programm ist - so wie auch einige andere in diesem Buch - modular aufgebaut. In diesem Falle sind die Module sequentielle Dateien, die verschiedenes Wissen enthalten. Viele Variablen haben wir also ausgelagert.

Diese sequentiellen Dateien können deshalb auch bei anderen, so z. B. bei Ihren eigenen Programmentwicklungen zum Einsatz kommen.

Das heißt, daß viele Variablen nicht als Datastatements im Programm stehen, sondern als Stringvariable eingelesen werden.

Der erste Teil, der nicht im Programm selbst enthalten ist, sind die Z80-Befehlscodes. Diese werden durch einen gesonderten Programmteil als ASCII-File entweder auf die Diskette oder Kassette geschrieben. Und dies ist der Sinn des nachfolgenden Hilfsprogrammes.

#### 10.1.1 Listing: Z80CODE

```
100 'z80code
110 '
120 MODE 2:PRINT"Dieses Programm schreibt die Mnemonic des Z80 als
130 PRINT"ASCII-FILE auf Kassette oder Diskette !":PRINT
140 PRINT"Zu Testzwecken kann die Ausgabe zunächst auch nur auf dem
150 PRINT"Bildschirm erfolgen.":PRINT
160 PRINT"Wird dies gewünscht, so ist auf die nachfolgende Frage
170 PRINT"mit N zu antworten!
180 PRINT
190 PRINT"Soll das File MNEMONIC.Z80 geschrieben werden (J/N) ?":PRINT
200 x$=INKEY$:IF x$="" THEN 200 ELSE IF x$="j" THEN outp = 9
210 '
220 IF outp=9 THEN OPENOUT"mnemonic.z80"
230 DIM r$(7),rs$(8),bs$(3),al$(7),rm$(255)
240 '
250 PRINT"Register, Rotate-, Schiebe-, arithm. + log. Befehle
260 FOR i = 0 TO 7
```

```

270 READ r$(i),rs$(i),al$(i)
280 WRITE #outp,r$(i)
290 WRITE #outp,rs$(i)
300 WRITE #outp,al$(i)
310 DATA B,RLC,ADD,C,RR,ADC,D,RL,SUB,E,RR,SBC,H
320 DATA SLA,AND,L,SRA,XOR,(HL),???,OR,A,SRL,CP
330 NEXT i
340 '
350 PRINT"Bitoperationen
360 FOR i = 1 TO 3:READ bo$(i)
370 WRITE #outp,bo$(i)
380 DATA BIT,RES,SET
390 NEXT i
400 '
410 PRINT"der Rest der Mnemonics
420 FOR i = 0 TO &FF:READ rm$(i)
430 'in=INSTR(rm$(i),"")
440 'IF in THEN MID$(rm$(i),in)="/"
450 WRITE #outp,rm$(i)
460 IF rm$(i)="-1" THEN i=&9F:GOTO 470
470 NEXT i
480 '
490 DATA NOP,"LD BC,nn","LD (BC),A",INC BC,INC B,DEC B,"LD B,n",RLCA
500 DATA "EX AF,AF'", "ADD HL,BC", "LD A,(BC)",DEC BC,INC C,DEC C,"LD C,n"
510 DATA RRCA,DJNZ e,"LD DE,nn","LD (DE),A",INC DE,INC D,DEC D,"LD D,n"
520 DATA RLA,JR e,"ADD HL,DE","LD A,(DE)",DEC DE,INC E,DEC E,"LD E,n",RRA
530 DATA "JR NZ,e","LD HL,nn","LD (nn),HL",INC HL,INC H,DEC H,"LD H,n",DAA
540 DATA "JR Z,e","ADD HL,HL","LD HL,(nn)",DEC HL,INC L,DEC L,"LD L,n",CPL
550 DATA "JR NC,e","LD SP,nn","LD (nn),A",INC SP,INC (HL),DEC (HL)
560 DATA "LD (HL),n",SCF,"JR C,e","ADD HL,SP","LD A,(nn)",DEC SP,INC A
570 DATA DEC A,"LD A,n",CCF,"IN B,(C)"
580 DATA "OUT (C),B","SBC HL,BC","LD (nn),BC",NEG,RETn,IM 0,"LD I,A"
590 DATA "IN C,(C)","OUT (C),C","ADC HL,BC","LD BC,(nn)", "LD C,H",RETI
600 DATA "LD C,(HL)","LD R,A","IN D,(C)","OUT (C),D","SBC HL,DE","LD (nn),DE"
610 DATA "LD D,H","LD D,L",IM 1,"LD A,I","IN E,(C)","OUT (C),E","ADC HL,DE"
620 DATA "LD DE,(nn)","LD E,H","LD D,L",IM 2,"LD A,R","IN H,(C)","OUT (C),H"
630 DATA "SBC HL,HL","LD (nn),HL","LD H,H","LD H,L","LD H,HL",RRD,"IN L,(C)"
640 DATA "OUT (C),L","ADC HL,HL","LD HL,(nn)","LD L,H","LD L,L","LD L,HL",RLD
650 DATA "LD (HL),B","LD (HL),C","SBC HL,SP","LD (nn),SP","LD (HL),H"
660 DATA "LD (HL),L",HALT,"LD (HL),A","IN A,(C)","OUT (C),A","ADC HL,SP"
670 DATA "LD SP,(nn)","LD A,H","LD A,L","LD A,(HL)","LD A,A",-1
680 DATA LDI,CPI,INI,OUTI,*,*,*,*,LDD,CPD,IND,OUTD,*,*,*,*
690 DATA LDIR,CPIR,INIR,OTIR,*,*,*,*,LDDR,CPDR,INDR,OTDR,*,*,*,*
700 DATA RET NZ,POP BC,"JP NZ,nn",JP nn,"CALL NZ,nn",PUSH BC,"ADD A,n",RST 0
710 DATA RET Z,RET,"JP Z,nn",*, "CALL Z,nn",CALL nn,"ADC A,n",RST 1
720 DATA RET NC,POP DE,"JP NC,nn","OUT (n),A","CALL NC,nn",PUSH DE,"SUB n"
730 DATA RST 2,RET C,EXX,"JP C,nn","IN A,(n)","CALL C,nn",*, "SBC A,n",RST 3
740 DATA RET PD,POP HL,"JP PD,nn","EX (SP),HL","CALL PD,nn",PUSH HL,"AND n"
750 DATA RST 4,RET PE,JP (HL),"JP PE,nn","EX DE,HL","CALL PE,nn",*, "XOR n"
760 DATA RST 5,RET P,POP AF,"JP P,nn",DI,"CALL P,nn",PUSH AF,"OR n",RST 6
770 DATA RET M,"LD SP,HL","JP M,nn",EI,"CALL M,nn",*, "CP n",RST 7
780 CLOSEOUT

```

Beachten Sie bitte unbedingt die in diesem Programm angewandte Groß-/Kleinschreibweise, da diese für den späteren Programmlauf so benötigt wird!



## Erläuterungen zum Programm Z80CODE

Zeile 190 und 200 entscheiden darüber, ob das sequentielle File auf den Bildschirm (zu Kontrollzwecken) oder auf Kassette bzw. Diskette geschrieben werden soll. Die Variable outp wird deshalb - je nach Antwort - auf Stream 0 (=Bildschirm) belassen oder auf 9 (=Diskette-Kassette) gesetzt.

In Zeile 220 wird dann, wenn abgespeichert werden soll, das File eröffnet.

In Zeile 230 werden die verschiedenen Variablen dimensioniert.

In den Zeilen 250 bis 330 werden die Registernamen, die Rotier-, Schieber-, arithmetischen und logischen Befehle aus den in den Zeilen 310 und 320 stehenden Datastatements gelesen und auf den entsprechenden Stream ausgegeben.

Die Bitoperationen erleiden in den Zeilen 350 bis 390 das gleiche Schicksal.

Zu guter Letzt werden bei den mnemonischen Codes auch noch der restliche Teil so behandelt. (Zeilen 410 bis 470.) Allerdings gibt es hier gegenüber den anderen mnemonischen Codes einen etwas geänderten Ablauf.

Die Dimensionierung der Restmnemonic beträgt 255. Aber es klafft auch eine Lücke bei diesen Befehlen. Deshalb wird durch die am Ende der Zeile 670 stehende "-1" nach Bearbeitung des ersten Teiles, der Wert für die Fortführung auf &9F (Zeile 460) gesetzt.

Ganz wichtig ist die Zeile 780, denn durch den Closeout-Befehl werden die restlichen noch nicht ausgegebenen Daten geschrieben und das File ordnungsgemäß geschlossen.

Zwischenzeitlich noch weitere Informationen, weshalb die Aufteilung in Programm und ASCII-Files erfolgt.

Da das Gesamt-Programm sehr umfangreich ist, kann es zu Problemen kommen, wenn das Programm noch nicht fehlerfrei ist. Das bedeutet, daß die Ausbesserungen von Fehlern nicht so ohne weiteres geschehen kann. Würden nämlich alle erforderlichen Stringvariablen in Datazeilen stehen, dann würde nach einem Programmabbruch die Korrektur einer Zeile zunächst nicht mehr möglich sein. Jede Änderung im Programm würde der CPC mit der Meldung STRING SPACE FULL quittieren! (Diesem kann aber durch Eingabe von CLEAR abgeholfen werden.)

Viel wichtiger erscheint uns aber, daß Sie wirklich ohne große Probleme das Programm zum Laufen zu bringen. Aus diesem Grunde können Sie durch die ebenfalls abgedruckten Prüfprogramme alle Variablenfiles noch einmal überprüfen, damit Sie sicher sind, daß in diesen keine Fehler stecken.

Da wir eben schon die Fehlermeldung: STRING SPACE FULL angesprochen haben wollen wir in diesem Zusammenhang auch gleich auf die GARBAGE-COLLECT-ROUTINE hinweisen.

Kurz zur Erklärung des Sinnes der Garbage-Collect Routine und deren Auswirkungen. Während des Abarbeitens von Programmen werden oft viele Strings erzeugt, die durch nachfolgende Veränderungen nicht mehr gültig sind, da den Variablennamen andere Inhalte zugeordnet wurden und die neuen Inhalte nicht etwa die ehemaligen "Ablageplätze" im RAM-Speicher überschreiben, sondern an anderen Stellen im Bereich des "Stringstacks" abgelegt werden. Dadurch verbleiben im Bereich des Stringspeichers "Stringleichen", man spricht auch von den sogenannten "Müllstrings". Diese werden aber meist erst dann beseitigt, wenn dem Computer der Platz knapp wird und genau dieser Vorgang kostet Zeit.

Ist der Speicher nun durch das Programm sehr knapp, dann kostet diese "Müllbeseitigung" sehr, sehr viel Zeit und der CPC "steht" und ist je nachdem Sekunden, Minuten, oder wie einer der Verfasser es mit einem anderen Computer erlebt hat, sogar stundenlang beschäftigt, nur um die erforderlich gewordene Bereinigung des Stringspeichers durchzuführen.

Vor allem CPC 464-Besitzer werden dies auch in Verbindung mit dem Disassembler zu spüren bekommen, denn dieser CPC hat eine andere Routine zur Beseitigung der nicht mehr benötigten Strings und verhält sich deshalb auch anders als der CPC 664 und 6128. Der CPC 464 benötigt wesentlich mehr Zeit, wenn die Garbage Collection aktiv wird.

Der Eindruck während der "Beseitigung" von Müllstrings ist für den Betrachter der, daß der Rechner "steht" oder anscheinend "abgestürzt" ist. Dabei hilft dann auch kein Betätigen der ESC-Taste, denn aus einer Firmwareroutine gibt es normalerweise kein Entkommen.

Durch einen Fehler im abgetippten Programm könnte (muß nicht) diese Routine "zuschlagen" und Wartezeiten von oft mehreren Minuten Dauer würden Ihnen eventuell die Lust am Abschreiben von Listings nehmen. Die Arbeitsweise der Garbage Collect Routine können Sie sich mit dem Programm CPC-UHR ansehen.

Wieder zurück zum Disassembler. Einigen Lesern wird beim Abtippen oder Studium des nachfolgenden Programmes READER nun vielleicht auffallen, daß die Zeilen 430 und 440 nicht am aktiven Programmaufbau beteiligt sind. Dies hat einen besonderen Grund, denn mit diesen Zeilen können Sie sich einmal den Unterschied zwischen dem Befehl WRITE und PRINT verdeutlichen.

Ersetzen Sie einfach einmal die WRITE-Befehle des Programmes Z80CODE durch Print und prüfen dann mit dem Prüfprogramm READER was wieder eingelesen wird. Dann sehen Sie, daß nicht das eingelesen wird, was Sie eigentlich erwarten. Jedes Komma wirkt nämlich bei dieser Art der Abspeicherung als ein Trennzeichen (Delimiter) und würde bewirken, daß alle mnemonischen Befehle, die ein Komma enthalten, getrennt wurden.

Deshalb sollten Sie zusätzlich zu den ausgetauschten WRITE-Befehlen (in den Zeilen 280, 290, 300, 370, 450) dann die Zeilen 430 und 440 durch entfernen der Semikolons nach der Zeilennummer "aktivieren".

Dadurch wird dann ein in der Mnemonic vorkommendes Komma in einen Schrägstrich umgewandelt. In diesen beiden Zeilen wird auch die zweite Möglichkeit der Benutzung von MID\$ gezeigt, die z. Beispiel keinen Mullstring anfallen läßt.

Beim Programm READER muß die ebenfalls schon vorhandene aber noch nicht arbeitende Gegenroutine dann ebenfalls aktiviert werden.

Damit Sie dies nun alles nachvollziehen und auch die korrekte Abspeicherung testen können, hier nun das Programm READER.

### 10.1.2 Listing: READER

```

100 MODE 2:WINDOW #0,1,80,3,24:WINDOW #1,1,80,1,2
110 MODE 2:PRINT#1,"READER - Testprogramm fuer MNEMONIC.Z80
120 OPENIN "mnemonic.z80":WINDOW #0,1,80,3,24:WINDOW #1,1,80,1,2
130 '
140 PRINT#1,"Register, Rotate- u. Schiebepfeile, arithm. + log. Operationen
150 FOR i = 0 TO 7
160 INPUT #9,r$(i),rs$(i),al$(i)
170 PRINT r$(i),rs$(i),al$(i)
180 NEXT i:GOSUB 340
190 '
200 PRINT #1,"Bitoperationen
210 FOR i = 1 TO 3:INPUT #9,bo$(i):PRINT bo$(i)
220 NEXT i:GOSUB 340
230 '
240 PRINT #1,"Rest der Mnemonics
250 FOR i = 0 TO &FF:INPUT #9,rm$
260 'in=INSTR(rm$,"/")
270 'IF in THEN MID$(rm$,in)=", "
280 IF rm$="-1" THEN i=&9F:GOTO 310
290 PRINT TAB (x)rm$;
300 x=x+18:IF x>81 THEN x=0
310 NEXT i:PRINT
320 CLOSEIN
330 END
340 FOR i = 1 TO 500:NEXT i:CLS:CLS#1:RETURN

```

Zu diesem Programm brauchen Sie wahrscheinlich keine großen Erklärungen, es verhält sich ähnlich wie das vorhergehende Programm, nur daß Daten nicht aus Datazeilen gelesen werden, sondern vom Massenspeicher. Da die gelesenen Daten auch kurz auf dem Schirm dargestellt werden, kann oberflächlich überprüft werden, ob diese in etwa stimmen. Zur genaueren Prüfung könnte in Zeile 340 beispielsweise der Befehl CALL &BB18 eingefügt werden, wodurch das Programm dann im Single-Step- (Einzelschritt-) Mode arbeiten würde und Sie ausreichend Zeit zur Überprüfung hätten.

Als nächstes werden die Firmware-Informationen genauso behandelt. Auch diese werden in einem ASCII-File abgelegt und hierzu dient das nächste Programm. Allerdings ist dieses nicht komplett, sondern Sie müssen die Tabelle in Anhang 1 noch hinzufügen, denn dies ist ein Teil der "Intelligenz", mit der der Disassembler versehen werden soll.

Da diese Tabelle mit Programmzeilen versehen ist, brauchen Sie die Datazeilen nur in der angegebenen Weise abzutippen und an das nun folgende Listing anzuhängen.

#### 10.1.3 Listing: Firmjump

```

100 'FIRMJUMP (245 Einsprungadressen fuer 664,6128 - 232 bei 464)
110 '
120 MODE 2:WINDOW #1,1,80,1,4:WINDOW #0,1,80,5,20:tr$=STRING$(79,"-")
130 PRINT #1,"Programmname: FIRMJUMP"
140 PRINT #1,"Funktion : Saven der Firmwarespruenge als Ascii-File"
150 PRINT #1,tr$
160 '
170 anz=245:DIM fwj$(anz),kom$(anz)
180 READ a$,b$
190 IF a$<>"p" THEN z = z + 1:fwj$(z)=a$:kom$(z)=b$
200 IF a$="p" THEN a$="*****"
210 IF b$="ende" THEN GOTO 240
220 PRINT a$,b$
230 GOTO 180
240 PRINT:PRINT"Ich speichere ab"
250 OPENOUT"fjumps"
260 FOR i = 1 TO anz
270 PRINT #9,fwj$(i)
280 PRINT #9,kom$(i)
290 NEXT
300 CLOSEOUT:MODE 2:PRINT"Fertig":END

```

In Zeile 170 ist die Anzahl anzupassen, wenn Sie weniger oder mehr Firmwaresprünge angehängt haben.

Auch hierfür eine kleine Prüfroutine, welche hilft, die korrekte Abspeicherung zu testen.

Dieses Programm unterstützt bei der Überprüfung, ob das File FJUMPS korrekt abgespeichert wurde.

#### 10.1.4 Listing: FJLESE

```

100 'FJLESE
110 '
120 MODE 2:WINDOW #1,1,80,1,7:WINDOW #0,1,80,8,25:tr$=STRING$(79,"-")
130 PRINT #1,"Programmname: FJLESE"
140 PRINT #1,"Funktion : Test auf korrekte Speicherung von FJUMPS"
150 PRINT#1,"Bitte anhand der Tabelle ueberpruefen !"
160 PAPER #1,1:PEN #1,0
170 PRINT#1,"Die Leertaste zum Programmlauf gedrueckt halten !"
180 PAPER #1,0:PEN #1,1
190 PRINT #1,tr$:PRINT #1,"Adresse      Kommentar"
200 LOCATE #1,60,6:PRINT #1,"lfd. Nr.:";
210 PRINT #1,tr$
220 OPENIN"fjumps"
230 LOCATE #1,40,6:PRINT #1,typ$
240 FOR i = 1 TO 245

```

```

250 INPUT #9,adr$,kom$
260 PRINT adr$,kom$:LOCATE #1,69,6:PRINT#1,i;
270 x$=INKEY$:IF x$<>" " THEN 270
280 NEXT:CLOSEIN
290 PRINT CHR$(7)
300 PRINT"Programmlauf beendet

```

(Wie Sie sehen, variieren wir manchmal etwas mit dem Programmaufbau, um Sie mit verschiedenen Möglichkeiten bekannt zu machen.)

Nun müssen wir die letzten, "weisemachenden" Informationen noch als ASCII-File erzeugen. Hierzu dient nun das nächste Programm mit dem ebenfalls folgenden Prüfprogramm.

An das Programm SAGEN müssen Sie allerdings noch die Datazeilen aus dem Anhang 2 anhängen. Wer anfangs noch Scheu davor hat, diese langen Listen in seinen CPC einzugeben, der kann natürlich seinen Disassembler mit weniger Intelligenz ausrüsten. Bei diesem Programmteil beispielsweise steht in der Zeile 260 die Anzahl der Informationen, mit denen der CPC versorgt werden soll. Wenn Sie diese verringern, dann brauchen Sie auch nur die dann gewählte Anzahl an Datastatements anzufügen. Gleiches gilt sinngemäß auch für die Anzahl der Firmware-Einsprünge. Daß die Prüfprogramme dann natürlich auch entsprechend abgeändert werden müssen, versteht sich von selbst. Doch hier nun das weitere Programm.

#### 10.1.5 Listing: SAGEN

```

100 ' SAGEN Programm zur Erzeugung des ASCII-FILES sysadres
110 '
120 MODE 2:WINDOW #0,1,80,5,25:WINDOW #1,1,80,1,4
130 typ=PEEK(&BD71):tr$=STRING$(79,"-")
140 IF typ = &E8 THEN Typ$="CPC 464":w=3
150 IF typ = &55 THEN Typ$="CPC 664":w=2
160 IF typ = &14 THEN Typ$="CPC 6128":w=1
170 IF typ$="" THEN PRINT"unbekannter CPC - Programmlauf unmöglich":END
180 '
190 PRINT #1,"Sie haben einen "typ$" !"
200 PRINT #1,"ASCII-File sysadres wird fuer diesen Typ gespeichert !"
210 OPENOUT"sysadres"
220 '
230 PRINT #1,"6128 664 464 Funktion";
240 PRINT #1,TAB (60)"geschrieben wird:
250 PRINT #1,tr$:PRINT #9,typ$
260 FOR i = 0 TO 400
270 READ wahl$(1),wahl$(2),wahl$(3),kom$
280 IF wahl$(2)="" THEN wahl$(2)=wahl$(1)
290 PRINT wahl$(1)" "wahl$(2)" "wahl$(3)" "kom$;PRINT TAB(65),wahl$(w)
310 NEXT i
311 FOR i = 0 TO 400
315 WRITE #9,wahl$(w):PRINT #9,kom$
316 NEXT i
320 PRINT tr$:CLOSEOUT:END

```

Obwohl die Systemadressen der drei CPC's unterschiedlich sind, kann ein- und dasselbe Programm verwendet werden, denn durch Zeile 130 wird erkannt, welcher CPC-Typ vorliegt. Die durch diese Programmzeile abgefragte Speicherstelle enthält nämlich je nach Typ unterschiedliche Werte, die dann in den nächsten Zeilen zur Auswertung kommen. Wir hätten auch eine andere Adresse nehmen können, die Auswahl dieser erfolgte willkürlich. Dadurch haben wir vermieden, mittels einer Maschinenspracheroutine die Basic-version aus dem ROM abfragen zu müssen.

Zu obigen Programm sei noch gesagt, daß es eigentlich einfacher geht, das für den CPC-Typ erforderliche File zu erzeugen. Aber wir haben alle in diesem Buch enthaltenen Programme, die für alle CPC's gedacht sind, auch auf allen Maschinen getestet und wollten nicht drei Versionen schreiben sondern eben mit einer "variablen" Anpassroutine auskommen.

Wer also (obwohl es Gründe dafür gibt daß auch derjenige der nie mit einem anderen CPC arbeiten will die kompletten Listings und Statements übernimmt) nicht die Adressen für die anderen CPCs mit eingeben will, muß obiges Programm dementsprechend ändern.

Hier nun auch noch für das letzte ASCII-File des analysierenden Disassembler die entsprechende Prüfroutine.

#### 10.1.6 Listing: SALESE

```

100 'SALESE
110 '
120 MODE 2:WINDOW #1,1,80,1,7:WINDOW #0,1,80,8,25:tr$=STRING$(79,"-")
130 PRINT #1,"Programmname: SALESE
140 PRINT #1,"Funktion      : Test auf korrekte Speicherung von SYSADRES
150 PRINT#1,"Bitte anhand der Tabelle ueberpruefen !
160 PAPER #1,1:PEN #1,0
170 PRINT#1,"Die Leertaste zum Programmlauf gedruickt halten !"
180 PAPER #1,0:PEN #1,1
190 PRINT #1,tr$:PRINT #1,"Adresse      Kommentar"
200 LOCATE #1,60,6:PRINT #1,"lfd. Nr.:";
210 PRINT #1,tr$
220 OPENIN"sysadres"
230 INPUT #9,typ$:LOCATE #1,40,6:PRINT #1,typ$
240 FOR i = 1 TO 360
250 INPUT #9,adr$,kom$
260 PRINT adr$,kom$:LOCATE #1,60,6:PRINT#1,i;
270 x$=INKEY$:IF x$<>" " THEN 270
280 NEXT:CLOSEIN
290 PRINT CHR$(7)
300 PRINT"Programmlauf beendet

```

So, nun fehlt nur noch der Basicteil, der dafür sorgt, daß alles richtig zusammenspielt. Nämlich der eigentliche Disassembler, der dann im nächsten Abschnitt auch noch zu einem Reassembler umfunktioniert wird.

Für Betreiber von Kassettenlaufwerken nun aber auch gleich die Information, in welcher Reihenfolge, welche Programmteile auf der Kassette stehen müssen, damit alles reibungslos klappt. Der Diskettenbenutzer hat diese Problematik durch den direkten Programmzugriff ja nicht.

#### Die Programmreihenfolge auf einer Kassette

- |                  |                                   |
|------------------|-----------------------------------|
| 1.) ANADIS       | (der Basisteil des Disassemblers) |
| 2.) MNEMONIC.Z80 | (wird durch Z80CODE erzeugt)      |
| 3.) FJUMPS       | (wird durch FIRMJUMP erzeugt)     |
| 4.) SYSADRES     | (wird durch SAGEN erzeugt)        |

Wenn also auch der Basisteil korrekt abgeschrieben wurde, dann kann mit diesem Programm gearbeitet werden. Die Ausgabe ist im Programm bisher nur auf den Bildschirm zugeschnitten. Aber einer Erweiterung für Ausgaben auf den Drucker, Diskette oder auf Kassette steht nichts im Wege. Denken Sie aber daran, je schlauer Sie das Programm machen, um so weniger Speicherplatz haben Sie, und der kann knapp werden. Aus diesem Grunde ist es vielleicht sinnvoll, die Ausgaben nur abzuspeichern und später zu drucken.

Hier nun der Basisteil und anschließend erfahren Sie dann noch mehr über das Programm selbst.

#### 10.1.7 Listing: ANADIS

```

100 ' Analysierender Disassembler fuer CPC 464 bis 6128
110 '
120 MODE 2:MEMORY &A4FF:PRINT"Analysierender Disassembler
130 '
140 'Array fuer Mnemonic dimensionieren
150 DIM r$(7),rs$(8),bo$(3),ai$(7),rm$(255)
160 '
170 'anz = Anzahl der bereits erkannten Adressen
180 anz1=245:anz2=409:anz=anz1+anz2
190 '
200 'Array fuer Adressen und Firmware dimensionieren
210 DIM fw$(anz),te$(anz):trenn$=STRING$(79,"-")
220 '
230 'LESERROUTINE (AUCH FUER ROMS) IN SPEICHER POKEN
240 FOR i = 0 TO 13:READ a:POKE &A500+i,a:NEXT i
250 DATA &df,&04,&a5,&c9,&07,&a5,&fc,&3a,&ff,&ff,&32,&0e,&a5,&c9
260 '
270 'outp = Ausgabestream sel = ROM/RAM-Select
280 outp=0:sel=&FC:POKE &A506,sel
290 '
300 PRINT"Moment bitte - ich lese die Mnemonics ein
310 OPENIN "mnemonic.z80":WINDOW #0,1,80,3,24:WINDOW #1,1,80,1,2
320 '
330 'Register, Rotate- u. Schiebepfeile, arithm. + log. Operationen einlesen
340 FOR i = 0 TO 7
350 INPUT #9,r$(i),rs$(i),ai$(i)
360 'PRINT r$(i),rs$(i),ai$(i)
370 NEXT i:PRINT
380 '

```

```

390 'Bitoperationen einlesen
400 FOR i = 1 TO 3:INPUT #9,bo$(i)
410 'PRINT bo$(i)
420 NEXT i:PRINT
430 '
440 'Rest der Mnemonics einlesen
450 FOR i = 0 TO &FF:INPUT #9,rm$(i)
460 in=INSTR(rm$(i),"/")
470 IF in THEN MID$(rm$(i),in)=", "
480 IF rm$(i)="-1" THEN i=&9F:GOTO 500
490 'PRINT rm$(i)
500 NEXT i:PRINT
510 CLOSEIN:CLS #1:PRINT #1,"Mnemonics eingelesen, lese bekannte Adressen
520 '
530 OPENIN"fjumps":PRINT #1,"Firmwarejumps
540 FOR i = 1 TO anz1:INPUT #9,fw$(i),te$(i)
550 'PRINT fw$(i),te$(i)
560 NEXT i:PRINT:CLOSEIN
570 '
580 OPENIN"sysadres
590 INPUT #9,typ$:PRINT#1,"Systemadressen":PRINT #1,"Computer-Typ: "typ$
600 FOR i = (anz1+1) TO anz:INPUT #9,fw$(i),te$(i)
610 'PRINT fw$(i),te$(i)
620 NEXT i:CLS
630 '
640 'Kopfleiste ausgeben
650 PRINT #1,"Adr Bytes      ASCII      MNEMONIC      Kommentar
660 PRINT #1,STRING$(79,"*")
670 '
680 'Speicherbereiche und Betriebsmode holen
690 READ x$:IF x$="ende" THEN END
700 x=VAL("&"&x$):IF x<0 THEN anf=x+2^16 ELSE anf=x
710 READ x$:x=VAL("&"&x$):IF x<0 THEN ende=x+2^16 ELSE ende=x
720 IF anf>ende THEN PRINT"FEHLER":STOP ELSE start=anf:zeig=start
730 '
740 READ bmode$:' Betriebsmode des Programmes
750 IF bmode$="t" THEN GOTO 1700:' Tabelle
760 IF bmode$<>"p" THEN PRINT"fehler":STOP
770 'die Betriebsmode stehen ab 10000
780 '-----
790 '-----
800 'START
810 a1$=HEX$(zeig,4):ind1 = 0:start=zeig
820 a2$="":a3$="":aus$=""
830 IF start>ende THEN 690
840 '-----
850 GOSUB 1600:GOSUB 1000:' Byte lesen und analysieren
860 IF ind1 THEN 1330
870 IF pwert=207 OR pwert=215 THEN of$=of$+",nn"
880 IF pwert=223 OR pwert=239 THEN of$=of$+",nn"
890 IF INSTR(of$,"n") THEN 1460 ELSE IF INSTR(of$,"e") THEN 1540
900 in=INSTR(of$," "):IF in=0 THEN aus$=of$:GOTO 930:'kein Leerzeichen im Code
910 aus$=LEFT$(of$,in-1)+" "+RIGHT$(of$,LEN(of$)-in)
920 '-----
930 FOR i = 1 TO anz:IF INSTR(aus$,fw$(i)) THEN te$=te$(i):i=anz+1:REM ***
940 NEXT i:REM ***
950 PRINT #outp,a1$+" "+a2$;TAB(20);a3$;TAB(27);aus$;TAB(48);te$:te$=""

```



```

960 IF INSTR(aus$, "J") OR INSTR(aus$, "RET") THEN PRINT#outp, trenns
970 IF start <= ENDE THEN 810
980 GOTO 690
990 '-----
1000 REM Byte analysieren bzw. interpretieren
1010 IF indi THEN 1030
1020 IF pwert=&DD THEN 1290 ELSE IF pwert =&FD THEN 1290
1030 IF pwert=&ED THEN 1250 ELSE IF pwert =&CB THEN 1180
1040 GOSUB 1140
1050 IF b76=1 THEN 1070 ELSE IF b76=2 THEN 1090
1060 GOTO 1110
1070 IF pwert=&76 THEN of$=rm$(pwert):GOTO 1120
1080 of$="LD "+r$(b53)+", "+reg$:GOTO 1120
1090 IF b53<2 OR b53=3 THEN x$=" A," ELSE x$=" "
1100 of$=a$(b53)+x$+reg$:GOTO 1120
1110 of$=rm$(pwert)
1120 RETURN
1130 '-----
1140 REM pwert splitten
1150 b20=(&7 AND pwert)/2^0:b53=(&38 AND pwert)/2^3:b76=(&C0 AND pwert)/2^6
1160 reg$=r$(b20):RETURN
1170 '-----
1180 REM Byte =cb
1190 GOSUB 1600:IF indi THEN br=pwert:GOSUB 1600
1200 GOSUB 1140
1210 IF b76=0 THEN of$=rs$(b53)+" "+reg$
1220 IF b76<>0 THEN of$=bc$(b76)+STR$(b53)+", "+reg$
1230 RETURN
1240 '-----
1250 REM Byte =ed
1260 GOSUB 1600:IF pwert<&40 OR pwert>&BF THEN of$="???":RETURN
1270 GOTO 1110
1280 '-----
1290 REM IX oder IY
1300 indi=1:IF pwert=221 THEN ind$="IX" ELSE ind$="IY"
1310 GOSUB 1600:GOTO 1000
1320 '-----
1330 REM indizierte Befehle
1340 in=INSTR(of$, "HL")
1350 IF in=0 THEN of$="???":GOTO 900 ELSE IF INSTR(of$, "(HL)") THEN 1390
1360 IF of$="EX DE,HL" THEN of$="???":GOTO 900
1370 IF of$="ADD HL,HL" THEN of$="ADD "+ind$+", "+ind$:GOTO 900
1380 of$=LEFT$(of$, in-1)+ind$+RIGHT$(of$, LEN(of$)-in-1):GOTO 870
1390 IF LEFT$(of$, 2)="JP" THEN 1380
1400 IF zeig-start<3 THEN GOSUB 1600:br=pwert
1410 IF br>=&80 THEN br$=STR$(br-&FF-1):GOTO 1430
1420 br$=" "+HEX$(br, 2)
1430 ind$=ind$+br$:GOTO 1380
1440 '-----
1450 '
1460 REM nn oder n einsetzen
1470 in=INSTR(of$, "nn"): IF in THEN 1490 ELSE in=INSTR(of$, "n"):GOSUB 1600
1480 of$=LEFT$(of$, in-1)+HEX$(pwert, 2)+RIGHT$(of$, LEN(of$)-in):GOTO 900
1490 GOSUB 1600:lbyte=pwert:GOSUB 1600:'zwei Byte lesen fuer Wert (we)
1500 we=pwert*256+lbyte
1510 of$=LEFT$(of$, in-1)+HEX$(we, 4)+RIGHT$(of$, LEN(of$)-in-1):GOTO 900
1520 '-----

```

```

1530 '
1540 REM e einsetzen
1550 in=INSTR(of$, "e"):GOSUB 1600:IF pwert>=80 THEN pwert=pwert-&FF-1
1560 pwert=pwert+2:x$=HEX$(zeig+pwert-2,4)
1570 of$=LEFT$(of$,in-1)+x$+RIGHT$(of$,LEN(of$)-in):GOTO 900
1580 '-----
1590 '
1600 REM Byte lesen
1610 'Berechnung Page und Pageadresse
1620 high=INT(zeig/256):low=zeig-high*256
1630 POKE &A508,low:POKE &A509,high:CALL &A500:pwert=PEEK (&A50E)
1640 zeig = zeig + 1
1650 a2$=a2$+HEX$(pwert,2)+" ":y=pwert:IF y<32 THEN y=32
1660 IF y>=125 THEN y = y-64:GOTO 1660:'Grafikzeichen ausblenden
1670 a3$=a3$+CHR$(y):RETURN
1680 '-----
1690 '
1700 'MONITORFUNKTION
1710 zeig =anf:merk =zeig
1720 FOR y = anf TO ende
1730 GOSUB 1600
1740 zz = zz + 1
1750 IF zz = 16 OR zeig>=ende+1 THEN 1770
1760 NEXT
1770 a1$="&"+HEX$(merk,4)
1780 PRINT #outp,a1$;" ";a2$;TAB(60);a3$:a2$="":a3$=""
1790 zz=0:merk = zeig
1800 IF zeig <ende THEN 1760
1810 PRINT#outp,trenn$
1820 GOTO 600
1830 '-----
18000 '-----
18010 REM -datas = bekannte adressbereiche (p=programm t= tabellen) -
18020 '
18030 'schon erkannte Adressbereiche: t=Tabelle, p=Programm
18040 DATA 0000,03a5,p,03A6,0590,t,0591,05c4,p,05c5,05e4,t,05e5,0687,p
18050 DATA 0688,06f8,t,06f9,0704,p,0705,0722,t,0723,0737,p,0738,0775,t
18060 DATA 0776,08dd,p,08de,0ab3,t,0ab4,0abf,p,08de,0ab3,t,0ab4,0adc,p
18070 DATA 0add,0ae8,t,0ae9,0d98,p,0d99,0db8,t,0db9,1051,p,1052,1073,t
18080 DATA 1074,108c,p,108D,109E,t,109f,1473,p,1474,14D3,t,14d4,14d7,p
18090 DATA 14d8,14e0,t,14e1,15df,p,15e0,15eb,t,15ec,1766,p
18100 DATA 1767,1fff,p
18110 DATA ende

```

#### Eine Anregung für die Benutzer von Kassettenlaufwerken:

Bei allen OPENIN- und OPENOUT-Befehlen sollte vor dem Filenamen ein Ausrufezeichen geschrieben werden, denn dadurch entfällt dann die Aufforderung die jeweiligen Tasten am Kassettenrecorder zu betätigen. Bei Abspeichervorgängen, die mitten oder am Ende eines Programmes erfolgen, muß dann nicht extra gewartet werden und auch die Aufforderung die Kassettentasten zu drücken, entfällt. Die Tasten können Sie bereits vorher drücken.

Beim Test des Programmes auf dem CPC 464 fiel uns Ubrigens auf, daß die Garbage Collection und auch das "Umschauen" von Files

aus dem Einlesebuffer in die entsprechenden Speicherstellen für den Programmlauf wesentlich länger dauerte als bei den beiden "Geschwistern". Dies gilt auch bei angeschlossener Diskettenstation.

#### 10.1.8 Programmbeschreibung: ANADIS

Viele Teile kennen Sie schon aus den kleinen Hilfsprogrammen. Deshalb wird hier nun nur auf die wichtigen Zeilen hingewiesen.

Die Zeilen 240 und 250 erzeugen ein kleines Maschinenprogramm, mit dem es möglich ist, auch auf die ROMs des CPC zuzugreifen.

Die Zeile 280 bestimmt auf welchen Stream ausgegeben werden soll und außerdem auch, welche Konfiguration selektiert wird. (Siehe hierzu auch Programmbeschreibung MONI.BAS.

Bis Zeile 670 dürfte Ihnen alles bereits bekannt sein.

Neu ist dann aber der Sinn der Zeilen 740 bis 760. Wie Sie wissen, kann ein Maschinensprache-Programm auch Tabellen enthalten, also nicht nur reine Befehle die abzuarbeiten sind. Es wäre nun völliger Unsinn, wenn der Disassembler nun versuchen soll diese Tabellen zu "interpretieren", da kommt nur Unsinn heraus, auch wenn er mnemonische Codes ausgibt. Deshalb muß dem Disassembler gesagt werden, wann er disassemblieren soll und wann die Ausgabe einer Tabelle zu erfolgen hat.

Diese Daten stehen ab der Zeile 10000 in Datastatements.

"p" bedeutet Programm, also muß disassembliert werden und "t" heißt Tabelle.

Ab Zeile 800 beginnt dann der Start des eigentlichen Disassemblers. In a\$ wird die jeweilige Speicheradresse zwischengespeichert und für die Ausgabe gemerkt.

"indi" dient als Flag und signalisiert ob ein indizierter Befehl gefunden wurde.

Als erstes wird ab Zeile 1600 die Adresse an das Maschinenprogramm übergeben, um dann ein Byte zu lesen. Die beiden Pokebefehle in Zeile 1630 übergeben diese Adresse in LOW- und HIGH-Byte aufgeteilt. Der Call-Befehl bewirkt, daß die Maschinenroutine das in dieser Adresse enthaltene Byte in die Speicherstelle &A50E übergibt, welche dann von Basic aus durch den Peek-Befehl ausgelesen werden kann.

In Zeile 1640 wird der Adresszeiger um eins erhöht. In Zeile 1650 wird die Folge der Hexwerte in a\$ zusammengefasst. Da auch eine ASCII-Ausgabe erfolgen soll und Steuerzeichen den Bildschirm zu einem Chaos werden lassen würden, werden diese zu Spaces umgewandelt (Zeile 1650).

Falls jemand einen Drucker ansteuern will, würde die Tatsache, daß der CPC nur einen 7-Bit-Druckerausgang hat, Zeichen mit ge-

setzten achten Bits zu Problemen führen, deshalb geschieht in Zeile 1660 das Ausblenden derartiger Zeichen, sie werden in 7-Bit-Werte umgewandelt. In 1670 werden die ASCII-Zeichen zu einem String zusammengefasst.

Da Z80-Befehle aus bis zu vier Bytes bestehen können, muß nun nachdem das erste Byte geholt wurde, festgestellt werden, um welche Klasse des Befehls es sich handelt. Dies geschieht ab Zeile 1000. Ab diesen Zeilen wird also der Befehl analysiert und interpretiert. Was die einzelnen Prüfungen sollen und auch bewirken, würde hier zu weit führen. Interessenten sei an dieser Stelle das Buch "Programmierung des Z80" von Rodney Zaks, aus dem Sybex-Verlag empfohlen, in denen die Hintergründe, für die in diesem Programm angewendeten Abfragen und Auswertungen sehr ausführlich beschrieben sind. Dieses Buch ist derzeit eigentlich das Standardwerk für die Programmierung des Z80-Prozessors.

Auch andere Programmteile sind mit den Auswertungen der einzelnen Bits betraut und führen letztendlich dazu, daß der Disassembler erkennt, wieviele Bytes zu diesem Befehl gehören, ob rückwärts oder vorwärts gerechnet werden muß, ob ein- oder zweiwertige Zahlen zu den Befehlen gehören, ob es sich um einen indizierten Befehl handelt, welche Bits hierfür zuständig sind, usw. Da auch eine genaue Beschreibung dieser Auswertung viele Seiten benötigen würde, beschränken wir uns nur auf das Wichtigste.

Je nachdem, welcher Befehl vorliegt, werden in die "Zwischenmnemonic" z.B. die Buchstaben "e" (bei einer Verzweigung), "n" (bei einem Einbytwert) "nn" (bei einem Zweibytewert) eingefügt. Auch das Flag für indizierte Befehle spielt dabei eine Rolle, denn dieses bestimmt wiederum, wie weiter zu verfahren ist. Im Programm sind diese Abschnitte durch entsprechende REMarks erläutert. Interessant für Sie ist evtl. die Auswertung, ob eine "Zwischenmnemonic" einen dieser Buchstaben enthält. Dies wird durch den Befehl "INSTR(betroffener String, Maske)" festgestellt. Ist ein Teil des betroffenen Strings mit der Maske deckungsgleich, dann wird die Variable "in" mit dem entsprechenden Wert geladen, ab der die Gleichheit herrscht, denn an diese Stelle muß nachher der berechnete Wert oder die andere Befehlsform eingesetzt werden. Die gleiche Technik wird auch zur Auswertung für bekannte Firmware- und Systemadressen angewandt.

Nach Jump (JP), nach Jump Subroutine (JR) und nach Return (RET) wird bei der Ausgabe ein Trennstrich angefügt, da nach diesen Befehlen evtl. eine Routine abgeschlossen ist (Zeile 960). Dies kann die Übersichtlichkeit beim Lesen des disassemblierten Listings erhöhen.

Die reine Monitorfunktion (für Tabellen) finden Sie von Zeile 1700 bis 1820.

Durch weglassen der Zeilen 930 und 940 arbeitet das Programm als ganz normaler Disassembler mit Monitorfunktion und gewinnt dadurch enorm an Geschwindigkeit.

Die Datastatements ab Zeile 10000 gelten übrigens nur für den CPC 6128 und stellen einen Teil der von uns erkannten Bereiche dar.

So nun viel Spass beim analysierenden Disassemblieren und viel Erfolg bei den Analysen.

Sehen Sie sich vielleicht auch einmal die im Programm enthaltene Maschinenroutine an, Sie können Sie ja nun disassemblieren!

## 10.2 Aus dem Disassembler wird ein Reassembler

Wenn Sie des öfteren einmal Programme mit Maschinenroutinen aus Zeitschriften oder Büchern abtippen, dann fällt Ihnen wahrscheinlich das manchmalige Fehlen von kommentierten Assemblerlistings auf. Dies ist keine böse Absicht des Autors oder des Verlages, sondern geschieht einfach aus Platzgründen.

Das heißt in diesem Falle steht Ihnen nur ein Ladeprogramm mit den wenig aussagenden Datastatements zur Verfügung. Wollen Sie diesen Programmteil aber auch bei anderen Programmen einsetzen und ist dadurch dann auch noch ein Assemblieren für einen anderen Speicherbereich erforderlich, dann bleibt Ihnen im Regelfalle nichts anderes übrig, als mittels eines Disassemblers diesen Teil zu disassemblieren und auszudrucken um dann mit einem Editor ein Quellcodefile neu zu schreiben.

Dies muß aber nicht sein, denn mit ein paar kleinen Änderungen wird unser Disassembler ganz schnell zu einem Reassembler, der Ihnen ein - mit einem Assembler - verarbeitbares File erzeugt.

Um das Programm abzuändern laden Sie bitte den Basicteil ANADIS und entfernen dann einige Zeilen.

Deshalb nach dem Ladevorgang

```
DELETE 160 - 220
DELETE 530 - 670
DELETE 930 - 960
DELETE 10010 -
```

eingeben. Danach müssen einige Zeilen hinzugefügt werden. Auch dies können Sie automatisieren indem Sie nachfolgende Zeilen als Programm geschrieben haben und hinzumergen. Sie können sie natürlich auch direkt ins Programm schreiben.

### 10.2.1 Listing: REASS

```
100 ' Reassembler fuer CPC 464 bis 6128
120 MODE 2:MEMORY &865F:PRINT"Reassembler
125 PRINT:INPUT"Abspeichern ?":flag$:IF LOWER$(flag$)="j" THEN df=1
130 IF df = 1 THEN OPENOUT"reafile"
140 'Array fuer Mnemonic dimensionieren
150 DIM r$(7),rs$(8),bc$(3),al$(7),rm$(255)
240 FOR i =0 TO 13: READ a:POKE &8660+i,a:NEXT i
250 DATA &df,&64,&86,&c9,&67,&86,&fc,&3a,&ff,&ff,&32,&6e,&86,&c9
280 outp=0:sel=&7:POKE &8666,sel
510 CLOSEIN:CLS #1:PRINT #1,"Mnemonics eingelesen
685 READ x$:anf1 = VAL("&"+x$):IF anf1<0 THEN anf1=anf1+2^16
220
```

```

686 READ x$:endi = VAL("&"&x$):IF endi<0 THEN endi=endi+2^16
690 READ x$:IF x$="ende" THEN CLOSEDOUT:END
930 IF df = 1 THEN PRINT #9,"R"&a1$&" "&aus$
940 PRINT "R"&a1$&" "&aus$
1480 of$=LEFT$(of$,in-1)+"&"&HEX$(pwert,2)+RIGHT$(of$,LEN(of$)-in):GOTO 900
1505 IF HEX$(we,4)<HEX$(anf1) THEN 2000
1506 IF HEX$(we,4)>HEX$(endi) THEN 2000
1510 of$=LEFT$(of$,in-1)+"R"&HEX$(we,4)+RIGHT$(of$,LEN(of$)-in-1):GOTO 900
1560 pwert=pwert+2:x$="R"&HEX$(zeig+pwert-2,4)
1630 POKE &8668,low:POKE &8669,high:CALL &8668:pwert=PEEK (&866E)
1725 high=INT(zeig/256):low=zeig-high*256
1730 POKE &8668,low:POKE &8669,high:CALL &8668:pwert=PEEK (&866E)
1740 zeig = zeig + 1
1750 a2$=a2$&"&"&HEX$(pwert,2)
1760 zz=zz+1:GOTO 1780
1770 NEXT
1780 a1$="R"&HEX$(merk,4)
1790 IF df = 1 THEN PRINT #9,a1$&" defb "&a2$
1800 PRINT a1$&" defb "&a2$&a3$&"&a3$&"
1810 zz=0:merk=zeig
1820 IF zeig <=ende THEN 1770
1830 GOTO 690
2000 of$=LEFT$(of$,in-1)+"&"&HEX$(we,4)+RIGHT$(of$,LEN(of$)-in-1):GOTO 900
10010 'Grenzen
10020 DATA c000,c100
10030 'Adressbereiche p=disassemblieren t=tabelle
10040 DATA c000,c005,t,c006,c06f,p,c072,c0bf,t,c0c0,c0f9,p
10050 DATA ende

```

Nun haben Sie aus dem Disassembler einen Reassembler gemacht und können diesen abspeichern.

#### Erklärungen zum Programm

Wie bereits an anderer Stelle geschrieben, wird bei Quelltexten für Maschinenprogramme mit Labels gearbeitet. Sinn und Zweck dieser Labels ist, stellvertretend für noch nicht bekannte Adressen zu stehen. Diese Labels müssen für fast alle Assembler mit einem Buchstaben beginnen.

Deshalb können Speicheradressen nicht als Label benutzt werden. Setzt man diesen Adressen aber einen Buchstaben voraus, dann ist das Problem schon gelöst. Genau so arbeitet der Reassembler. Vor jeden disassemblierten Befehl setzt er dessen Startadresse mit einem "R" davor um darauf hinzuweisen, daß es sich um einen reassemblierten "Quell-" Code handelt.

Da aber Adressen, die außerhalb des Maschinenprogrammbereiches liegen, nicht mit diesem Vorzeichen versehen werden sollen und auch nicht dürfen, muß dem Programm mitgeteilt werden, daß es nur bei Adressen innerhalb des zu reassemblierenden Bereiches den Vorsatz "R" an die jeweiligen Sprungadressen usw. anfügen soll. Deshalb werden dem Programm in Zeile 10020 diese Grenzen mitgeteilt.

Ein großes Problem sind Tabellen, denn auf diese wird oft indiziert und manchmal auch direkt "mittenhinein" zugegriffen. Deshalb wird, um den Programmaufwand nicht zu hoch werden zu lassen, bei Tabellen jedes Byte mit einem Label versehen.

Dies kostet zwar enorm viel Speicherplatz, aber wenn die Tabellen nicht allzugroß sind, kann ein Maschinen-Programm von 2 KByte Länge durchaus bewältigt werden. Eine genaue Angabe kann nicht gemacht werden, da dies ja auch sehr stark davon abhängt, welche Befehlsklassen (1,2,3 und 4 Byte) im Programm vorkommen.

Da die meisten Maschinenprogramme im oberen Speicherbereich liegen, wurde die Adresse für das Zugriffsprogramm nach unten gelegt und der Basicspeicher auf &865F begrenzt (Zeile 120).

Damit haben Sie nun einige sehr starke Routinen zur Maschinensprache-Programmierung und aufgrund der Anstrengungen, die Sie dadurch beim Eintippen der Programme machen mußten, zur Erleichterung ein kürzeres Programm einer Uhr.

### 10.3 Die Garbage-Collect-Routine

Nachdem in diesem Kapitel die Garbage-Collect-Routine mehrmals angesprochen wurde, sollen Sie mittels eines kleinen Programmes auch sehen, wie diese "zuschlägt".

Über den Sinn dieser Routine hatten wir schon informiert. Doch nun soll an einem Beispiel gezeigt werden, wie man den Aufruf dieser Routine vermeiden kann.

Ein kleines Demonstrationsprogramm soll aufzeigen, wie dies geht.

#### 10.3.1 Listing: CPC-Uhr

```
100 MODE 1:PRINT"Demo-Programm fuer die Uhrzeitausgabe
110 menge = 150:DIM dummy$(menge):anzahl = -1:ti$=""
120 pi=3
130 FOR i=0 TO menge:dummy$(i)=STRING$(255,"*"):NEXT
140 LOCATE 1,pi:PRINT"Mit oder ohne Garbage collect ?"
150 PRINT"      (i= mit / 0 = ohne)"
160 x$=INKEY$:IF x$="" THEN 160 ELSE x=VAL(x$)
170 LOCATE 1,pi:PRINT"
180 IF x = 1 THEN PRINT"      (mit Garbage-Collect)
190 IF x = 0 THEN PRINT"      (ohne Garbage-Collect)
200 '
210 divisor=300.480769' entspricht ca. 300 ms
220 '
230 LOCATE 1,pi+4:INPUT"Zeitstring (Form: hhmmss)  ":zs$
240 '
250 'Werte fuer CPC 464
260 tim1=&B100:tim2=&B109:tim3=&B10A:tim0=&B107
270 '
280 'Kein CPC 464
290 IF PEEK(&AC01)<> &C9 THEN tim1=&B0B5:tim2=&B0B6:tim3=&B0B7:tim0=&B0B4
300 :
```

```

310 t1=VAL(LEFT$(zs$,2))'           Stunden
320 t2=VAL(MID$(zs$,3,2))'         Minuten
330 t3=VAL(RIGHT$(zs$,2))'         Sekunden
340 '
350 'Umrechnen fuer Takt
360 '
370 ti=divisor*(t3+60*t2+3600*t1)
380 ti=INT(ti/2^24):ti=ti-2^24*t1
390 t2=INT(ti/2^16):ti=ti-2^16*t2
400 t3=INT(ti/256):ti=ti-256*t3
410 '
420 ' Eingegebene Uhrzeit poken
430 POKE tim1,t3:POKE tim2,t2:POKE tim3,t1:POKE tim0,ti
440 '----- Eingabe beendet -----
450 LOCATE 30,pi+4:PRINT"          "
460 :
470 LOCATE 1,pi+4:PRINT"Die CPC-Uhrzeit ist:          "
480 LOCATE 1,pi+7:PRINT"Haeufigkeit der GC : "
490 LOCATE 1,pi+9:PRINT"Stringlocation      : "
500 LOCATE 1,pi+13:PRINT"  Garbage collect"
510 LOCATE 1,20:PRINT"E = Ende
520 '
530 'Mitternacht ? (24*3600*divisor)
540 IF TIME >=24*3600*divisor THEN zs$="000000":GOTO 310:REM >= 240000
550 :
560 gsec=FIX(TIME/divisor)'           Zeit in Sekunden
570 hrs=FIX(gsec/3600)
580 rsec=gsec-hrs*3600
590 minut=FIX(rsec/60)
600 sec=rsec-minut*60
610 '
620 altpl =loc:LOCATE 24,pi+4
630 '
640 IF alt=sec THEN 690
650 PRINT #0, USING "##: ";hrs;
660 PRINT #0, USING "##: ";minut;
670 PRINT #0, USING "##: ";sec:alt=sec
680 '
690 IF LOWER$(INKEY$) ="e" THEN LOCATE 1,23:CLEAR:END
700 IF x = 1 THEN ti$=STR$(TIME)
710 strpl=(0 ti$)
720 LOCATE 1,pi+13:PRINT"No":LOCATE 1,pi+13:PRINT"  "
730 loc=PEEK(strpl+2)*256+PEEK(strpl+1):LOCATE 24,pi+9:PRINT HEX$(loc,4)
740 IF altpl <loc THEN anzahl=anzahl +1:LOCATE 23,pi+7:PRINT anzahl
750 '
760 GOTO 540

```

In diesem Programm stecken aber noch weitere Informationen, die für Sie von Interesse sein können, deshalb erfolgt nun eine detaillierte Programmbeschreibung.

In Zeile 110 wird, um die Garbage-Collect-Routine möglichst oft aktiv werden zu lassen, ein großes Array dimensioniert. Die Anzahl der durchgeführten GC-Aufrufe wird auf -1 gesetzt, da das Programm die Anzahl in Zeile 740 hochzählt.



Um mittels des Variablenpointers zugreifen zu können, muß die Variable `ti$` einen Inhalt erhalten.

In Zeile 120 wird ein Positionswert für die späteren LOCATE-Befehle festgelegt.

In Zeile 130 werden die Arrays "gefüllt".

Die Zeilen 140 bis 190 sind für die Auswahl, ob das Programm so arbeiten soll, daß die GC aufgerufen wird, oder auch nicht, zuständig.

Der Divisor in Zeile 210 ist nichts anderes als der Korrekturfaktor für die Zeit. (Grober Wert 300 ms). Durch Anpassung kann hier die Genauigkeit der Uhr beeinflußt werden.

Durch Zeile 230 wird die Eingabe der Uhrzeit gefordert.

Die Zeilen 250 bis 290 weisen den Timer-Speicherstellen je nach CPC-Typ die richtigen Adressen zu.

Die Zeilen 310 bis 330 isolieren aus dem eingegebenen "Zeitstring" die Stunden, Minuten und Sekunden.

In Zeile 370 wird der Wert berechnet, der dann in den entsprechenden Speicherstelle stehen würde, wenn der CPC um 00.00.00 Uhr eingeschaltet worden wäre. Es ist die abgelaufene Zeit in Sekunden \* Divisor.

Aus diesem Wert werden in den Zeilen 380 bis 400 die durch die Zeile 430 zu pokenden Werte berechnet. Damit ist die Uhr auf die eingegebene Uhrzeit gestellt.

Die Zeilen 470 bis 510 geben verschiedene Texte aus.

In 570 wird überprüft, ob die Uhrzeit gleich oder größer 24.00 Uhr ist, wenn ja, dann wird die Uhr auf 00.00.00 gestellt.

Durch Zeile 560 wird die Gesamtsekundenzahl, die seit 00.00.00 Uhr vergangen ist, abgefragt und berechnet.

Durch die Zeilen 570 bis 600 werden die Stunden (hrs), die verbleibenden Restsekunden (wenn die Stunden abgezogen sind), daraus dann die Minuten und dann die Sekunden berechnet.

In 620 wird der Variablen `altpi` (alter Platz) die Adresse der Platzierung (`loc = location`) des Strings im Speicher zugewiesen.

In 640 wird abgefragt, ob die Uhr noch auf dem gleichen Sekundenwert steht, denn wenn diese noch nicht weitergelaufen ist, braucht auch keine neue Anzeige zu erfolgen, deswegen erfolgt dann ein Sprung zu Zeile 690.

Die Zeilen 650 bis 670 bewerkstelligen die Zeitausgabe auf den Bildschirm und der neue Sekundenwert wird in "alt" abgelegt.

Die Zeile 690 prüft ob das Programm abgebrochen werden soll und löscht alle Variablen, denn sonst könnten Sie sich beispielsweise nicht einmal mehr ein Disketteninhaltsverzeichnis ausgeben lassen, da der Speicher "zu voll" ist.

Falls Sie das Arbeiten der GC sehen wollen, haben Sie dies ja in Zeile 160 ausgewählt. Der Variablen x wurde daraufhin der Wert 0 oder 1 zugewiesen. Nun wird in Zeile 700, dann wenn Sie die Aktivitäten der GC sehen wollen, ein String erzeugt. Da wir ja bereits schon geschrieben haben, daß Strings nicht an einem Ort überschrieben werden, sondern nach unten laufen, können Sie es wegen dieser Stringzuweisung sehen.

In Zeile 710 wird der Variablenpointer (strpl), der Stringvariablen ti\$ geholt. Hinter dieser Speicherstelle läuft der wirkliche Zeiger auf die Speicherstelle mit, an welcher der letzte Stringeintrag für ti\$ zu finden ist. Dieser Wert wird dann in 730 ausgelesen, berechnet und als Hexzahl ausgegeben.

In Zeile 720 wird das Wort "No" auf dem Schirm ausgegeben und gleich wieder gelöscht. Dies bedeutet beim Programmlauf, daß dem in Zeile 500 ausgegebenen Text "Garbage collect" das Wort "No" vorangestellt wird. Dies wirkt sich beim Programmlauf so aus, daß das Wort No "blinkt", solange die GC nicht arbeitet.

In 740 wird dann festgestellt, ob die GC aktiv war. Bei der GC wird der Speicher ja aufgeräumt, das heißt, daß Plätze an denen sich Müllstrings befinden, für weitere Strings freigegeben werden. Ist der Wert in loc plötzlich größer als der von altp, dann wurde ti\$ oberhalb seiner bisherigen Adresse plaziert und das bedeutet, die GC muß aktiv gewesen sein.

Daraufhin wird dann (ebenfalls in 740) ausgegeben, wie oft dies während des Programmlaufes schon geschah.

Zeile 760 sorgt für den weiteren Programmlauf.

Ist die GC aktiv, so zeigt sich dies dadurch, daß der Rythmus des "No"-Blinkens durch eine Pause unterbrochen wird. Auch die Anzeige der Adresse, an der sich der String befand, steht für kurze Zeit.

Wer den CPC 464 und einen der Nachfolger hat kann nun sehr schön feststellen, daß die GC-Routinen geändert wurden. Während der CPC 464 diese Routine weniger oft aufruft, aber dann dafür länger braucht, sind die Nachfolger schneller aber die GC findet öfter statt. Insgesamt ist der CPC 464 langsamer.

Damit ist auch geklärt, warum CPC 464-Besitzer beim Einlesen der sequentiellen Files für den Disassembler längere Wartezeiten haben.

Haben Sie sich die Garbage Collect angesehen? Ja, dann können wir wieder weitermachen und Ihnen zum Schluß dieses Kapitels noch etwas über die Parameter, die bei einem Call-Befehl mitgegeben werden können sagen.

#### 10.4 Der Call-Befehl

Wie Sie bei vielen unserer Programme sehen konnten, ist es sehr komfortabel, Parameter gleich bei einem Call mitzuliefern.

Trotz der doch sehr umfangreichen Handbücher für die CPC's bleiben - auch nach gründlichem Studium derselben - noch viele Fragen offen.

Dies trifft auch für den Befehl CALL zu. So ist beispielsweise im Handbuch des CPC 6128 unter dem Stichwort CALL nachzulesen:

"Springt in ein Unterprogramm außerhalb des Basicbereiches, das an der angegebenen Speicheradresse beginnt. In unserem Beispiel wird der Computer durch diesen Befehl ganz zurückgesetzt. Dieser Befehl sollte mit Vorsicht angewendet werden. Verwandte Befehle: UNT" (Ende des Handbuchauszuges).

Wohl dem, der mit diesen Angaben sofort etwas anfangen kann. Abgesehen davon, daß mit CALL auch in den Basicprogrammbereich gesprungen werden kann, ist nur schwer - falls überhaupt - zu verstehen, weshalb dieser Befehl mit UNT verwandt ist.

(Daß auch sinnvoll in ein Basic-Programm gesprungen werden kann, zeigen die sogenannten REM-Routinen, siehe 10.5!)

Ohne entsprechende Erfahrung oder auch dem Studium des Firmwarehandbuches kann mit diesen Angaben sehr wenig angefangen werden. Falls man aber schon soweit "klarkommt", daß bei einem Einprung in ein Maschinenspracheprogramm Parameter übergeben werden können, so weiß man doch noch nicht, wie man diese Werte auch wieder findet bzw. wie sie weiterverarbeitet werden können. Ganz zu schweigen davon, daß die maximal erlaubte Anzahl der Parameter ebenfalls noch nicht bekannt ist.

Dabei ist dies alles sehr einfach, wenn man es weiß und hätte im Handbuch mit ein paar Sätzen erklärt werden können. Deshalb gleich zu den korrekten Angaben. Nach einem Maschinenprogramm-Einsprung durch den Befehl

CALL Adresse, Liste der Parameter

enthält der Akkumulator die Anzahl der übergebenen Parameter und im Doppel-Register IX ist die Adresse des Speicherplatzes enthalten, an der die "mitgegebenen" Parameter zu finden sind. Maximal können 32 Werte übergeben werden.

Das Beispielprogramm soll nur demonstrieren, wie man die übergebenen Werte "retten" kann. Sinnvoll wird eine derartige Übernahme von Parametern aber erst durch ein Maschinenprogramm selbst.

Experimentieren Sie zum besseren Verständnis am besten dadurch, daß Sie die Anzahl der Werte und die Werte selbst in Zeile 210 des Basicteiles verändern, dann sehen Sie sehr schnell wie auch ein Maschinenprogramm mit diesen Werten arbeiten kann.

## 10.4.1 Assemblerlisting: CALLEDemo.ED1

```

;*****
;*** calldemo.ed1 ***
;*****

; Diese Routine zeigt, dass bis zu 32 Variable beim
; CALL-Aufruf durch Basic uebergeben werden koennen
; Das zugehoerige Basicprogramm zur Demonstration
; heisst->      calldemo.bas

anzahl equ &a100      ;Register A  enthaelt Parameteranzahl
quelle equ &a102      ;Indexregister X enthaelt Parameteradresse

org &a000
add a,a              ;Anzahl verdoppeln ( Variablen =LOW+HIGH-Bytes =2 Bytes)
ld (anzahl),a        ;Anzahl der Parameter merken
ld (quelle),ix       ;Parameteradresse merken

ld bc,(anzahl)       ;Zum Verschieben muss in BC die Anzahl der Bytes stehen
ld hl,(quelle)       ;In HL die Quelle
ld de,&a110          ;in DE das Ziel
ldir                ; verschieben
ret                 ; zurueck nach Basic

```

## 10.4.2 Listing: CALLEDemo.BAS

```

100 REM *****
110 REM ***** Demonstrationsprogramm *****
120 REM *****      calldemo.bas      *****
130 REM ***** mit Maschinenprogrammteil *****
140 REM *****      30.01.85      *****
150 REM *****
160 '
170 MODE 1:FOR i = 0 TO 20:READ d$:d$="&"&d$:POKE &A000+i,VAL(d$):NEXT i
180 '
190 merkad=&A100:wertad=&A110
200 '
210 CALL &A000,i,2,3,4,5,6,7,8,9,&400,&8000,&FFFF
220 '
230 PRINT"Anzahl: ";PEEK (merkad)/2;:anzahl =PEEK(merkad)
240 '
250 PRINT"      Indexregister : "HEX$(PEEK(merkad+2)+PEEK(merkad+3)*256)
260 PRINT
270 '
280 FOR i = 1 TO anzahl STEP 2
290 PRINT"Wert "(i+1)/2;" :";TAB(20);PEEK(wertad+i-1)+PEEK(wertad+i)*256
300 NEXT i
310 '
320 DATA 87,32,0,a1,dd,22,02,a1,ed,4b,00,a1,2a,02,a1,11,10,a1,ed,b0,c9

```

## 10.5 REM-Routinen

Nun ganz zum Abschluß dieses Kapitels wollen wir Ihnen noch zeigen, daß Maschinensprache-Routinen nicht nur oberhalb von HIMEM, im freien Bereich des Bildschirmspeichers oder unterhalb von Basic abgelegt werden können, sondern auch mitten in einem Basicprogramm stehen können.

Dies ist dann interessant, wenn man kleine "Untersuchungsroutinen" hat, die bei der Analyse von fremden Programmen keinen Speicherplatz außerhalb des Basic-Bereiches belegen dürfen, weil dort ggf. der "Prüfling" aktiv ist oder war.

Ohne viele Worte zeigen wir Ihnen, wie eine Maschinenroutine in einem Basicprogramm aussehen kann.

Ausgefeiltere Formen derartiger Routinen sind unter dem Namen REM-Routinen bekannt.

## 10.5.1 Assemblerlisting: BASMPG1

```

;*****
;* BASMPG1 *
;*****

print equ &bb5a      ;Printausgabe
org &182             ;Startadresse

ld hl, textad        ;Adresse holen

weiter
ld a, (hl)           ;Akku mit Zeichen laden
cp &ff               ;mit &FF vergleichen
ret z                ;ja, Textende >Basic

call print           ;Zeichen ausgeben
inc hl               ;Textadresse erhöhen
jp weiter            ;weitermachen

textad text""        ;ab hier steht der Text
;-----

```

Und nun ein Basic-Demo, damit Sie sich das "Vorher" und "Nachher" ansehen können. Starten Sie dieses Programm beim ersten Male aber nicht durch RUN sondern durch RUN 110!

## 10.5.2 Listing: BASMPGB

```

100 CALL &182:END
110 'Im Bereich dieser Basiczeile steht nach Run 130 das Maschinenprogramm
120 '
130 FOR i = 0 TO 15
140 READ a$
150 POKE &182+i, VAL("&" + a$)

```

```
160 NEXT i
170 '
180 READ w$:FOR i = 16 TO 16 + LEN(w$)-1:x$=MID$(w$,i-15,1)
190 POKE &182+i,ASC(x$):NEXT:POKE &182+i,&FF
200 '
210 DATA 21,90,01,7e,fe,ff,c8,cd,5a,bb,23,c3,85,01,0d,0a
220 DATA "Das Maschinenprogramm steckt direkt im Basicteil !"
```

## 11. DEC\$ auf dem CPC 464

Obwohl auf den Schneider CPC 464 allerseits Lobeshymnen bezüglich seines starken Basic-Befehlssatzes gesungen werden, darf nicht vergessen werden, daß doch kleine Mißklänge bei der "Melodie" mitschwingen können.

So gut die Programmierer der Firmware auch gearbeitet haben, kleine und durchaus vermeidbare Flüchtigkeitsfehler sind leider im Basic-Interpreter vorhanden.

Verschiedene, uns bereits bekannte Fehler wurden dann bei den Nachfolgemodellen wieder "ausgebügelt". Dies kann dann bereits zu Kompatibilitäts-Problemen führen.

Einige kleine Fehler sind nie so richtig bekannt geworden. Beispielsweise kann es bei verschachtelten Schleifen und der vorherigen Festlegung der Laufvariablen durch DEFINT vorkommen, daß das Schleifenziel überlaufen wird.

Das gleiche Programm "läuft" aber auf dem CPC 664 und CPC 6128 völlig fehlerfrei. Es ist also durchaus möglich, daß trotz vermeintlicher syntaktisch richtig eingesetzter Befehle das Programm auf einem CPC 464 nicht korrekt abläuft, auf den beiden "Brüdern" aber völlig problemlos "spielt" und zu richtigen Ergebnissen führt. Im obengenannten "Schleifenproblem" half bereits das Weglassen der DEFINT-Anweisung.

Wer auf derartige Probleme stößt, sollte um anderen CPC-Besitzern mühevoll Suchen zu ersparen, solche kleinen "bugs" und am besten natürlich auch die Abhilfe, der einen oder anderen Fachzeitschrift mitteilen, damit durch deren Veröffentlichung die Informationen einem möglichst großen Kreis von CPC-Besitzern bekannt werden.

Aber auch andere "Problemchen" sind noch nicht ausreichend bekannt, weshalb wir hier auf einen Punkt dieser Art eingehen.

Wenn man sich näher mit den CPC's beschäftigt, läßt man sich irgendwann auch einmal den gesamten Befehlssatz des CPC's ausgeben und stößt dann auch auf Befehle, die entweder im Handbuch überhaupt nicht aufgeführt sind, oder aber nur unzureichend erklärt werden.

DEC\$ ist einer dieser Befehle. Im CPC 464-Handbuch ist er zwar nicht aufgeführt, bei genauerer Betrachtung des Interpreterteiles aber ist er zu finden.

In der TOKEN-Tabelle steht er als Befehlswort ab &E5E6. Der entsprechende Maschinenprogrammteil dieser Funktion ist ab \$F8EA zu finden. Lange Zeit wurde auch herumgerätselt, was dieser Befehl soll und kann. Fast alle Versuche führten aber zur Meldung: SYNTAX ERROR.

Aber wie schon gesagt, der Befehl ist vorhanden, aber leider muß er syntaktisch falsch eingesetzt werden. Nun aber erst einmal zur Klärung, was der Befehl eigentlich bewirkt.

Er ermöglicht die formatierte Ausgabe von Dezimalzahlen, wodurch ohne großen Aufwand zum Beispiel bei Tabellen auch alle Zahlen stellenrichtig untereinander ausgegeben werden können. Dieser Befehl ist auf allen CPC'S vorhanden, muß aber beim CPC 464 mit einer öffnenden Klammer mehr versehen werden als eigentlich erforderlich. Wie dieser Befehl zum Einsatz kommen kann, zeigt das nachfolgende kleine Programmlisting anhand zweier Beispiele sehr deutlich.

Besitzer der CPC 664 und 6128 brauchen die erste "öffnende" Klammer nach dem Befehlswort DEC\$ in den Zeilen 250 und 300 nur wegzulassen, dann läuft das Demoprogramm auch auf diesen Rechnern.

Andererseits brauchen CPC 464-Besitzer bei Programmen für den CPC 664 oder 6128 nur eine zusätzliche Klammer einzufügen und haben dann ein derartiges Programm auch für ihren Computer angepasst.

#### 11.1 Listing: DEC\$

```

100 REM *****
110 REM * Der Befehl DEC$ auf dem CPC 464 *
120 REM *****
130 '
140 MODE 2:PRINT"Der Befehl DEC$":PRINT
150 PRINT"Der Befehl DEC$ ist in allen CPC-Computern vorhanden. Leider ist
160 PRINT"im Betriebssystem des CPC 464 ein Fehler, wodurch dieser Befehl
170 PRINT"bei diesem Computer syntaktisch falsch angewendet werden muss.
180 PRINT"Der CPC 464 verlangt öffnend eine Klammer mehr als schliessend!
190 '
200 PRINT"Das nachfolgende kleine Beispielprogramm zeigt, wie es klappt:
210 PRINT:PRINT"Beispiel 1:
220 PRINT" Die Zahl x:", "x$ durch DEC$ gewonnen":PRINT
230 '
240 FOR i = 1 TO 5
250 x=RND(1)*1000:PRINT x,, "x$=DEC$((x, "###.###)":PRINT x$
260 NEXT i
270 '
280 PRINT:PRINT"Beispiel 2:
290 PRINT:PRINT"Die Zahl 123.22";:
300 x$=DEC$((123.22, "###.###)":PRINT, x$
310 '
320 '
330 PRINT:PRINT"Der Befehl erlaubt also eine formatierte Ausgabe !
340 PRINT"Bitte auch nachsehen unter : PRINT USING

```



Damit sind wir nun am Schluß der Grundlagenartikel angelangt. Wir hoffen, daß Sie - bereits bis jetzt - vieles haben lernen können. Im nächsten Kapitel beginnen die Anwendungsprogramme in denen aber auch sehr detailliert auf die wichtigen Punkte eingegangen wird.

Vorher aber noch zwei kleine Programme, die Sie vielleicht gebrauchen können. Versuchen Sie aber selbst einmal herauszufinden, was diese Programme tun und vor allem, wie sie bedient werden müssen, denn beide sind ja sehr kurz.

### 11.2 Listing: BEFEHLE

```
100 :::
110 MODE 2:PRINT"Dieses Programm listet den Befehlssatz Ihres CPC
120 PRINT"Das Programm wird durch die Taste 0 des Ziffernblockes bedient
130 '
140 PRINT:PRINT"Um das Programm zu beenden muss (!!!!)
150 PRINT"die kleine ENTER-Taste betätigt werden!
160 PRINT:PRINT"Bitte druecken Sie nun die Taste '0' immer wieder,"
170 PRINT"bis rechts die Ziffer 255 erscheint, oder solange bis Sie
180 PRINT"aufhoeren wollen.
190 CALL &BB18
200 '
210 CLS:WINDOW #0,1,80,1,1:WINDOW #1,1,80,3,20
220 WINDOW #2,1,6,2,24:WINDOW #3,75,79,2,2
230 a$="poke 374,255:poke 375,i:? #3,i:!"
240 b$="i=i+1:cls #2:list 100,# 1"+CHR$(13)
250 KEY 0,a#+b$
260 '
270 c$="call &bb00:mode 2:"
280 d$="poke 374,1:poke 375,1"+CHR$(13)
290 KEY 139,c#+d$
```

### 11.3 Listing: \$-Finder

```
1 CLS:PRINT"$-Finder"
2 ad=&170:INPUT"Suchstring"i$:=LEN(s$)
3 v$="": z1=PEEK(ad):zn = PEEK(ad+2)+PEEK(ad+3)*256
4 IF zn=0 THEN PRINT"ende":END
5 FOR i = ad+4 TO ad+z1
6 v$=v$+CHR$(PEEK(i)):NEXT i
7 IF INSTR(1,v$,s$)THEN PRINT zn
8 ad =ad + z1:GOTO 3
9 END
```

## 12. Das Floppy-RAM läßt sich verschieben!

Da ist man nun stolzer Besitzer einer Diskettenstation für seinen CPC-464 und hat in nächtelanger Arbeit die gesamte Softwareammlung vom Cassettenrecorder auf die Disketten überspielt.

Man will sich dann von der Mühe bei einem Spiel entspannen und lädt sein Lieblingsprogramm, doch was ist das? Im günstigsten Fall erscheint eine Fehlermeldung "Memory Full", der Computer kann aber auch gleich aussteigen.

Wer nun an seinen Computer-Kenntnissen zweifelt, entdeckt im Benutzer-Handbuch, das bei der Diskettenstation DDI-1 mitgeliefert wird, den Hinweis, daß das Betriebssystem zur Verwaltung der Diskettenstation rund ein Kilobyte benötigt.

Irgendwoher muß es diesen RAM-Block ja nehmen und schneidet sich ganz einfach am oberen Ende des Speichers direkt unter HIMEM genau 1280 Bytes ab, die den Programmen dann fehlen.

Nicht verzweifeln - Abhilfe ist durchaus möglich. Das bedeutet nämlich nicht, daß Sie den Computer nun bei ausgeschalteter Diskettenstation betreiben und die Programme weiterhin von der Cassette laden müssen, wie es das Handbuch empfiehlt. Denn in den meisten Fällen sind im Speicher noch viele Kilobytes frei, nur die Maschinenprogramme wurden für denjenigen Bereich assembliert, den jetzt die Floppy für sich beansprucht. Da es aber äußerst schwer ist, Maschinenprogramme ohne Kenntnis des Quellcodes im Speicher zu verschieben, bietet es sich doch eigentlich an, den Speicherblock der Diskettenstation in einem unbenutzten Bereich anzusiedeln ...

Das funktioniert mit einigen Tricks sogar: Beim Einschalten des Computers initialisiert das Betriebssystem alle vorhandenen Erweiterungs-ROMs. Zu diesen gehört auch das Betriebssystem der Floppy.

Da nun keine Software-Firma genau vorhersehen könnte, welchen Speicherbereich ein ROM-Modul einer anderen Firma zu seiner eigenen Verwaltung verlangt, haben die Entwickler bei Amstrad vorgeschrieben, daß diese RAM-Bereiche nur dynamisch vergeben werden dürfen. "Dynamisch" heißt, daß die Erweiterungs-ROMs nicht bestimmen dürfen, welchen Bereich sie benutzen werden, sondern nur, wieviele Bytes sie brauchen. Dadurch erfahren sie erst, wenn sie aufgerufen werden, wo der Speicher liegt. Bis zum nächsten Reset ist er dann für jedes ROM genau festgelegt.

Bei der Initialisierung der Erweiterungs-ROMs springt der Computer immer den ersten Eintrag in dessen RSX-Tabelle an, zum Beispiel heißt der erste RSX-Befehl des Floppy-ROMs "CPM ROM".

Durch das Leerzeichen im Namen haben Sie übrigens keine Möglichkeit, dieses RSX-Kommando von Basic aus aufzurufen. Vor dem Anspringen dieser Start-RSXen lädt das Betriebssystem das DE-Register des Z80-Prozessors mit einem Zeiger auf das untere Ende des "Memory Pools" und das HL-Register mit einem Zeiger auf dessen Obergrenze.

Der "Memory Pool" ist der Hauptspeicherbereich des Computers und liegt normalerweise im Bereich zwischen &0040 und &AC00. Die Initialisierungs-Routine des Steckkarten-ROMs kann nun die Registerinhalte verändern und dann dem Betriebssystem zurückgeben. Setzt sie zum Beispiel das HL-Register auf den Wert &A900, ist der Bereich zwischen &A900 und &AC00 geschützt und steht dann dem Erweiterungs-ROM exklusiv zur Verfügung. Beim Aufruf eines RSX-Programms, das in einem Erweiterungs-ROM zu finden ist, lädt das Betriebssystem das Indexregister IX des Prozessors mit einem Zeiger auf den bereitgestellten RAM-Bereich. So kann das Programm im ROM indirekt über das IX-Register auf den Speicher zugreifen, zum Beispiel durch Befehle wie LD A,(IX+5), CP (IX+20) oder INC (IX+50). Die Benutzung eines Bereichs von &0040 aufwärts, also am unteren Ende des Memory Pools, wird zwar vom Betriebssystem ebenfalls unterstützt, führt aber zu Problemen und wird von Amstrad nicht empfohlen.

Hier ist das Maschinenprogramm, mit dem Sie das Floppy-RAM verschieben können:

#### 12.1 Assemblerlisting: FMOVE.ASM

```
; *****
; *
; *          FMOVE.ASM verschiebt den Floppy-RAM
; *
; *****

                ORG          25000

; Definitionen *****

PCHL            EQU          &001B      ; KL FAR PCHL
BKINIT          EQU          &BCCE      ; KL INIT BACK
FINDCM          EQU          &BCD4      ; KL FIND COMMAND

; Hauptprogramm *****

MOVE            LD           HL,TAPE     ; Zeiger auf "TAPE"
                CALL        FINDCM      ; KL FIND COMMAND sucht RSX-Adresse
                RET           NC         ; Nicht gefunden? Basic-Rücksprung
                SUB          A           ; Keine Parameter Übergeben!
                CALL        PCHL        ; RSX im Floppy-ROM aufrufen

DSINIT          LD           C,7         ; RDM-Auswahl-Adresse der Floppy
                LD           DE,&004     ; Low-Adresse des Memory Pool
                LD           HL,35000    ; Höchste erlaubte Disk-RAM-Adresse
                CALL        BKINIT      ; KL BACK INIT re-initialisiert Disk
                RET           ; Rücksprung nach Basic

TAPE            DEFM         "TAP"      ; RSX-Name "TAPE"
                DEFB         &C5        ; Im letzten Byte ist das Highbyte=1

                END                ;
*****
```

Wenn Sie sich das Quellcode-Listing des Maschinenprogramms ansehen, dürfte Ihnen klar werden, wie sich das Floppy-RAM verschieben läßt.

Das hervorragend ausgestattete Betriebssystem der Schneider-Computer besitzt zur Initialisierung einzelner Erweiterungs-ROMs die Routine KL INIT BACK, die über einen Vektor an der Adresse &BCCE aufgerufen werden kann.

Das Maschinenprogramm schaltet zuerst auf den Cassettenrecorder um, um sicherzustellen, daß die Diskettenstation nicht mehr in Betrieb ist, und um das einwandfreie Weiterarbeiten mit den RSX-Befehlen im AMSDOS-ROM zu gewährleisten. Der Programmteil DSINIT lädt das C-Register mit dem Wert 7. Diese Zahl ist die Nummer des AMSDOS-ROMs. Alle Erweiterungs-ROMs erhalten Nummern, anhand derer das Betriebssystem sie verwalten kann. In den Registern DE und HL werden die Unter- und Obergrenze des Memory Pools übergeben. Wenn Sie wie im Programm den Wert 35000 als Obergrenze einsetzen, bedeutet das, daß der Bereich zwischen 35000 und dem Ende des Speichers ohne Floppy-Station nicht mehr durch das Disketten-RAM blockiert ist und frei verwendet werden kann. CALL BKINIT ruft die Betriebssystem-Routine auf, die das ROM Nummer 7 erneut initialisiert, und RET springt nach Basic zurück.

Da das Hantieren mit dem Quellcode beim wirklichen Einsatz des Programms aber recht umständlich ist, finden Sie hier auch noch einen Basic-Lader, der das Maschinenprogramm in DATA-Zeilen enthält (Zeilen 140 und 150):

## 12.2 Listing: FLMOVE.BAS

```
100 ' PROGRAMM ZUM VERSCHIEBEN DES FLOPPY-RAMs *****
110 '
120 DEF FNmsb(a)=255 AND INT(a/256)
130 DEF FNlsb(a)=255 AND UNT(a)
140 DATA &21,&BF,&61,&CD,&D4,&BC,&D0,&97,&CD,&1B,&00,&0E,&07,&11,&40
150 DATA &00,&21,&B8,&88,&CD,&CE,&BC,&C9,&54,&41,&50,&C5
160 MEMORY 24999 ' Voruebergewendender HIMEM
170 FOR i=&61A8 TO &61C2:READ a:POKE i,a:NEXT i
180 MODE 1:BORDER 2:INK 0,1:INK 1,24:LOCATE 1,2:PRINT STRING$(40,210);
190 PRINT CHR$(24)+SPACE$(9)+"FLOPPY-RAM VERSCHIEBEN"+SPACE$(9)+CHR$(24)
200 PRINT:PRINT:INPUT " Neue Startadresse: ",start
210 start=start+1279 ' Floppy-RAM benoetigt 1279 Bytes
220 POKE 25017,FNlsb(start):POKE 25018,FNmsb(start)
230 CALL 25000 ' Floppy-RAM verschieben
240 fstart=PEEK(&BE7D)+PEEK(&BE7E)*256
250 MEMORY fstart-5
260 PRINT:PRINT " Der neue HIMEM liegt bei ";MID$(STR$(HIMEM),2);"."
270 PRINT:PRINT:END
```

Dieser Lader bietet Ihnen die Möglichkeit, das Floppy-RAM in jeden beliebigen RAM-Speicher zu verlegen. Er druckt außerdem den neuen HIMEM-Wert aus.

Falls Sie sich wundern, warum dieser genau fünf Bytes kleiner als der von Ihnen gewünschte ist:

Das Betriebssystem muß die Erweiterungs-ROMs verwalten und benötigt vier Bytes, um die ROMs wiederfinden zu können. Diese vier Bytes legt das System immer vor dem RAM-Block ab, den sich ein Modul reserviert hat.

Wenn Sie nach dem Programmlauf noch den SYMBOL-AFTER-Wert verändern wollen, um eigene Grafikzeichen zu definieren, ist es ratsam, vor dem Programmstart SYMBOL AFTER 256 einzugeben. Mit diesem Befehl sagen Sie dem Computer, daß er alle benutzerdefinierten Zeichen löschen und den gesamten Zeichensatz ins ROM verlegen soll. Sie können dann MEMORY eingeben und trotzdem mit SYMBOL AFTER wieder Speicherplatz für die Zeichen belegen, ohne daß der Computer Sie durch "Improper Argument" daran hindert.

Mit diesem kleinen Programm läßt sich der größte Teil der nicht mehr lauffähigen Maschinenroutinen wieder starten. Neben den erheblich kürzeren Ladezeiten von der Diskette im Gegensatz zur Cassette können Sie auch während des Programmablaufs auf die Diskette zugreifen. Manche "lahme" Adressenverwaltung erwacht da zu neuer Blüte ...

Verschiedene Programme sind aber immer noch nicht wieder zu gebrauchen, und das aus einem ganz einfachen Grund: Sie belegen soviel Speicher (zum Beispiel für die Variablen), daß die 1280 Bytes, die die Diskettenstation haben will, nirgendwo mehr aufzutreiben sind. Aber auch da ist Hilfe möglich: Sie laden das Programm von der Diskette und de-initialisieren dann das Laufwerk.

Mit einigen POKE-Befehlen ist der Speicher wieder frei, und das Programm kann gestartet werden:

:DISC	Schaltet auf Diskettenbetrieb um.
LOAD "PROG.BAS"	Lädt das abzuarbeitende Programm von der Diskette in den Speicher. Den entsprechenden Programmnamen müssen Sie hier eben einsetzen.
SYMBOL AFTER 256	Löscht alle benutzerdefinierten Zeichenmatrizen und läßt den Computer nur auf den ROM-Zeichensatz zugreifen.
:TAPE	Wählt den Cassettenbetrieb. Dies ist die letzte Chance, vor dem de-Initialisieren einen RSX-Befehl des AMSDOS-ROMs ausführen zu lassen.
CALL &BCC8	Ist die Betriebssystem-Routine KL CHOKO OFF, die den Computer weitmöglichst zurücksetzt und alle Zugriffsmöglichkeiten auf Erweiterungs-ROMs unterbindet. Der RAM-Bereich wird durch den Call-Aufruf aber nicht

	gelöscht: Die Programme im Speicher bleiben also erhalten!
POKE &AE7B,&FF	Setzt den einen der beiden internen Zeiger auf den HIMEM-Wert &ABFF. (Lowbyte) Wenn Sie einen CPC-664 oder CPC-6128 benutzen, verwenden Sie bitte &AE5E.
POKE &AE7C,&AB	Setzt den einen der beiden internen Zeiger auf den HIMEM-Wert &ABFF. (Highbyte) Für Besitzer eines CPC-664 oder CPC-6128: &AE5F.
POKE &AE7D,&FF	Setzt den zweiten der internen Zeiger auf den HIMEM-Wert &ABFF. (Lowbyte) CPC-664- und CPC-6128-Besitzer aufgepaßt! Sie verwenden hier &AE60.
POKE &AE7E,&AB	Setzt den zweiten der internen Zeiger auf den HIMEM-Wert &ABFF. (Highbyte) Für den CPC-664 und CPC-6128: &AE61
SYMBOL AFTER 240	Ist der Standardwert für den Zeichensatz nach dem Einschalten des Computers.
RUN	Startet das Programm endlich.

Programmen, die sich mit all diesen Tricks nicht starten lassen, ist nicht mehr zu helfen: Schalten Sie für diese Fälle die Floppy-Station aus und laden Sie das Programm von der Cassette!

Solche Programme sind aber sehr selten und werden glücklicherweise auch immer seltener.

### 13. DIRSYS listet auf Systemfiles

Wer sich schon etwas mit CP/M beschäftigt hat, kennt sicher das STAT-Programm. Mit diesem lassen sich - unter anderem - Dateien mit einem speziellen Schutz versehen:

Sie tauchen nicht mehr im Directory (Inhaltsverzeichnis) der Diskette auf.

Programme oder Dateien auf diese Art verschwinden zu lassen, ist ganz einfach:

Laden Sie mit dem RSX-Befehl !CPM das CP/M-Betriebssystem von einer Diskette, die STAT.COM enthält.

Wenn Sie einen CPC-6128 besitzen, verwenden Sie dazu bitte eine Kopie der Seite 4 der mitgelieferten Systemdisketten. Auf dieser Seite befindet sich nämlich das Betriebssystem CP/M 2.2. Bei der normalerweise auf dem CPC-6128 verwendeten Version 3.0 (CP/M Plus) wurde STAT.COM in die Programme SET.COM und SHOW.COM aufgesplittet.

Sobald der CP/M-Prompt "A>" erscheint, können Sie Dateien verschwinden lassen, wenn die Diskette nicht gerade mit einem Schreibschutz versehen ist, wie das bei den gelieferten Systemdisketten der Fall ist.

Haben Sie also eine Diskette ohne Schreibschutz im Laufwerk, können Sie zum Beispiel das STAT-Programm selbst verstecken:

```
A>STAT STAT.COM $SYS
```

Durch die Angabe "\$SYS" setzt das Betriebssystem das Dateiattribut "System-File". Ein nachfolgendes DIR unterschlägt dann den Verzeichniseintrag für das STAT.COM-Programm.

Nur STAT \*.\* listet auch STAT.COM wieder auf, nämlich in Klammern:

```
(STAT.COM)
```

Aufheben läßt sich der Schutz durch Angabe des Attributs "\$DIR" für "Directory-File":

```
A>STAT STAT.COM $DIR
```

Da aber die Arbeit in CP/M nicht jedermanns Sache ist und viele sich in Basic wohler fühlen, finden Sie hier ein Programm, das alle Dateien einer Diskette - also auch die Systemfiles - auflistet. Der Befehl wird als RSX-Erweiterung ins System integriert. Er soll genauso wie der !DIR-Befehl funktionieren und so auch die

Angabe eines Dateinamens als String gestatten:

```
!DIRSYS
A$="*.BAS":!DIRSYS,@A$
B$="*.COM":!DIRSYS,@B$
X$="A:WS??????.*":!DIRSYS,@X$
```

Es ist aber sehr schwierig, einen solchen Befehl ohne tiefere Systemkenntnisse zu entwickeln. Auch würde es nicht gerade zur Kürze des Programms beitragen, müßte man die Directory-Sektoren von der Diskette auslesen und in ein brauchbares Bildschirm-Format bringen. In einem solchen Fall bietet sich eine riesige Programmbibliothek an: das Floppy-ROM! Irgendwo muß in diesem 16K-Bereich ja der !DIR-Befehl programmiert sein. Mit der Hilfe eines Disassemblers oder eines ROM-Listings kommt man der RSX "!DIR" auf die Schliche: An der Adresse &C0AD im AMSDOS-ROM ist der RSX-Name abgespeichert, der zugehörige Sprungbefehl steht in &C02A. Wird diese Adresse angesprungen, reicht sie den Sprung nach &D42E weiter.

Und tatsächlich - dort kann man den kompletten !DIR-Befehl entdecken. Vollzieht man die Logik der Programmierer nach, findet man auch recht schnell die Stelle, an der Dateien mit SYS-Attribut unterdrückt werden:

```
&D454 - CALL &D9DF
&D457 - JR C,&D44F
&D459 - EX (SP),HL
```

Die ROM-Routine ab &D9DF stellt fest, ob eine Datei mit \$SYS geschützt ist. Dies ist im Directory vermerkt: Sofern das höchste Bit des ersten Zeichens der Extension gesetzt ist, ist \$SYS wirksam, zum Beispiel im Programmnamen:

#### PROGRAMM.BAS

Dort setzt STAT.COM das siebte Bit im "B" von ".BAS" auf Eins. Die ROM-Routine wertet den Dateinamen dementsprechend aus und setzt das Carry-Flag, wenn das Bit gesetzt ist. Andernfalls löscht das Unterprogramm das Carry-Flag des Z80-Prozessors.

Doch allzu viel nützt diese Kenntnis des internen Aufbaus von !DIR nichts: Ohne EPROM-Brenner kann man nun einmal nichts an Festwertspeichern wie ROMs, EPROMs oder EEPROMs ändern. Und das wäre doch wohl ein zu großer Aufwand.

Es geht viel einfacher: Man muß sich nur ein Programm schreiben, das an den ROM-Inhalt angelehnt und in den entsprechenden Teilen modifiziert ist.



Der Quellcode von DIRSYS zeigt, wie man's macht:

### 13.1 Assemblerlisting: DIRSYS

```

; *****
; *
; * DIRSYS - Zeigt im Directory auch SYS-Dateien an *
; *
; *****
;

A000 (A000)          ORG  &A000      ; Loader relocatibel!

A000 (B90F)          ROMSEL EQU  &B90F      ; KL ROM SELECT
A000 (B918)          DESELEC EQU  &B918      ; KL ROM DESELECT
A000 (BB5A)          TXTOUT EQU  &BB5A      ; TXT OUTPUT
A000 (BCD1)          LOGEXT EQU  &BCD1      ; KL LOG EXTERNAL

;
; Routinen im Floppy-ROM
;

A000 (CDC2)          CHKPAR EQU  &CDC2      ; Parameterzahl prüfen
A000 (CDC7)          GETVAR EQU  &CDC7      ; Varpointer holen
A000 (DAA6)          CORREC EQU  &DAA6      ; Filenamen korrigieren
A000 (CE14)          GETDPH EQU  &CE14      ; DPH nach HL holen
A000 (DBD0)          DRVUSR EQU  &DBD0      ; Drive & User ausgeben
A000 (D472)          SCRFOR EQU  &D472      ; Screenformat bestimmen
A000 (D683)          BLKMAP EQU  &D683      ; Blockmap bestimmen
A000 (D698)          SEARCH EQU  &D698      ; Namen & Daten suchen
A000 (D9DF)          IFSYS EQU  &D9DF      ; Prüfen, ob SYS-File
A000 (DBC4)          SPACE3 EQU  &DBC4      ; Drei Spaces ausgeben
A000 (DBC8)          FILNAM EQU  &DBC8      ; Filenamen ausgeben
A000 (CAEB)          MSG EQU  &CAEB        ; Systemmeldungen ausgeben
A000 (D8C2)          USED EQU  &D8C2        ; Belegte Blocks berechnen

A000 (0002)          ROMSTA DEFS 2          ; ROM-Status bei DIRSYS
A002 (0004)          SPACE DEFS 4          ; Speicher fuer RSXen

A006 0B A0          RSXTAB DEFW RSXNAM      ; Zeiger auf RSX-Namen
A008 C3 2B A0          JP DIRSYS          ; Sprung zur Routine

A00B 44 49 52 53    RSXNAM DEFM "DIRSY"    ; RSX-Name "DIRSYS"
A00F 59
A010 D3              DEFB "S"+&80          ; 7.Bit = Ende des Namens
A011 00              DEFB &00             ; 0 = Ende der Tabelle

A012 07 42 61 64    ERRMSG DEFB 7,"Bad Command"
A016 20 43 6F 6D
A01A 6D 61 6E 64
A01E 0D 0A 0A          DEFB 13,10,10

A021 01 06 A0        INIT LD BC,RSXTAB    ; Zeiger auf Sprungtabelle
A024 21 02 A0        LD HL,SPACE          ; Zeiger auf Hilfsspeicher
A027 CD D1 BC        CALL LOGEXT          ; RSX einbinden

```

```

A02A C9                RET                ; Rücksprung nach Basic

; ***** Hauptprogramm DIRSYS *****

A02B 0E 07            DIRSYS LD C,7        ; Nummer des Floppy-ROMs
A02D F5              PUSH AF            ; Akkuinhalt retten
A02E CD 0F B9        CALL ROMSEL        ; Floppy-ROM einblenden
A031 ED 43 00 A0     LD (ROMSTA),BC    ; ROM-Status sichern
A035 F1              POP AF            ; Akku wiederherstellen
A036 FD 2A 7D BE     LD IY,(&BE7D)    ; IY = Floppy-Memory Pool

A03A 06 00          LD B,0            ; Stringlänge = 0
A03C B7            OR A              ; Parameter aus Basic?
A03D 28 18        JR Z,CONTIN        ; Nein - Überspringen
A03F FE 01        CP 1              ; Ein Parameter?
A041 28 0E        JR Z,PARAMS        ; Ja - Überspringen

A043 06 0F        ERROR LD B,15       ; Fehler - 15 Buchstaben
A045 21 12 A0     LD HL,ERRMSG        ; ausgeben "Bad
A048 7E          ERROUTP LD A,(HL)    ; "Command", CR, LF
A049 CD 5A BB     CALL TXTOUT         ; TXT OUTPUT &B5A
A04C 23          INC HL              ;
A04D 10 F9       DJNZ ERROUTP        ;
A04F 18 3F       JR OLDROM           ; Rücksprung nach Basic

A051 CD C2 CD     PARAMS CALL CHKPAR   ; Max.1 Parameter zulassen
A054 CD C7 CD     CALL GETVAR         ; Stringpointer holen

A057 CD A8 DA     CONTIN CALL CORREC  ; Korrekten Namen bilden
A05A CD 14 CE     CALL GETDPH        ; DPH-Zeiger nach HL
A05D CD D0 DB     CALL DRVUSR        ; Drive & User ausgeben
A060 3E 0C       LD A,&0C            ; Maximale Namenslänge
A062 CD 72 D4     CALL SCRFOR        ; Screenformat berechnen
A065 65          LD H,L              ; L-Inhalt ins H-Register
A066 E5          PUSH HL             ; HL-Register sichern
A067 CD 83 D8     CALL BLKMAP        ; Blockmap holen

A06A CD 98 D6     NXTENT CALL SEARCH  ; Filenamen suchen
A06D 30 18       JR NC,EXIT          ; Wenn fertig ...
A06F CD DF D9     CALL IFSYS         ; Ein SYS-File?

; Normalerweise wird hier ein SYS-Eintrag unter-
; drückt. DIRSYS drückt alles aus ...

A072 E3          EX (SP),HL          ; Screenformat vom Stack
A073 C5          PUSH BC             ; BC-Register sichern
A074 7C          LD A,H              ; CP-Befehl vorbereiten
A075 BD          CP L                ; Zeile voll?
A076 C4 C4 DB     CALL NZ,SPACE3     ; Nein, drei Spaces zeigen
A079 CC 98 A0     CALL Z,CRLF        ; Ja, CR & LF ausgeben
A07C CD C8 DB     CALL FILNAM        ; Filenamen ausgeben
A07F 2D          DEC L               ; Zahl der Einträge -1
A080 20 01       JR NZ,CORRBC        ; Weiter, wenn <>0
A082 6C          LD L,H              ; Sonst Wert übernehmen

A083 C1          CORRBC POP BC        ; BC wiederherstellen
A084 E3          EX (SP),HL          ; Stack korrigieren

```

```

A085 10 E3          JR    NXTENT      ; Zum nächsten Eintrag

A087 E1            EXIT    POP    HL    ; HL wiederherstellen
A088 CD C2 D8      CALL    USED      ; Zahl der belegten Blocks
A08B 3E 03         LD      A,3        ; Message 3 "... K free"
A08D CD EB CA      CALL    MSG        ; Meldung ausgeben

A090 ED 4B 00 A0   OLDROM LD      BC,(ROMSTA) ; ROM-Status holen
A094 CD 18 B9      CALL    DESELEC    ; ROM deselektieren
A097 C9            RET                ; Rücksprung nach Basic

A098 3E 0A         CRLF   LD      A,10    ; Code für LF
A09A CD 5A BB      CALL    TXTOUT      ; Ausgeben
A09D 3E 0D         LD      A,13        ; Code für CR
A09F CD 5A BB      CALL    TXTOUT      ; Ausgeben
A0A2 C9            RET                ; Rücksprung
A0A3 (A0A3)       END                ; -----

```

## Symboltabelle:

D683 BLKMAP	CDC2 CHKPAR	DAA6 CORREC	A057 CONTIN
A083 CORRBC	A098 CRLF	B918 DESELEC	BD00 DRVUSR
A02B DIRSYS	A012 ERRMSG	A043 ERROR	A048 ERRUTP
A087 EXIT	DBC8 FILNAM	CDC7 GETVAR	CE14 GETDPH
D9DF IFSYS	A021 INIT	BCD1 LOGEXT	CAEB MSG
A06A NXTENT	A090 OLDROM	A051 PARAMS	B90F ROMSEL
A000 ROMSTA	A006 RSXTAB	A00B RSXNAM	D472 SCRFOR
D698 SEARCH	DBC4 SPACE3	A002 SPACE	BB5A TXTOUT
D8C2 USED			

Ab dem Label DIRSYS beginnt das eigentliche Programm. Davor befinden sich Deklarationen und Definitionen sowie der Programmblock INIT, der die RSX-Erweiterung installiert und im Betriebssystem verankert.

DIRSYS selektiert zuerst das Floppy-AMSDOS-ROM und sichert den alten RAM/ROM-Status in der Speicherstelle ROMSTA. Das IY-Register wird mit einem Zeiger auf den RAM-Speicher des Disketten-Betriebssystems geladen. Außerdem wird geprüft, ob ein Parameter aus Basic übergeben wird.

In diesem Fall lädt nämlich das Betriebssystem den Akkumulator des Z80 mit der Zahl der übergebenen Parameter.

Wurde kein Stringparameter angegeben, springt das Programm zur Marke CONTIN, andernfalls prüft es, ob wirklich genau ein Parameter gefunden wurde. Sind es mehrere, gibt die RSX-Erweiterung die Fehlermeldung "Bad Command" aus und kehrt unverrichteter Dinge nach Basic zurück. Andernfalls holt sich DIRSYS mit Hilfe zweier ROM-Routinen (CHKPAR ab &CDC2 und GETVAR, Adresse &CDC7) den String.

An der Marke CONTIN weist das Programm den Computer an, aus dem String einen korrekten Datei-Namen zu bilden (ROM-Routine CORREC ab &DAA6). CORREC wird auch aufgerufen, wenn dem Programm gar

kein String als Datei-Name übergeben wurde. In diesem Fall ist die Stringlänge im B-Register Null. Die ROM-Routine legt einen Wildcard-String "\*. \*" beziehungsweise "????????.???" an.

Die folgenden Programmzeilen sind im wesentlichen eine Übernahme aus dem AMSDOS-ROM:

Sie holen einen Zeiger auf den Disk-Parameter-Header (DPH), geben "Drive: x user y" aus, bestimmen ein für den jeweiligen MODE geeignetes Ausgabeformat, suchen die Dateinamen im Directory und prüfen mittels der Routine IFSYS (Adresse &D9DF), ob der auszugebende Dateiname als SYS-Systemfile gekennzeichnet ist.

Im Original-Code steht hier ein bedingter Sprungbefehl, der ausgeführt wird, wenn der Computer die Datei als SYS-File erkennt. Das Programm DIRSYS verzichtet auf den Sprung und druckt damit alle Dateinamen aus.

Das Programm befindet sich ständig in einer Schleife zwischen NXTENT ("Next Entry") und CORRBC ("Correct BC-Register"). Diese wird verlassen, wenn SEARCH (Adresse &D698) das Carry-Flag löscht. In diesem Fall liegen keine weiteren Einträge mehr vor, und der Computer springt zum Label EXIT. Dort gibt DIRSYS noch die Abschlußmeldung "x K free" aus. Dazu verwendet es die Systemmeldung 3, die im ROM an der Adresse &CB90 bis &CB98 zu finden ist. Anhand der Nummer 3 erkennt MSG ("Message Output", Adresse &CAEB) die richtige Meldung und gibt sie aus.

OLDROM stellt den alten ROM-Status wieder her und verwendet dazu die Betriebssystem-Routine KL ROM DESELECT an der Adresse &B90F. Sie verlangt im BC-Register den alten RAM/ROM-Zustand, der in ROMSTA am Programmstart gesichert wurde. RET sorgt für den Rücksprung nach Basic.

Bei der Programmeingabe können Sie - wie bei Maschinenprogrammen üblich - wählen, ob Sie lieber den Quellcode oder den Basic-Loader abtippen. Der Basic-Loader erzeugt einen verschiebbaren Programmcode und legt DIRSYS immer unmittelbar unter HIMEM ab, so daß Sie nur 166 Bytes des freien RAM-Speichers hergeben müssen.

Das Programm initialisiert auch gleich die RSX-Erweiterung und gibt ein Anwendungsbeispiel für DIRSYS. Hier ist das zugehörige Listing:

### 13.2 Listing: DIRSYS.BAS

```

100 ' *****
110 ' *
120 ' * DIRSYS - Systemdateien listen *
130 ' *
140 ' *****
150 '
160 DEF FNmsb(a)=255 AND INT(a/256)
170 DEF FNlsb(a)=255 AND UNT(a)
180 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
190 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20

```

```

200 LOCATE 1,1:PRINT STRING$(40,210);
210 LOCATE 1,2
220 PRINT CHR$(24)+SPACE$(17)+"DIRSYS"+SPACE$(17)+CHR$(24)
230 LOCATE 10,5:PRINT CHR$(164);" Martin Kotulla 1986"
240 PRINT:PRINT STRING$(40,210):PRINT
250 PEN 1:PRINT " DIRSYS ist eine RSX-Erweiterung, mit"
260 PRINT:PRINT " der analog zum DIR-Befehl auch die"
270 PRINT:PRINT " Systemfiles einer Diskette angezeigt"
280 PRINT:PRINT " werden."
290 PEN 2:PRINT:PRINT TAB(17);"!DIRSYS"
300 FOR i=40960 TO 41123:READ a:sum=sum+a OR (UNT(i) AND 255):NEXT i
310 IF sum=25855 THEN 340
320 PRINT:PRINT CHR$(7);"* Fehler in den DATA-Zeilen!":PRINT
330 LIST 460:END
340 RESTORE:MEMORY HIMEM-166:start=HIMEM+1
350 FOR i=start TO start+163:READ a:POKE i,a:NEXT i
360 FOR i=1 TO 8:READ a:a=a-40960+start
370 value=PEEK(a)+PEEK(a+1)*256-40960+start
380 POKE a,FNlsb(value):POKE a+1,FNmsb(value)
390 NEXT i
400 CALL start+21 ' RSX initialisieren
410 MODE 2:PEN 1:WINDOW 2,79,2,24
420 PRINT " DIRSYS":PRINT " ";STRING$(6,"-");
430 !DIRSYS ' So sieht der neue Befehl aus
440 PRINT " DIR":PRINT " ";STRING$(3,"-");:!DIR
450 ' Maschinencode-Daten *****
460 DATA &00,&00,&00,&00,&00,&00,&0B,&A0,&C3,&2B,&A0,&44,&49,&52,&53,&59
470 DATA &D3,&00,&07,&42,&61,&64,&20,&43,&6F,&6D,&61,&6E,&64,&0D,&0A
480 DATA &0A,&01,&06,&A0,&21,&02,&A0,&CD,&D1,&BC,&C9,&0E,&07,&F5,&CD,&0F
490 DATA &B9,&ED,&43,&00,&A0,&F1,&FD,&2A,&7D,&BE,&06,&00,&B7,&28,&18,&FE
500 DATA &01,&28,&0E,&06,&0F,&21,&12,&A0,&7E,&CD,&5A,&BB,&23,&10,&F9,&18
510 DATA &3F,&CD,&C2,&CD,&CD,&C7,&CD,&CD,&A6,&DA,&CD,&14,&CE,&CD,&D0,&DB
520 DATA &3E,&0C,&CD,&72,&D4,&65,&E5,&CD,&83,&D6,&CD,&98,&D6,&30,&18,&CD
530 DATA &DF,&D9,&E3,&C5,&7C,&BD,&C4,&C4,&DB,&CC,&98,&A0,&CD,&C8,&DB,&2D
540 DATA &20,&01,&6C,&C1,&E3,&18,&E3,&E1,&CD,&C2,&D8,&3E,&03,&CD,&EB,&CA
550 DATA &ED,&4B,&00,&A0,&CD,&18,&B9,&C9,&3E,&0A,&CD,&5A,&BB,&3E,&0D,&CD
560 DATA &5A,&BB,&C9,&00
570 ' Daten zum Verschieben des Maschinencodes *****
580 DATA &A006,&A009,&A022,&A025,&A033,&A046,&A07A,&A092
590 END ' -----

```

## 14. Echte Unterlängen für den Drucker GP-500

Neben dem von Schneider selbst angebotenen Drucker NLQ401 - Ubri-  
gens ein OEM-Fabrikat von Centronics und Brother - hat sich vor  
allem wegen des günstigen Preises der Drucker GP500-CPC von Sei-  
kosha auf dem Markt etabliert.

Er hat nur einen großen Nachteil: das äußerst mäßige Schriftbild. Am störendsten sind wohl die fehlenden Unterlängen bei den g's, j's, p's, q's und beim Ypsilon. Aber auch die anderen Buchstaben schauen nicht gerade perfekt aus. Das folgende Programm schafft für beide Fälle Abhilfe.

Es benutzt die hochauflösende Grafik des Druckers, um schönere Symbole darzustellen. Da das Programm natürlich die Ausgabe- geschwindigkeit herabsetzt und auch eine ganze Menge Speicherplatz im Computer kostet, ist es nicht sehr sinnvoll, es in Maschinen- sprache zu schreiben und so ins Betriebssystem zu integrieren, daß alle Druckerausgaben durch die hochauflösenden Zeichen er- setzt werden. Es ist viel einfacher, die Zeichenausgabe in Basic zu schreiben und nur auf das Ausdrucken einzelner Disketten- oder Cassetten-Dateien zu beschränken.

Daß auch das "Patching" des Vektors für die Druckerausgabe möglich ist, zeigt das ebenfalls in diesem Buch abgedruckte Programm, das dem Seikosha-Drucker zusätzlich zu einem amerikanischen Zeichensatz verhilft.

Doch hier ist erst einmal das Listing des "Unterlängen-Programms":

#### 14.1 Listing: SEIKO.BAS

```

100 ' *****
105 ' * *
110 ' * SEIKOSHA & UNTERLAENGEN *
115 ' * *
120 ' *****
125 '
130 DIM char$(126):SYMBOL AFTER 250
135 file$=SPACE$(255) ' String vorbereiten, da Fehler im Interpreter!
140 SYMBOL 254,&66,&0,&66,&66,&66,&3E,&0
145 SYMBOL 255,&66,&0,&78,&C,&7C,&CC,&76,&0
150 WIDTH 255 ' CR/LF muessen unterdrueckt werden!
155 ' Titelbild und Benutzereingaben *****
160 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
165 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
170 LOCATE 1,1:PRINT STRING$(40,210);
175 LOCATE 1,2
180 PRINT CHR$(24)+SPACE$(8)+"SEIKOSHA & UNTERLAENGEN"+SPACE$(9)+CHR$(24)
185 LOCATE 10,5:PRINT CHR$(164);" Martin Kotulla 1985"
190 PRINT:PRINT STRING$(40,210):PRINT
195 PEN 1:PRINT " Dieses Programm verleiht dem Drucker"
200 PRINT:PRINT " Seikosha GP-500CPC die F"+CHR$(255)+"higkeit,"
205 PRINT:PRINT " Texte aus Dateien mit echten Unter-"
210 PRINT:PRINT " !"+CHR$(255)+"ngen auszudrucken. Zudem werden"

```

```

215 PRINT:PRINT " auch die anderen Zeichen besser lesbar."
220 PEN 3:PRINT STRING$(40,210)
225 WHILE INKEY$<>"":WEND ' Tastaturpuffer loeschen
230 PEN 2:PRINT:PRINT " Bitte dr"+CHR$(254)+"cken Sie eine Taste!"
235 WHILE INKEY$="":WEND ' Auf Tastendruck warten
240 MODE 1:PRINT:PRINT
245 PEN 2:PRINT " Zeilen!"+CHR$(255)+"nge? ";
250 PEN 1:LINE INPUT linelen$:linelen=VAL(linelen$)
255 PEN 2:PRINT:PRINT " Dateiname?";SPACE$(3);
260 PEN 1:LINE INPUT file$:INK 2,0
265 IF LEFT$(file$,1)<>"!" THEN file$="!" + file$
270 PEN 2:PRINT:PRINT " Seiten!"+CHR$(255)+"nge? ";
275 PEN 1:LINE INPUT allowedlines$:allowedlines=VAL(allowedlines$)
280 WINDOW 1,40,10,24
285 ' Zeichendefinitionen einlesen *****
290 FOR i=32 TO 126
295 READ dummy$
300 FOR j=1 TO 6
305 READ b$
310 char$(i)=char$(i)+CHR$(VAL(b$))
315 NEXT j
320 READ dummy$
325 NEXT i
330 ' Datei ausdrucken *****
335 garbage=FRE(""):OPENIN file$
340 WHILE NOT EOF
345 LINE INPUT #9,b$:lencont=1
350 a$=MID$(b$,lencont,linelen):lencont=lencont+linelen
355 IF LEN(a$)=0 THEN 500
360 ul$=""
365 FOR i=1 TO LEN(a$)
370 s=ASC(MID$(a$,i))
375 IF s<32 THEN PRINT #8,CHR$(s);:GOTO 400
380 PRINT #8,CHR$(27);CHR$(75);CHR$(0);CHR$(6);char$(s);
385 ptpos=ptpos+1
390 s$=CHR$(s)
395 IF INSTR("gjpqy",s$)<>0 THEN ul$=ul$+s$ ELSE ul$=ul$+SPACE$(1)
400 NEXT i
405 PRINT #8
410 FOR i=1 TO LEN(ul$)
415 s=ASC(MID$(ul$,i)):s$=CHR$(s)
420 PRINT #8,CHR$(27);CHR$(75);CHR$(0);CHR$(6);
425 IF s$="g" THEN PRINT #8,STRING$(2,0)+STRING$(3,2)+CHR$(1);
430 IF s$="j" THEN PRINT #8,STRING$(2,0)+STRING$(2,1)+STRING$(2,0);
435 IF s$="p" THEN PRINT #8,CHR$(0)+CHR$(3)+STRING$(4,0);
440 IF s$="q" THEN PRINT #8,STRING$(5,0)+CHR$(3);
445 IF s$="y" THEN PRINT #8,CHR$(2)+CHR$(1)+STRING$(4,0);
450 IF s$=" " THEN PRINT #8,STRING$(6,0);
455 NEXT i
460 PRINT #8
465 lpage=lpage+1:IF lpage<allowedlines THEN 490
470 WHILE INKEY$<>"":WEND ' Tastaturpuffer loeschen
475 PRINT " Bitte neue Seite einlegen!"
480 WHILE INKEY$="":WEND ' Auf Tastendruck warten
485 lpage=0
490 ptpos=0
495 IF lencont<255 THEN 350

```

```

500 WEND
505 CLOSEIN:MODE 1:PEN 1:END
510 ' Zeichendefinitionen; *?: Zeichen nicht darstellbar *****
515 DATA *?*,&00,&00,&00,&00,&00,&00,&00
520 DATA !!!,&00,&00,&00,&5F,&00,&00,&00
525 DATA *?*,&00,&00,&07,&00,&07,&00,&00
530 DATA ###,&00,&14,&7F,&14,&7F,&14,&00
535 DATA ***,&00,&26,&49,&7F,&49,&32,&00
540 DATA %%%,&00,&23,&13,&08,&64,&62,&00
545 DATA &&&,&00,&36,&49,&56,&28,&50,&00
550 DATA ''',&00,&00,&00,&04,&03,&00,&00
555 DATA (((,&00,&00,&1C,&22,&41,&00,&00
560 DATA ))),&00,&00,&41,&22,&1C,&00,&00
565 DATA ***,&00,&08,&2A,&1C,&2A,&08,&00
570 DATA +++,&00,&08,&08,&3E,&08,&08,&00
575 DATA *?*,&00,&00,&50,&30,&00,&00,&00
580 DATA ---,&00,&08,&08,&08,&08,&08,&00
585 DATA ...,&00,&00,&60,&60,&00,&00,&00
590 DATA ///,&00,&20,&10,&08,&04,&02,&00
595 DATA 000,&00,&3E,&41,&41,&41,&3E,&00
600 DATA 111,&00,&00,&42,&7F,&40,&00,&00
605 DATA 222,&00,&42,&61,&51,&49,&46,&00
610 DATA 333,&00,&22,&41,&49,&49,&36,&00
615 DATA 444,&00,&18,&14,&12,&7F,&10,&00
620 DATA 555,&00,&27,&45,&45,&45,&39,&00
625 DATA 666,&00,&3C,&4A,&49,&49,&30,&00
630 DATA 777,&00,&01,&71,&09,&05,&03,&00
635 DATA 888,&00,&36,&49,&49,&49,&36,&00
640 DATA 999,&00,&06,&49,&49,&29,&1E,&00
645 DATA *?*,&00,&00,&36,&36,&00,&00,&00
650 DATA ;;;,&00,&00,&56,&36,&00,&00,&00
655 DATA <<<,&00,&08,&14,&22,&41,&00,&00
660 DATA ==,&00,&14,&14,&14,&14,&14,&00
665 DATA >>>,&00,&00,&41,&22,&14,&08,&00
670 DATA ???,&00,&02,&01,&51,&09,&05,&00
675 DATA 000,&00,&0A,&55,&55,&55,&28,&00
680 DATA AAA,&00,&7E,&09,&09,&09,&7E,&00
685 DATA BBB,&00,&41,&7F,&49,&49,&36,&00
690 DATA CCC,&00,&3E,&41,&41,&41,&22,&00
695 DATA DDD,&00,&41,&7F,&41,&41,&3E,&00
700 DATA EEE,&00,&7F,&49,&49,&49,&41,&00
705 DATA FFF,&00,&7F,&09,&09,&09,&01,&00
710 DATA GGG,&00,&3E,&41,&49,&49,&39,&00
715 DATA HHH,&00,&7F,&08,&08,&08,&7F,&00
720 DATA III,&00,&00,&41,&7F,&41,&00,&00
725 DATA JJJ,&00,&20,&40,&40,&40,&3F,&00
730 DATA KKK,&00,&7F,&08,&14,&22,&41,&00
735 DATA LLL,&00,&7F,&40,&40,&40,&40,&00
740 DATA MMM,&00,&7F,&02,&0C,&02,&7F,&00
745 DATA NNN,&00,&7F,&06,&08,&30,&7F,&00
750 DATA OOO,&00,&7F,&41,&41,&41,&7F,&00
755 DATA PPP,&00,&7F,&09,&09,&09,&06,&00
760 DATA QQQ,&00,&3E,&41,&51,&21,&5E,&00
765 DATA RRR,&00,&7F,&09,&19,&29,&46,&00
770 DATA SSS,&00,&26,&49,&49,&49,&32,&00
775 DATA TTT,&00,&01,&01,&7F,&01,&01,&00
780 DATA UUU,&00,&3F,&40,&40,&40,&3F,&00

```



```

785 DATA VVV,&00,&07,&18,&60,&18,&07,&00
790 DATA WWW,&00,&3F,&40,&38,&40,&3F,&00
795 DATA XXX,&00,&63,&14,&08,&14,&63,&00
800 DATA YYY,&00,&03,&04,&78,&04,&03,&00
805 DATA ZZZ,&00,&61,&51,&49,&45,&43,&00
810 DATA [[I,&00,&79,&14,&12,&14,&79,&00
815 DATA \\I,&00,&3D,&42,&42,&42,&3D,&00
820 DATA JJJ,&00,&3D,&40,&40,&40,&3D,&00
825 DATA ^^^,&00,&08,&04,&02,&04,&08,&00
830 DATA _,&00,&40,&40,&40,&40,&40,&00
835 DATA ``,`&00,&00,&02,&04,&00,&00,&00
840 DATA aaa,&00,&38,&44,&44,&3C,&40,&00
845 DATA bbb,&00,&7F,&44,&44,&44,&38,&00
850 DATA ccc,&00,&3B,&44,&44,&44,&44,&00
855 DATA ddd,&00,&38,&44,&44,&44,&7F,&00
860 DATA eee,&00,&3B,&54,&54,&54,&4B,&00
865 DATA fff,&00,&08,&7E,&09,&01,&02,&00
870 DATA ggg,&00,&18,&24,&24,&24,&7C,&00
875 DATA hhh,&00,&7F,&04,&04,&04,&78,&00
880 DATA iii,&00,&00,&44,&7D,&40,&00,&00
885 DATA jjj,&00,&40,&00,&04,&7D,&00,&00
890 DATA kkk,&00,&7F,&10,&28,&44,&00,&00
895 DATA lll,&00,&40,&41,&7F,&40,&40,&00
900 DATA mmm,&00,&7C,&04,&7C,&04,&78,&00
905 DATA nnn,&00,&7C,&08,&04,&04,&78,&00
910 DATA ooo,&00,&3B,&44,&44,&44,&38,&00
915 DATA ppp,&00,&7C,&24,&24,&24,&18,&00
920 DATA qqq,&00,&18,&24,&24,&24,&7C,&00
925 DATA rrr,&00,&00,&7C,&08,&04,&04,&00
930 DATA sss,&00,&48,&54,&54,&54,&24,&00
935 DATA ttt,&00,&04,&3F,&44,&40,&00,&00
940 DATA uuu,&00,&3C,&40,&40,&40,&7C,&00
945 DATA vvv,&00,&0C,&30,&40,&30,&0C,&00
950 DATA www,&00,&3C,&40,&30,&40,&3C,&00
955 DATA xxx,&00,&44,&28,&10,&28,&44,&00
960 DATA yyy,&00,&0C,&50,&20,&10,&0C,&00
965 DATA zzz,&00,&44,&64,&54,&4C,&44,&00
970 DATA {I,&00,&38,&45,&44,&3d,&40,&00
975 DATA *?*,&00,&39,&44,&44,&44,&39,&00
980 DATA }}},&00,&3D,&40,&40,&40,&7D,&00
985 DATA ~~~,&00,&7E,&01,&21,&25,&1A,&00
990 END ' *****

```

Wenn Sie das Programm abgetippt haben, können Sie es durch RUN starten.

Das Programm gibt kurze Hinweise auf seine Funktionen und Bedienung aus und erwartet von Ihnen die Eingabe der gewünschten Länge der Druckzeile (normalerweise ist dies der Wert 80) und des Namens der auszudruckenden Datei. Zudem werden Sie nach der Seitenlänge gefragt. Das ist die Zahl der Zeilen, die auf eine Seite passen sollen. Wird beim Ausdrucken dieser Wert erreicht, fordert Sie das Programm auf, ein neues Blatt einzulegen oder das Endlospapier weiterzukurbein und eine Taste zu drücken. 47 Zeilen pro Seite sind ein, in den meisten Fällen, geeigneter Wert.

Sie können sogar das Programm durch sich selbst ausdrucken lassen! Speichern Sie es zum Beispiel mit SAVE "SEIKOSHA.BAS", A als Datei im ASCII-Format und starten Sie es.

Als Dateinamen geben Sie dann SEIKOSHA.BAS an, als Zeilenlänge 80 und als Seitenlänge 47. Wenn Sie das Programm vorher schon mit LIST #8 ausgedruckt haben, werden Sie den erheblichen Unterschied im Schriftbild bestimmt bemerken!

Das ist alles, was Sie zur Bedienung wissen müssen.

Wenn Sie sich auch für die Funktionsweise interessieren, können Sie hier weiterlesen: Das Programm enthält alle Zeichendefinitionen in den Zeilen 515 bis 985 in hexadezimaler Schreibweise. Damit das Listing übersichtlicher ist, steht vor jeder Definition das zugehörige Symbol. Für die "kritischen" Buchstaben mit Unterlängen sind in den DATA-Zeilen nur die oberen Hälften der Zeichen angegeben. Um das Programm zu beschleunigen, werden in den Zeilen 290 bis 325 alle Zeichendefinitionen in das String-Datenfeld `chr$()` eingelesen. Das Hauptprogramm befindet sich in den Zeilen 335 bis 505:

Die Zeile 335 öffnet die einzulesende Datei. In der WHILE-WEND-Schleife zwischen den Zeilen 340 bis 500 wird je ein Datensatz eingelesen und so in Portionen zerteilt, daß immer nur so viele Zeichen übrigbleiben, wie sie die Zeilenlänge erlaubt. In der FOR-NEXT-Schleife zwischen den Programmzeilen 365 und 400 nimmt der Computer vom String immer ein Zeichen ab und fragt ab, ob es sich um ein Steuerzeichen mit einem ASCII-Code zwischen 0 und 31 handelt. In diesem Fall wird das Zeichen unverändert dem Drucker übermittelt. Schließlich sollen ja Codes wie CR (Wagenrücklauf) oder LF (Line-Feed) vom Drucker ausgeführt und nicht in Grafiken umgesetzt werden!

Für die übrigen Zeichen ist die Zeile 380 von Bedeutung: in ihr schickt der Computer den Code für die hochauflösende Grafik - `CHR$(27)+CHR$(75)+CHR$(0)+CHR$(6)` - und den im Stringarray `chr$()` gespeicherten Code an den Drucker. Wenn es sich um ein Zeichen mit Unterlängen handelt (Zeile 395) schreibt das Programm in einen Hilfsstring `ul$` dessen ASCII-Code, andernfalls ein Leerzeichen.

Ist der Computer mit einer Druckzeile fertig, wertet das Programm in den Zeilen 410 bis 455 diesen String aus:

Nach einem Wagenrücklauf, der ja im Grafikmodus einen geringeren Abstand der Zeilen als im Textmodus verursacht (Zeile 405), werden die einzelnen Unterlängen, passend zum jeweiligen Buchstaben, ausgedruckt und erscheinen unmittelbar unter der oberen Hälfte des Buchstabens. Auch nach dieser Schleife sorgt ein PRINT #8 in der Zeile 460 für einen Wagenrücklauf. In den Zeilen 465 bis 495 trägt das Programm dafür Sorge, daß Sie rechtzeitig in die Lage versetzt werden, ein neues Blatt Papier einlegen zu können.

Sobald das Ende der Datei erreicht ist, die Bedingung WHILE NOT EOF also nicht mehr erfüllt wird, verläßt der Basic-Interpreter die Schleife und springt in die Zeile 505. Dort schließt er die Datei, löscht den Bildschirm, setzt die PEN-Farbe auf den Normal-

wert und beendet das Programm.

Falls Ihnen einige der Zeichen nicht gefallen oder Sie noch andere Zeichen auf dem Drucker ausgeben wollen, können Sie das Hilfsprogramm "TRANSFORMER" verwenden. Es wandelt den internen Code, mit dem der Schneider-CPC Bildschirmzeichen definiert, in den Druckercode um.

Der Computer berechnet für die Bildschirmausgabe alle Zeichen zeilenweise:

```

. * * * * . . = 01111100 = &7C
* * . . . * * = 11000110 = &C6
* * . * * * . = 11011110 = &DE
* * . * * * . = 11011110 = &DE
* * . * * * . = 11011110 = &DE
* * . . . . . = 11000000 = &C0
. * * * * . . = 01111100 = &7C
. . . . . = 00000000 = &00

```

Im Gegensatz dazu verlangt der Drucker die Zeichendefinitionen in Spaltenform:

```

. * * * * . .
* * . . . * *
* * . * * * .
* * . * * * .
* * . * * * .
* * . . . . .
. * * * * . .
. . . . .

! ! ! ! ! ! !
&!&!&!&!&!&!&
7!F!8!B!B!F!7!0
C!E!2!A!A!A!8!0
! ! ! ! ! ! !

```

Da die Umrechnungen zwischen den beiden Darstellungsweisen recht kompliziert und vor allem zeitaufwendig sind, liegt es nahe, diese stumpfsinnige Arbeit dem Computer zu überlassen.

Das Programm TRANSFORMER können Sie dazu verwenden.

Es fragt Sie nach den acht Byte-Werten für jede einzelne Rasterzelle des Zeichens. Nach kurzer Rechenzeit gibt es das Symbol in Blockgrafik normal und - für den Drucker passend - um 90 Grad gedreht aus.

Zum Schluß meldet es noch alle acht Hexadezimalwerte, mit denen Sie das Zeichen auf dem Drucker ausgeben können. Es versucht in der Folge, das Symbol auszudrucken. Schalten Sie dazu bitte Ihren Drucker ein, weil das Programm wartet, bis es die Zeichendefini-

tion absetzen kann. TRANSFORMER ist in Basic geschrieben und sieht so aus:

#### 14.2 Listing: TRANSF.BAS

```

100 ' TRANSFORMER: Setzt den internen Zeichencode in
110 '           den Druckercode fuer hochaufloesene Grafik um
120 '
130 ' *****
140 '
150 ' Bitte setzen Sie in Zeile 730 den Code Ihres Druckers ein!
160 '
170 ' *****
180 '
190 ' Eingabe des Codes -----
200 MODE 1:INK 1,0:INK 0,13:BORDER 10:PEN 1:PAPER 0:PRINT:PRINT
210 PRINT " ZEICHENTRANSFORMATION FUER DEN DRUCKER"
220 PRINT:PRINT STRING$(40,154):WINDOW 1,40,7,24
230 FOR i=1 TO 8
240   PRINT TAB(4):RIGHT$(STR$(i),1);".Code: ";
250   INPUT a(i)
260   b$(i)=BIN$(a(i),8)
270 NEXT i
280 ' Umsetzung in ein Binaerfeld -----
290 FOR i=1 TO 8
300   FOR j=1 TO 8
310     char$(i,j)=MID$(b$(i),j,1)
320   NEXT j
330 NEXT i
340 ' Ausgabe als Blockgrafikzeichen -----
350 PRINT:PRINT
360 FOR i=1 TO 8
370   PRINT TAB(10);
380   FOR j=1 TO 8
390     IF char$(i,j)="0" THEN PRINT " "; ELSE PRINT CHR$(233);
400   NEXT j
410   PRINT
420 NEXT i
430 ' Drehung des Zeichens um 90 Grad -----
440 FOR i=1 TO 8
450   FOR j=1 TO 8
460     char2$(j,i)=char$(i,j)
470   NEXT j
480 NEXT i
490 ' Umwandlung in 8-Bit-Binaerzahlen -----
500 FOR i=1 TO 8
510   FOR j=1 TO 8
520     brcode$(i)=bincode$(i)+char2$(i,j)
530   NEXT j
540 NEXT i
550 ' Ausgabe des gedrehten Zeichens -----
560 SYMBOL AFTER 48
570 SYMBOL 48,0,0,0,0,0,0,0,0
580 SYMBOL 49,0,126,126,126,126,126,126
590 FOR i=1 TO 8
600   PRINT TAB(10);brcode$(i)

```

```

610 NEXT I
620 SYMBOL AFTER 256
630 ' Umwandlung in Hexcodes -----
640 FOR i=1 TO 8
650   hexcode$(i)=HEX$(VAL("&X"+bincode$(i)),2)
660 NEXT i
670 ' Ausgabe der Hexcodes -----
680 PRINT:PRINT:PRINT " Hexadezimale Druckerodes: ":PRINT
690 FOR i=1 TO 8
700   PRINT TAB(20);"&";hexcode$(i)
710 NEXT i
720 ' Ausgabe des definierten Zeichens auf dem Drucker -----
730 PRINT #8,CHR$(27);CHR$(75);CHR$(8);CHR$(8);
740 FOR i=1 TO 8
750   PRINT #8,CHR$(VAL("&"+hexcode$(i)));
760 NEXT i
770 PRINT:PRINT " Bitte druecken Sie eine Taste!":PRINT:CALL &BB18
780 CLEAR:RUN
790 END ' *****

```

Entsprechend dem benutzten Drucker müssen Sie die Zeile 730, die die Escape-Sequenz für die hochauflösende Grafik ausgibt, abändern. Im Programmlisting steht der Code für den Schneider-Drucker NLQ401 und den baugleichen Brother M-1009.

Falls Sie den Seikosha-Drucker GP-500CPC verwenden, sollten Sie die Zeile so ändern:

```
730 PRINT #8,CHR$(27);CHR$(75);CHR$(8);CHR$(8);
```

Besitzen Sie einen anderen Drucker, müssen Sie gegebenenfalls in Ihrem Drucker-Handbuch nachsehen, um auf die Steuercode-Sequenz zu kommen.

Da der Schneider-CPC leider nur einen 7-Bit-Druckerausgang hat, kommen Zeichen, bei denen die oberste Rasterzeile belegt ist, nur verstümmelt auf dem Drucker an.

Es gibt zwei Wege zur Abhilfe: Entweder Sie verwenden diese oberste Zeile einfach nicht oder Sie basteln sich einen echten 8-Bit-Druckerausgang.

Durch die klare Gliederung und die große Anzahl an REM-Zeilen dürfte das Verständnis der Programmfunktionen keine unlösbaren Schwierigkeiten mit sich bringen:

- Zeile 190-270: EINGABE DES CODES gibt das Titelbild aus und erfragt von Ihnen die acht Bytes, die auf dem Schneider-CPC ein Zeichen definieren. Sie können Ihre Eingaben dezimal (zum Beispiel 123), hexadezimal (&4F) oder binär (&X0101001) machen. Das Programm wandelt Ihre Eingabe sowieso in einen Binärstring um.

- Zeile 280-330: UMSETZUNG IN EIN BINÄRFELD zerlegt die jeweils acht Zeichen langen Strings des eindimensionalen Feldes b\$() so,

daß jedes Feldelement des zweidimensionalen Arrays `char$(,)` ein Bit enthält, das darüber entscheidet, ob ein Rasterpunkt des Zeichens ein- oder ausgeschaltet ist:

```
b$(1)="10110110"
b$(2)="01010100"
```

Daraus folgt für `char$(,)`:

<code>char\$(1,1)="1"</code>	<code>char\$(2,1)="0"</code>
<code>char\$(1,2)="0"</code>	<code>char\$(2,2)="1"</code>
<code>char\$(1,3)="1"</code>	<code>char\$(2,3)="0"</code>
<code>char\$(1,4)="1"</code>	<code>char\$(2,4)="1"</code>
<code>char\$(1,5)="0"</code>	<code>char\$(2,5)="0"</code>
<code>char\$(1,6)="1"</code>	<code>char\$(2,6)="1"</code>
<code>char\$(1,7)="1"</code>	<code>char\$(2,7)="0"</code>
<code>char\$(1,8)="0"</code>	<code>char\$(2,8)="0"</code>

- Zeile 340-420: AUSGABE ALS BLOCKGRAFIKZEICHEN druckt auf dem Bildschirm für jedes Null-Bit ein Leerzeichen und für jede Eins das ASCII-Zeichen 233, einen ausgefüllten Block. So können Sie sehen, wie Ihr eingegebenes Zeichen achtfach vergrößert aussieht.

- Zeile 430-480: DREHUNG DES ZEICHENS UM 90 GRAD verwandelt die Bildschirm-Binärdarstellung in die Drucker-Binärdarstellung. Dies geht ganz einfach, da das Matrixfeld quadratisch ist. Durch Vertauschen der Indices - aus `char$(3,4)` wird `char2$(4,3)` - setzt das Programm das gesamte Zeichen in den Printercode um.

- Zeile 490-540: UMWANDLUNG IN 8-BIT-BINÄRZAHLEN "bastelt" aus den einzelnen Bits des zweidimensionalen Feldes `char2$(,)` wieder achtstellige Binärzahlen im eindimensionalen Stringfeld `bincode$( )` zusammen.

- Zeile 550-620: AUSGABE DES GEDREHTEN ZEICHENS zeigt, wie das Zeichen vom Computer gedreht wurde: die ehemals rechte Seite ist jetzt unten, die obere Seite ist die rechte Seite usw.

- Zeile 630-660: UMWANDLUNG IN HEXCODES macht aus den Binärzahlen in `bincode$( )` wieder Hexadezimalzahlen im Feld `hexcode$( )`.

- Zeile 670-710: AUSGABE DES HEXCODES druckt alle acht Hexadezimalzahlen, die das Zeichen für den Drucker beschreiben, aus. Sie können Sie dann aufschreiben oder auch auswendiglernen.

- Zeile 720-780: AUSGABE DER DEFINIERTEN ZEICHEN AUF DEM DRUCKER schickt das Zeichen in hochauflösender Grafik an den Druckerausgang, so daß der Drucker - natürlich unter der Voraussetzung, daß der Steuercode in der Zeile 730 einwandfrei ist - das Zeichen aufs Papier "plotten" kann. Der Computer wartet dann auf einen Tastendruck von Ihnen und startet das Programm neu.

Abbrechen können Sie die Arbeit mit TRANSFORMER jederzeit durch zweimaliges Drücken der ESCAPE-Taste.

## 15. Zwei Zeichensätze für den Seikosha-Drucker

Laut Handbuch besitzt der Seikosha GP-500CPC-Matrixdrucker die Möglichkeit, zwischen verschiedenen nationalen Zeichensätzen hin- und herzuschalten. Doch wer sich das Gerät anschaut, kann lange nach den zugehörigen DIP-Schaltern suchen: Sie sind einfach nicht zu entdecken. Vielleicht stecken sie irgendwo im Gerät. Ohne es zu zerlegen, ist da aber nichts zu machen. Doch wer verliert dadurch schon gerne seine Garantie auf den Drucker?

Andererseits stört der voreingestellte deutsche Zeichensatz beim Ausdrucken von Programmlistings ganz erheblich: Der Klammeraffe wird zum Paragraphenzeichen, und die eckigen und geschweiften Klammern gibt der Drucker als Umlaute wieder. Wer schon einmal solche mit Sonderzeichen gespickten Listings eintippen mußte, wird wissen, welche Mühe es bedeutet, ständig in Zeichensatz-Tabellen nachzusehen, um zu erfahren, welches internationale Symbol auf der Tastatur zum abgedruckten deutschen Zeichen paßt.

Da nun einmal der Seikosha-Drucker in der Lage ist, hochauflösende Grafik aufs Papier zu bringen, mußte es eigentlich möglich sein, auch die amerikanischen und damit internationalen Zeichen auszudrucken.

Das nachfolgend aufgelistete Programm zeigt, wie sich das erreichen läßt. Wahlweise können Sie den Quellcode des Maschinenprogramms eingeben oder den Basic-Lader:

### 15.1 Assemblerlisting: IZGP500

```
; *****
; *
; *   Internationaler Zeichensatz auf dem Seikosha GP-500CPC   *
; *
; *****

ORG  &A000

;
; **** Daten und Definitionen *****
;

PCHAR EQU  &BD2B      ; MC PRINT CHAR
PCHAR2 EQU &BD2C      ; PCHAR+1

ASC64  DEFB &1B,&4B,&00,&06,&3E,&41,&5D,&55,&5E,&00
ASC91  DEFB &1B,&4B,&00,&06,&00,&7F,&41,&41,&00,&00
ASC92  DEFB &1B,&4B,&00,&06,&02,&04,&0B,&10,&20,&00
ASC93  DEFB &1B,&4B,&00,&06,&00,&41,&41,&7F,&00,&00
ASC123 DEFB &1B,&4B,&00,&06,&08,&36,&41,&41,&00,&00
ASC124 DEFB &1B,&4B,&00,&06,&00,&00,&77,&00,&00,&00
ASC125 DEFB &1B,&4B,&00,&06,&00,&41,&41,&36,&08,&00,&00
ASC126 DEFB &1B,&4B,&00,&06,&04,&02,&04,&08,&04,&00

PBUF  DEFS 3          ; Speicher für den MC PRINT CHAR-Vektor
ACTIVE DEFS 1         ; Flag, ob amerikanischer Zeichensatz aktiv
```

```

WIDTH  DEFB 14          ; Flag für Schriftbreite

;
; **** Initialisierungsroutine ****
;

INIT  LD  HL,PCHAR      ; Zeiger auf MC PRINT CHAR
      LD  DE,PBUFF      ; Zeiger auf zugehörigen Pufferspeicher
      LD  BC,3          ; 3 Bytes kopieren
      LDIR              ; Block verschieben

      LD  A,&C9          ; Z80-Code für RET
      LD  (INIT),A      ; RET in INIT unterbindet zweiten Aufruf
      LD  A,0           ; Akku=0
      LD  (ACTIVE),A    ; ACTIVE-Flag auf Null setzen

      LD  A,&C3          ; Z80-Code für JP
      LD  (PCHAR),A     ; nach MC PRINT CHAR
      LD  HL,PNEW       ; Zeiger auf neuen Druckertreiber
      LD  (PCHAR2),HL   ; nach MC PRINT CHAR +1
      RET              ; Rücksprung nach Basic

;
; **** Ersatz für die Druckeroutine ****
;

PNEW  CP  250           ; CHR(250) schaltet deutschen Zeichensatz ein
      JR  NZ,TST251     ; Überspringen, wenn nicht 250
      SUB A             ; Akku=0
      LD  (ACTIVE),A    ; ACTIVE-Flag löschen
      SCF              ; Carry-Flag für erfolgreiche Zeichenausgabe
      RET              ; Rücksprung zum aufrufenden Programm

TST251 CP  251          ; CHR(251) schaltet den US-Zeichensatz ein
      JR  NZ,TSTWID     ; Wenn nicht 250, dann Überspringen
      LD  A,255         ; Akku auf logisch Eins
      LD  (ACTIVE),A    ; ACTIVE-Flag setzen
      SCF              ; Carry-Flag für erfolgreiche Zeichenausgabe
      RET              ; Rücksprung zum aufrufenden Programm

TSTWID AND &7F          ; 7.Bit des Zeichens löschen
      CP  14            ; SD=CHR(14) ~ Doppelte Schriftbreite
      JR  NZ,TSTSMA     ; Wenn nicht, dann nach TEST SMALL
      LD  (WIDTH),A     ; Schriftbreite merken
      JR  IDTIFY        ; Und Test auf normale Schrift Überspringen

TSTSMA CP  15           ; SI=CHR(15) - Rückstellung auf Normalschrift
      JR  NZ,IDTIFY     ; Wenn nicht, dann zur Zeichenausgabe
      LD  (WIDTH),A    ; Schriftbreite merken
IDTIFY PUSH AF          ; Akkuinhalt retten
      LD  A,(ACTIVE)    ; Ist der amerikanische Zeichensatz aktiv?
      CP  0             ; Flag zur Prüfung setzen
      JR  NZ,USA        ; Aktiv, Sprung zur Ausgabe der US-Zeichen
      POP AF           ; Akkuinhalt wiederherstellen
      JP  PBUFF         ; Zeichen normal ausgeben

USA   POP AF           ; Akkuinhalt wiederherstellen

```



```

PUSH BC      ; BC-Register sichern
PUSH HL      ; HL-Register sichern

CP 64        ; Ein Klammerschließende ?
JR Z,CHAR64

CP 91        ; Eckige Klammer auf A?
JR Z,CHAR91

CP 92        ; Umgekehrter Schrägstrich ?
JR Z,CHAR92

CP 93        ; Eckige Klammer zu U?
JR Z,CHAR93

CP 123       ; Geschweifte Klammer auf A?
JR Z,CHR123

CP 124       ; Senkrechter Strich "8"?
JR Z,CHR124

CP 125       ; Geschweifte Klammer zu U?
JR Z,CHR125

CP 126       ; Tilde ?
JR Z,CHR126

NORMAL POP HL ; Kein Sonderzeichen - HL wiederherstellen
POP BC       ;
JP PBUF      ; BC wiederherstellen
              ; normal ausgeben

CHAR64 LD HL,ASC64 ; Zeiger auf ASCII-Tabelle 64
JR      OUTPUT

CHAR91 LD HL,ASC91 ; Zeiger auf ASCII-Tabelle 91
JR      OUTPUT

CHAR92 LD HL,ASC92 ; Zeiger auf ASCII-Tabelle 92
JR      OUTPUT

CHAR93 LD HL,ASC93 ; Zeiger auf ASCII-Tabelle 93
JR      OUTPUT

CHR123 LD HL,ASC123 ; Zeiger auf ASCII-Tabelle 123
JR      OUTPUT

CHR124 LD HL,ASC124 ; Zeiger auf ASCII-Tabelle 124
JR      OUTPUT

CHR125 LD HL,ASC125 ; Zeiger auf ASCII-Tabelle 125
JR      OUTPUT

CHR126 LD HL,ASC126 ; Zeiger auf ASCII-Tabelle 126

OUTPUT LD B,10      ; Schiebefenstärker, 10 Bytes auszugeben

OUTLP LD A,(HL)     ; Ein Byte des Hires-Zeichens lesen

```

```

CALL PBUFF      ; Druckerausgabe versuchen
JR NC,OUTLP     ; Solange versuchen, bis es klappt
INC HL          ; Adresszeiger erhöhen
DJNZ OUTLP      ; Schleifenzähler vermindern, bis fertig

LINDIF LD A,(WIDTH) ; Rückstellbefehl zur Zeilenlänge laden
CALL PBUFF      ; Zeichenausgabe versuchen
JR NC,LINDIF    ; Bis es klappt

PTRETN POP HL    ; HL-Register wiederherstellen
POP BC          ; BC-Register wiederherstellen
RET             ; Rücksprung zum aufrufenden Programm

END             ; -----

```

Der Basic-Lader verschiebt den Maschinencode immer direkt unter HIMEM, so daß das Programm mit anderen Maschinenprogrammen - zumindest von der Speicherbelegung her gesehen - zusammenarbeiten kann und nur 280 Bytes Speicherplatz kostet. Das Ladeprogramm gibt außerdem Hinweise zur Bedienung des Programms und initialisiert die Betriebssystem-Erweiterung.

## 15.2 Listing: IZGP500.BAS

```

100 ' *****
110 ' *
120 ' * SEIKOSHA GP-500CPC: Internationale Zeichen *
130 ' *
140 ' *****
150 '
160 DEF FNmsb(a)=255 AND INT(a/256)
170 DEF FNlsb(a)=255 AND UNT(a)
180 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
190 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
200 LOCATE 1,1:PRINT STRING$(40,210);
210 LOCATE 1,2
220 PRINT CHR$(24)+SPACE$(13)+"SEIKOSHA-PRINT"+SPACE$(13)+CHR$(24)
230 LOCATE 10,5:PRINT CHR$(164);" Martin Kotulla 1986"
240 PRINT:PRINT STRING$(40,210):PRINT
250 PEN 1:PRINT " Dieses Maschinenprogramm erlaubt den"
260 PRINT:PRINT " gemischten Ausdruck von deutschen und"
270 PRINT:PRINT " internationalen Sonderzeichen auf dem"
280 PRINT:PRINT " Seikosha-Drucker GP-500CPC:"
290 PEN 3:PRINT:PRINT STRING$(40,210):PRINT
300 PEN 2:PRINT " PRINT #8,CHR$(250); ";PEN 1:PRINT "Deutsch"
310 PEN 2:PRINT " PRINT #8,CHR$(251); ";PEN 1:PRINT "International"
320 PRINT
330 FOR i=40960 TO 41229:READ a:sum=sum+a:NEXT i
340 IF sum=21262 THEN 370
350 PRINT CHR$(7);"* Fehler in den DATA-Zeilen!":PRINT
360 LIST 450:END
370 RESTORE:MEMORY HIMEM-280:start=HIMEM+1
380 FOR i=start TO start+269:READ a:POKE i,a:NEXT i
390 FOR i=1 TO 22:READ a:a=a-40960+start

```

```

400 value=PEEK(a)+PEEK(a+1)*256-40960+start
410 POKE a,FNlsb(value):POKE a+1,FNmsb(value)
420 NEXT i
430 CALL start+85 ' Druckererweiterung initialisieren
440 ' Maschinencode-Daten *****
450 DATA &1B,&4B,&00,&06,&3E,&41,&5D,&55,&5E,&00,&1B,&4B,&00,&06,&00,&7F
460 DATA &41,&41,&00,&00,&1B,&4B,&00,&06,&02,&04,&08,&10,&20,&00,&1B,&4B
470 DATA &00,&06,&00,&41,&41,&7F,&00,&00,&1B,&4B,&00,&06,&08,&36,&41,&41
480 DATA &00,&00,&1B,&4B,&00,&06,&00,&00,&77,&00,&00,&00,&1B,&4B,&00,&06
490 DATA &00,&41,&41,&36,&08,&00,&1B,&4B,&00,&06,&04,&02,&04,&08,&04,&00
500 DATA &00,&00,&00,&00,&0E,&21,&2B,&BD,&11,&50,&A0,&01,&03,&00,&ED,&B0
510 DATA &3E,&C9,&32,&55,&A0,&3E,&00,&32,&53,&A0,&3E,&C3,&32,&2B,&BD,&21
520 DATA &76,&A0,&22,&2C,&8D,&C9,&FE,&FA,&20,&06,&97,&32,&53,&A0,&37,&C9
530 DATA &FE,&FB,&20,&07,&3E,&FF,&32,&53,&A0,&37,&C9,&E6,&7F,&FE,&0E,&20
540 DATA &05,&32,&54,&A0,&18,&07,&FE,&0F,&20,&03,&32,&54,&A0,&F5,&3A,&53
550 DATA &A0,&FE,&00,&20,&04,&F1,&C3,&50,&A0,&F1,&C5,&E5,&FE,&40,&28,&21
560 DATA &FE,&5B,&28,&22,&FE,&5C,&28,&23,&FE,&5D,&28,&24,&FE,&7B,&28,&25
570 DATA &FE,&7C,&28,&26,&FE,&7D,&28,&27,&FE,&7E,&28,&28,&E1,&C1,&C3,&50
580 DATA &A0,&21,&00,&A0,&18,&21,&21,&0A,&A0,&18,&1C,&21,&14,&A0,&18,&17
590 DATA &21,&1E,&A0,&18,&12,&21,&28,&A0,&18,&0D,&21,&32,&A0,&18,&08,&21
600 DATA &3C,&A0,&18,&03,&21,&46,&A0,&06,&0A,&7E,&CD,&50,&A0,&30,&FA,&23
610 DATA &10,&F7,&3A,&54,&A0,&CD,&50,&A0,&30,&F8,&E1,&C1,&C9,&00
620 ' Daten zum Verschieben des Maschinencodes *****
630 DATA &A059,&A063,&A068,&A070,&A07C,&A087,&A092,&A09B,&A09F,&A0A7
640 DATA &A0CF,&A0D2,&A0D7,&A0DC,&A0E1,&A0E6,&A0EB,&A0F0,&A0F5,&A0FB
650 DATA &A103,&A106
660 END ' *****

```

Übliche Betriebssystem-Erweiterungen werden über RSX-Befehle ("Resident System Extensions") gesteuert. Doch diese wären in diesem Fall nur recht kompliziert handzuhaben. Stattdessen wird die Zeichensatz-Auswahl durch PRINT#8-Befehle gesteuert! Damit verhält sich der Drucker für den Anwender praktisch so, als verstände er die Auswahlbefehle für die nationalen Zeichensätze.

Ein PRINT #8,CHR\$(250) schaltet den deutschen Zeichensatz ein. Dies ist der Normalzustand, es können das Paragraphensymbol (§), die Umlaute und das Eszet (ß) dargestellt werden.

PRINT #8,CHR\$(251) hingegen aktiviert das Maschinenprogramm.

Alle ASCII-Codes, deren Nummer mit der der deutschen Zeichen kollidiert, werden abgefangen und als amerikanische Zeichen ausgegeben. Dies betrifft die folgenden Symbole:

	Deutsch	International
ASCII-Code 64	§	@
ASCII-Code 91	Å	{
ASCII-Code 92	ß	\
ASCII-Code 93	Ü	}
ASCII-Code 123	ä	{
ASCII-Code 124	ö	
ASCII-Code 125	u	}
ASCII-Code 126	ß	~

Ein kleines Beispielprogramm mag die Funktionsweise des Maschinencode-Programms verdeutlichen. Es wird dabei vorausgesetzt, daß Sie die Maschinenroutine bereits geladen haben:

### 15.3 Listing: ZEISADEM.BAS

```
100 PRINT #8,CHR$(250); ' deutscher Zeichensatz
110 FOR i=32 TO 126:PRINT #8,CHR$(i);:NEXT i
120 PRINT #8:PRINT #8
130 PRINT #8,CHR$(251); ' internationaler Zeichensatz
140 FOR i=32 TO 126:PRINT #8,CHR$(i);:NEXT i
150 PRINT #8:PRINT #8:PRINT #8
```

Das Programm schaltet in der Zeile 100 auf den deutschen Zeichensatz um und druckt in der Zeile 110 den gesamten ASCII-Zeichenvorrat mit den deutschen Sonderzeichen aus. Die Zeile 120 gibt zwei Leerzeilen aus, damit sich die beiden Zeichensatz-Teile besser voneinander abheben. Die Zeile 130 weist das Maschinenprogramm im Speicher an, den internationalen Zeichensatz anzuwählen.

In der Programmzeile 140 druckt der Computer den ASCII-Zeichensatz, diesmal aber mit den internationalen Symbolen.

Die Zeile 150 schließlich gibt drei Zeilenvorschub-Befehle an den Drucker. So können Sie das Resultat besser betrachten.

Die interne Funktionsweise des Maschinenprogramms dürfte am deutlichsten werden, wenn Sie sich den Quellcode ansehen: Das Label PCHAR wird in der Zeile 1080 auf die Vektoradresse &BD2B definiert.

Dort steht ein Sprung in die Betriebssystem-Routine MC PRINT CHAR. PCHAR2 entspricht PCHAR+1, also der Adresse &BD2C. Damit zeigt PCHAR2 auf die Sprungadresse im Vektor, die ja nach dem Z80-Sprungbefehl RST1 (RST &0000) zu finden ist.

Die Zeilen 1110 bis 1180 zeigen die Zeichendefinitionen für die internationalen Symbole, die statt der deutschen eingesetzt werden. Der Aufbau der DEFB-Zeilen ist immer gleich:

- |          |     |   |
|----------|-----|---|
| 1. Byte: | &1B | Ist das ESC-Zeichen und Einleitung für Drucker-Steuerbefehle.   |
| 2. Byte: | &4B | Ist der Steuercode, der den Seikosha-Drucker auf hochauflösende Grafik umschaltet.                                  |
| 3. Byte: | &00 | Ist das Highbyte der Zahl der folgenden Grafikbytes.  |
| 4. Byte: | &06 | Ist das Lowbyte der Zahl der folgenden Grafikbytes. Für jedes Zeichen müssen nämlich sechs Bytes ausgegeben werden. |

5. bis 10. Byte:                    Das sind die eigentlichen  
Zeichendefinitionen für den Drucker.

Die Marke PBUFF speichert den Vektor, der in MC PRINT CHAR stand, bevor der dortige Sprung auf das Maschinenprogramm umgebogen wurde. Alle tatsächlichen Druckerausgaben müssen natürlich jetzt von der Maschinenroutine aus mit CALL PBUFF und nicht mit CALL PCHAR aufgerufen werden. Andernfalls läuft das Programm in eine Endlosschleife, aus der es nur durch einen Reset "erlöst" werden kann.

Das Label ACTIVE dient als Flag, das anzeigt, ob der deutsche oder der amerikanische Zeichensatz aktiv ist. Ist ACTIVE=255, dann werden die amerikanischen Zeichen benutzt, bei ACTIVE=0 hingegen die deutschen. WIDTH wird vom Programm verwendet, um festzustellen, ob ein Befehl zur Umschaltung in die Normal- oder Breitschrift an den Drucker geschickt wird. Dies ist wichtig, weil die ASCII-Codes 14 und 15 auch zur Rückstellung vom Zeilenabstand bei der Grafikausgabe auf den normalen Zeichenabstand benötigt werden.

INIT ist die Initialisierungsroutine, die den Vektor in MC PRINT CHAR nach PBUFF kopiert. Zudem lädt sie nach dem ersten Aufruf den Einsprung ins Programm mit dem RET-Befehl des Z80-Prozessors, um einen zweimaligen versehentlichen Aufruf des Programms zu verhindern, der den Absturz des Computers zur Folge hätte: Dann würde ja der neue Vektor in MC PRINT CHAR als der alte angesehen und nach PBUFF kopiert. Dort überschreibe er den richtigen Vektor, und das Programm liefere sich schließlich in einer Endlosschleife "tot". INIT trägt außerdem in MC PRINT CHAR den Z80-Befehl JP PNEW ein, der alle Druckerausgaben wie PRINT #8, LIST #8 und WRITE #8 auf das Maschinenprogramm umlenkt.

Nach Abarbeitung dieser Startbefehle kehrt INIT mit RET nach Basic zurück.

PNEW ist jetzt in die Druckerausgabe eingeklinkt und prüft in der Zeile 1400, ob ein Umschaltbefehl auf den deutschen Zeichensatz gegeben wurde.

In diesem Fall setzt PNEW das ACTIVE-Flag auf Null und setzt mit SCF das Carry-Flag. Das Carry-Flag wird vom aufrufenden Programm ausgewertet: Bei geglückter Zeichenausgabe ist es gesetzt. War der Drucker hingegen nicht empfangsbereit, wird es von MC PRINT CHAR gelöscht.

Da PNEW dem aufrufenden Programm vormachen muß, es hätte das Zeichen ausgegeben, ist SCF nötig.

TST251 stellt fest, ob ein CHR\$(251), der Umschaltcode für den internationalen Zeichensatz angekommen ist. Bei positiver Rückmeldung setzt es das ACTIVE-Flag auf 255, und das Programm kehrt nach Basic zurück.

TSTWID verknüpft das Zeichen, das im Akkumulator übergeben wurde, mit 127. So ist sichergestellt, daß das achte Bit des Zeichens gelöscht ist und nur sieben Bits übermittelt werden. Ferner wird

hier und bei TSTSMA (Test Small) nachgefragt, ob ein ASCII-Zeichen 14 oder 15 ausgegeben werden soll. Abhängig davon wird das WIDTH-Flag verändert.

IDTIFY entscheidet, ob ein Zeichen mit dem deutschen oder amerikanischen Zeichensatz gedruckt werden soll. Dazu zieht IDTIFY die Speicherstelle ACTIVE als Referenz heran. Merkt das Programm, daß die US-amerikanischen Zeichen verwendet werden sollen, springt es zum Label USA. Dort prüft es, ob das Zeichen eines der besonders zu berücksichtigenden, da umzucodierenden, ist.

Trifft dies nicht zu, kann das Symbol einfach bei der Marke NOR-MAL über die Standard-Routine abgeschickt werden.

Wurde allerdings eines der speziellen Zeichen entdeckt, springt der Prozessor zu einer der Marken CHAR64, CHAR91, CHAR92, CHAR93, CHR123, CHR124, CHR125 oder CHR126. In diesen wird jeweils das HL-Register der CPU mit einem Zeiger auf die entsprechenden Zeichencodes geladen, und der Computer springt zum Label OUTPUT. OUTPUT und die Fortsetzung OUTLP geben die insgesamt zehn Bytes jedes Sonderzeichens an den Drucker. LINDIF ("Line Difference") lädt dann den Rückstellbefehl, der wieder vom Zeilenabstand bei der Grafikausgabe auf den Textzeilenabstand umschaltet. Hier wird das vorher eventuell gespeicherte Byte in WIDTH benötigt. PTRETN stellt alle gesicherten Register wieder her und springt nach Basic zurück.

Das Programm ist im Ubrigen sehr einfach zu erweitern. Sie müssen nur am Quellcode gewisse Änderungen durchführen und diesen neu assemblieren. Wollen Sie zum Beispiel das Leerzeichen durch einen schwarzen Block ersetzen - ob das sinnvoll ist, ist eine andere Frage! -, setzen Sie dazu die folgenden Zeilen ein:

```
.
.
ASC126 DEFB &1B,&4B,&00,&06,&04,&02,&04,&08,&04,&00
ASC32  DEFB &1B,&4B,&00,&06,&7F,&7F,&7F,&7F,&7F,&7F
.
.
      CP    126
      JR    Z,CHR126
      CP    32
      JR    Z,CHAR32
.
.
CHAR64 LD    HL,ASC64
      JR    OUTPUT
CHAR32 LD    HL,ASC32
      JR    OUTPUT
CHAR91 LD    HL,ASC91
      JR    OUTPUT
.
.
```

Nach dieser Methode kann das Programm um beliebig viele Umcodierungen ergänzt werden, die aber alle genau zehn Bytes umfassen müssen.

## 16. Auch Maschinenprogramme lassen sich unterbrechen!

Jeder Basic-Programmierer weiß, daß sich Basic-Programme durch Drücken der ESCape-Taste anhalten und durch nochmaliges Drücken der Taste beenden lassen. Bei Maschinenprogrammen sieht es in dieser Hinsicht sehr schlecht aus: Befinden sie sich in einer endlosen Schleife, gibt es keine andere Möglichkeit, als einen Reset auszulösen.

In dieser Situation hat man dann meistens gerade keine Sicherheitskopien von den Daten und Programmen im Speicher gemacht - stundenlanges Neueintippen der Programme ist nicht unbedingt der Ausnahmefall!

Maschinenprogramme lassen sich nicht unterbrechen - diese unumstößliche Regel gilt schon seit Programmierer-Generationen. Doch gerade der Schneider-CPC macht das möglich! Denn er besitzt eine ausgezeichnete Interruptverarbeitung. Ahnen Sie es schon?

Richtig: Per Interrupt wird eine bestimmte Tastenkombination abgefragt. Gegebenenfalls springt dann das Programm in eine Unterbrechungsroutine und bei Bedarf in den Bildschirm-Editor des Basic-Interpreters zurück.

Das Programm "MCode-Breaker", das Sie entweder als Quellcode eintippen und assemblieren oder mit dem Basic-Ladeprogramm eingeben können, ist die Lösung: Es klinkt sich in die Interrupt-Kette ein und fragt dann ständig die Kombination der Tasten SHIFT und ESC ab.

So können alle Maschinencode-Routinen, auch solche in ROMs, unterbrochen werden, sofern diese nicht gerade mit dem Z80-Maschinenbefehl DI (Disable Interrupts) die Interrupts unterdrücken. DI wird aber kaum verwendet, da es eine Reihe von Problemen mit sich bringt, auch die normalen Interrupt-Routinen zu verbieten. So sind zum Beispiel der Floppy-Motor, die Tonausgabe und die Tastaturabfrage interruptgesteuert.

### 16.1 Assemblerlisting: MCBREAK

```

1000 ' ; *****
1010 ' ; *
1020 ' ; * MCODE-BREAK: Unterbrechung von MCode-Programmen mit SHIFT & ESC *
1030 ' ; *
1040 ' ; *****
1050 '
1060 '          ORG  &0160          ; Fester Einsprung bei &0160!
1070 '
1080 '          JP  INIT          ; Sprung zum tatsächlichen Programm
1090 '
1100 ' AFSAVE DEFS 2          ; Speicher fuer AF-Register
1110 ' BCSAVE DEFS 2          ; Speicher fuer BC-Register
1120 ' DESAVE DEFS 2          ; Speicher fuer DE-Register
1130 ' HLSAVE DEFS 2          ; Speicher fuer HL-Register
1140 ' SAvEXX DEFS 2          ; Speicher fuer Indexregister IX
1150 ' SAvERY DEFS 2          ; Speicher fuer Indexregister IY

```

```

1160 '
1170 ' UROMON EQU &B900 ; KL UPPER ROM ENABLE
1180 ' WAITC EQU &BB06 ; KM WAIT CHAR
1190 ' RDCHAR EQU &BB09 ; KM READ CHAR
1200 ' TSTKEY EQU &BB1E ; KM TEST KEY
1210 ' OUTPUT EQU &BB5A ; TXT OUTPUT
1220 ' PLACE EQU &BB8A ; TXT PLACE CURSOR
1230 ' REMOVE EQU &BB8D ; TXT REMOVE CURSOR
1240 ' CASINI EQU &BC65 ; CAS INITIALISE
1250 ' NEWFAS EQU &BCE0 ; KL NEW FAST TICKER
1260 ' DELFAS EQU &BCE6 ; KL DEL FAST TICKER
1270 ' READY EQU &C064
1280 ' ESCAPE EQU 66 ; Tastencode der ESC-Taste
1290 '
1300 ' ORG &A000 ; Mit dem Basic-Loader verschiebbar!
1310 '
1320 ' TICLST DEFW 0,0 ; Fast Ticker List - 4-Byte-Queue
1330 ' DEFB 0,83 ; Fortsetzung als Event-Block
1340 ' DEFW IRQ
1350 ' DEFS 2
1360 '
1370 ' ; ***** Initialisierung des Interrupts *****
1380 '
1390 ' INIT LD HL,TICLST ; Zeiger auf Ticker-Liste
1400 ' LD DE,IRQ ; Zeiger auf Interrupt-Routine
1410 ' LD B,&83
1420 ' LD C,&00
1430 ' CALL NEWFAS ; KL NEW FAST TICKER
1440 ' RET ; Rücksprung nach Basic
1450 '
1460 ' ; ***** Interrupt-Routine *****
1470 '
1480 ' IRQ DI ; Interrupts unterdrücken
1490 ' LD (BCSAVE),BC ; BC-Register sichern
1500 ' LD (DESAVE),DE ; DE-Register sichern
1510 ' LD (HLSAVE),HL ; HL-Register sichern
1520 ' LD (SAVERX),IX ; IX-Register sichern
1530 ' LD (SAVERY),IY ; IY-Register sichern
1540 ' PUSH AF ; > ersetzt den fehlenden Befehl
1550 ' POP HL ; > LD HL,AF
1560 ' LD (AFSAVE),HL ; AF-Register sichern
1570 '
1580 ' LD A,ESCAPE ; Interner Tastencode von ESCape
1590 ' CALL TSTKEY ; KM TEST KEY wie INKEY(x) in Basic
1600 ' JR Z,LDREGS ; Rücksprung, wenn nicht ESC gedrückt
1610 ' BIT 5,C ; Wurde SHIFT gedrückt?
1620 ' JR Z,LDREGS ; Rücksprung, wenn nicht SHIFT & ESC!
1630 '
1640 ' BRKEVE CALL RDCHAR ; Tastenpuffer von SHIFT & ESC leeren
1650 ' CALL PLACE ; TXT PLACE CURSOR
1660 ' CALL WAITC ; KM WAIT CHAR
1670 ' PUSH AF ; Akku vor TXT REMOVE CURSOR retten
1680 ' CALL REMOVE ; TXT REMOVE CURSOR
1690 ' POP AF ; Akkuinhalt wiederherstellen
1700 ' CP &FC ; Nochmals ESC gedrückt?
1710 ' JR NZ,LDREGS ; Nein - Rücksprung zum Programm
1720 '

```



```

1730 ' PRESSD LD A,10 ; Akku mit LF-Code laden
1740 ' CALL OUTPUT ; Zeilenvorschub durchführen
1750 ' CALL CASINI ; CAS INITIALISE aufrufen
1760 ' CALL UROMON ; Basic-ROM einschalten
1770 ' EI ; Interrupts wieder zulassen
1780 ' JP READY ; Sprung in den READY-Modus
1790 '
1800 ' LDREGS LD HL,(AFSAVE) ; HL mit Akku und Flags laden
1810 ' PUSH HL ; > ersetzt den fehlenden Befehl
1820 ' POP AF ; > LD AF,HL
1830 ' LD DE,(DESAVE) ; DE-Register wiederherstellen
1840 ' LD HL,(HLSAVE) ; HL-Register wiederherstellen
1850 ' LD BC,(BCSAVE) ; BC-Register wiederherstellen
1860 ' LD IX,(SAVERX) ; IX-Register wiederherstellen
1870 ' LD IY,(SAVERY) ; IY-Register wiederherstellen
1880 ' EI ; Interrupts wieder zulassen
1890 ' RET ; Rücksprung zum Maschinenprogramm
1900 '
1910 ' ; ***** TESTER: Beispielprogramm *****
1920 '
1930 ' TESTER LD A,42 ; ASCII-Code für "*"
1940 ' CALL OUTPUT ; Ausgeben
1950 ' JR TESTER ; Und dauernd wiederholen
1960 ' END
1970 '
1980 ' ; -----

```

Wenn Sie den Basic-Lader starten, können Sie angeben, an welche Adresse der Maschinencode geladen werden soll. Damit dürfte der MCode-Breaker mit fast allen anderen Maschinenprogrammen verträglich zusammenarbeiten. Der Lader setzt HIMEM auf den Wert Startadresse-1.

## 16.2 Listing: MCBREAK.BAS

```

100 ' *****
110 ' * *
120 ' * MCODE-BREAK *
130 ' * *
140 ' *****
150 '
160 ' Achtung Diskettenbenutzer: SAVE nur auf Cassette möglich!
170 '
180 DEF FNmsb(a)=255 AND INT(a/256)
190 DEF FNlsb(a)=255 AND UNT(a)
200 SYMBOL 255,&66,&0,&78,&C,&7C,&CC,&76,&0
210 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
220 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
230 ' Titelbild und Benutzeranweisungen *****
240 LOCATE 1,1:PRINT STRING$(40,210);
250 LOCATE 1,2
260 PRINT CHR$(24)+SPACE$(14)+"MCODE-BREAK"+SPACE$(15)+CHR$(24)
270 LOCATE 10,5:PRINT CHR$(164)+" Martin Kotulla 1986"
280 PRINT:PRINT STRING$(40,210)
290 PEN 1:PRINT " Dieses Programm erlaubt es Ihnen,"

```

```

300 PRINT:PRINT " laufende Maschinenprogramme zu"
310 PRINT:PRINT " unterbrechen, sofern das Programm"
320 PRINT:PRINT " Interrupts zul"+CHR$(255)+CHR$(177)+"t."
330 FOR i=40960 TO 41083:READ a:sum=sum+a:NEXT i
340 IF sum=13981 THEN 370
350 PRINT:PRINT CHR$(7); " * Fehler in den DATA-Zeilen!":PRINT
360 LIST 600-:END
370 PEN 3:PRINT STRING$(40,210):PEN 1
380 PRINT:LINE INPUT " Start an welcher Adresse? ",start$
390 start=VAL(start$):IF start<0 THEN start=start+65536
400 ' Maschinencode einlesen und Adressen anpassen *****
410 MEMORY start-1:RESTORE 600
420 FOR i=start TO start+123:READ a:POKE i,a:NEXT i
430 FOR i=1 TO 3:READ a:a=a-40960+start
440 value=PEEK(a)+PEEK(a+1)*256-40960+start
450 POKE a,FNlsb(value):POKE a+1,FNmsb(a)
460 NEXT i
470 ' Computer feststellen und an den CPC-664 bzw. CPC-6128 anpassen *****
480 vsn$=CHR$(&CD)+CHR$(&D)+CHR$(&B9)+CHR$(&3A)+CHR$(&2)+CHR$(&C0)
490 vsn$=vsns$+CHR$(&32)+CHR$(&66)+CHR$(&1)+CHR$(&C9)
500 CALL PEEK(@vsns$+1)+PEEK(@vsns$+2)*256 ' Computertyp erkennen
510 version=PEEK(&166)
520 IF version=0 THEN 550
530 POKE start+&59,&58:POKE start+&60,&C0 ' READY-Sprung bei 664/6128
540 ' Sprungvektor auf das Programm nach &0160 schreiben *****
550 POKE &160,&C3:POKE &161,FNlsb(start+8):POKE &162,FNmsb(start+8)
560 PEN 2:PRINT:PRINT " INIT: CALL &0160"
565 PRINT " TEST: CALL &";HEX$(start+&75)
570 PRINT " CONT: CALL &0160"
580 PEN 1:END
590 ' Maschinencode-Programm *****
600 DATA &00,&00,&00,&53,&16,&A0,&00,&00,&21,&00,&A0,&11,&16,&A0,&06,&83
610 DATA &0E,&00,&CD,&E0,&BC,&C9,&F3,&ED,&43,&65,&01,&ED,&53,&67,&01,&22
620 DATA &69,&01,&DD,&22,&6B,&01,&FD,&22,&6D,&01,&F5,&E1,&22,&63,&01,&3E
630 DATA &42,&CD,&1E,&BB,&20,&25,&CB,&69,&20,&21,&CD,&09,&BB,&CD,&8A,&BB
640 DATA &CD,&06,&BB,&F5,&CD,&8D,&BB,&F1,&FE,&FC,&20,&0F,&3E,&0A,&CD,&5A
650 DATA &BB,&CD,&65,&BC,&CD,&00,&B9,&FB,&C3,&64,&C0,&2A,&63,&01,&E5,&F1
660 DATA &ED,&5B,&67,&01,&2A,&69,&01,&ED,&4B,&65,&01,&DD,&2A,&6B,&01,&FD
670 DATA &2A,&6D,&01,&FB,&C9,&3E,&2A,&CD,&5A,&BB,&18,&F9
680 ' Daten zum Verschieben des MCode-Breakers *****
690 DATA 40964,40969,40972
700 END ' -----

```

Initialisiert wird die Interrupt-Routine mit CALL &0160. Diese Startadresse ist fest, damit Sie durch verschiedene Ladeadressen für das Programm nicht verwirrt werden.

Gerade in dieser "Krisensituation" hätte ein fehlerhafter Aufruf verheerende Folgen:

Die letzte Chance, Ihre Daten zu retten, wäre vertan!

Ab sofort können Sie mit SHIFT-ESC Maschinenprogramme anhalten. Ein weiteres ESC bricht das Programm ab und führt den Computer nach Basic zurück. Jeder andere Tastendruck setzt das unterbrochene Programm fort.

Testen läßt sich der MCode-Breaker mit dem Aufruf CALL test, wobei "test" für die Startadresse+75 steht. Beim Ansprung dieser Routine gerät der Computer in eine Endlosschleife, in der andauernd Sternchen (\*) ausgedruckt werden.

Wenn Sie SHIFT-ESC drücken, hält der Computer die Zeichenausgabe an. Sie sehen auch, daß sogar die Zeichenausgabe-Routine TXT OUTPUT (&BB5A), die vom Testprogramm verwendet wird, unterbrochen werden kann: Ab und zu wird beim Drücken von SHIFT-ESC die Druckroutine TXT OUTPUT so gestoppt, daß auch mal nur ein halber Stern auf dem Monitor zu entdecken ist.

Wenn Sie das Programm dann weiterarbeiten lassen, wird der Stern vollendet.

In der Break-Routine erscheint der Cursor, wie Sie ihn vom einmaligen Drücken der ESC-Taste in Basic her kennen. Drücken Sie wiederum ESC, wenn Sie in der Break-Routine sind, bricht der Computer die Ausgabe der Sternchen ab und kehrt in den Basic-Editor zurück.

Wenn Sie jetzt versuchen, Befehle einzugeben, werden Sie bemerken, daß der Computer die Kommandos ignoriert, ein LIST-, SAVE- oder RUN-Befehl führt zum Beispiel zu einem völlig unmotivierten SYNTAX ERROR. Es haben sich nämlich anscheinend irgendwelche internen Zeiger des Betriebssystems und des Basic-Interpreters verstellt.

Korrigieren läßt sich dies durch ein erneutes CALL &0160. Jetzt läßt sich das Programm wieder listen und abspeichern. Aber Achtung: Die Datenspeicherung mit der Diskettenstation kann zu Problemen führen; gehen Sie lieber auf Nummer Sicher und speichern Sie auf Cassette. Nun sollten Sie nicht unbedingt weiterarbeiten, sondern einen Reset auslösen, zum Beispiel mit Control-Shift-Escape. Daraufhin können Sie das Programm wieder laden, verändern und natürlich auch der Fehlerursache nachgehen.

Auch sollten Sie vor dem fälligen Reset nicht wieder Maschinenroutinen aufrufen, da der MCode-Breaker unwirksam geworden ist. Selbst eine erneute Initialisierung mit CALL &0160 ist - wiederum anscheinend aus Gründen, die in der internen Verwaltung zu suchen sind - nicht möglich.

Trotz dieser Einschränkungen ist das Programm "MCode-Breaker" eine wertvolle Hilfe. Sie können sicher leichter die angegebenen Vorsichtsregeln und Einschränkungen beachten, als wichtige und unwiederbringliche Daten und Programme zu verlieren!

Für die Maschinensprache-Freaks unter den Lesern, wird hier noch die Funktionsweise des "MCode-Breakers" erläutert: Der weder vom Basic-Interpreter noch vom Schneider-Betriebssystem benutzte Speicherbereich zwischen &0160 und &0170 dient als Speicher für die wichtigsten Programmdateien: JP INIT an der Adresse &0160 bietet einen festen Einsprung für die Initialisierungs-Routine, unabhängig von der tatsächlichen Ablageadresse für das Programm. AFSAVE speichert während der Interruptbearbeitung und in der Break-Routine den Akkumulator und das Flag-Register des Z80-Pro-

zessors, BCSAVE das Doppelregister BC, DESAVE das 16-Bit-Register DE und HLSAVE den "16-Bit-Akku" HL. In SAVERX merkt sich das Programm das IX-Indexregister, in SAVERY das Indexregister IY.

Beginnend mit der Zeile 1300 ist das tatsächliche Maschinenprogramm zu finden: TICLST wird vom Betriebssystem als "Fast-Ticker-List" verwendet. Die Daten unter der Marke TICLST dienen zur korrekten Abarbeitung des Interrupt-Programms. Interrupt-Listen sind beim Schneider-CPC recht flexibel zu handhaben, da sie eine Reihe von Möglichkeiten bieten.

TICLST ist folgendermaßen aufgebaut:

Die ersten beiden 16-Bit-Worte, also vier Bytes, sind für die "Fast-Ticker-List" reserviert. Sie werden vom Betriebssystem durch eine ROM-Routine automatisch auf die geeigneten Werte gebracht.

Die restlichen sechs Bytes sind ein ganz normaler Event-Block:

- Das erste Byte stellt den Ereignis-Zähler dar. Er wird benötigt, wenn Interrupts schneller gegeben werden, als sie bearbeitet werden können. Um die Abarbeitung der Interrupt-Routine sofort zu beginnen, ist hier der Wert &00 einzutragen.

- Das zweite Byte (&83) gibt die Ereignis-Klasse an. Die Zahl &83 sieht als binärer Wert so aus: &X10000011. Jedes Bit hat eine besondere Bedeutung:

Bit 0 legt fest, ob die Adresse der Interrupt-Service-Routine eine "Near Address" (Bit gesetzt) oder eine "Far Address" ist (Bit gelöscht). Nahe Adressen dürfen nur innerhalb der zentralen 32 Kilobyte RAM des Computers liegen, also im Speicherbereich zwischen &4000 (16384) und &C000 (49152). Da solche Routinen schneller bearbeitet werden können, ist beim Programmieren eine "Near Address" vorzuziehen.

Die Bits 1 bis 4 geben die Priorität eines synchronen Ereignisses an. Der Computer ist bei asynchronen Ereignissen gezwungen, diese unmittelbar zu bearbeiten. Synchrone Ereignisse werden hingegen erst einmal in eine Warteschlange eingeordnet und der Reihe nach abgearbeitet. Da der MCode-Breaker ein asynchrones Ereignis ist, sind die Prioritäts-Bits ohne Bedeutung.

Das Bit 5 der Ereignis-Klasse muß nach den Vorschriften der Amstrad-Programmierer immer auf Null gesetzt sein.

Das Bit 6 bestimmt, ob ein Ereignis "normal" oder "speziell" ist. Es gibt im Betriebssystem Wege, um normale Ereignisse zu sperren und spezielle Ereignisse weiterhin zuzulassen. Da der MCode-Breaker möglichst häufig abgearbeitet werden soll, wurde ein spezielles Ereignis gewählt.

Bit 7 sagt aus, ob ein Ereignis synchron oder asynchron ist. Wie bereits erwähnt, ist das Programm ein asynchrones Ereignis. Demnach muß das Bit auf Eins gesetzt werden. Synchrone Ereignisse

bedingen in diesem Bit eine Null.

Die Interrupt-Bearbeitung wird durch die Routine INIT gestartet, sie reiht den Programmteil IRQ in die Interrupt-Kette ein. Dazu kann die Routine KL NEW FAST TICKER benutzt werden. Diese erwartet im HL-Register einen Zeiger auf den Fast-Ticker-Block, im DE-Doppelregister einen Zeiger auf die Interrupt-Routine selbst, im B-Register das Byte für die Ereignis-Klasse und im C-Register die ROM-Auswahladresse. Da die Möglichkeit der Verwendung einer Near-Address genutzt wird, ist der Inhalt des C-Register ohne jegliche Bedeutung.

Die eigentliche Interrupt-Routine, die 300 Mal in der Sekunde vom Computer aufgerufen wird, ist ab dem Label IRQ zu finden. Zuerst unterdrückt sie alle Interrupts mit dem Z80-Maschinenbefehl DI und sichert alle Register. Dann fragt sie durch die ROM-Routine KL TEST KEY die Escape-Taste ab. Ist das Zero-Flag gelöscht, hat der Benutzer Escape gedrückt. Sofern auch noch das fünfte Bit im C-Register gesetzt ist, hat der Computer die Tastenkombination SHIFT & ESC entdeckt.

In diesem Fall leert das Programm bei der Marke BRKEVE ("Break Event") den Tastaturpuffer mit CALL KM READ CHAR, zaubert mit der Systemroutine CALL TXT PLACE CURSOR den Cursor auf den Bildschirm und wartet mit CALL KM WAIT CHAR auf einen Tastendruck.

Ist das wieder die Escape-Taste, führt das Programm einen Zeilen-vorschub aus, initialisiert die Cassettenverwaltung (CALL CAS INITIALISE), schaltet das Basic-ROM ein (CALL KL UPPER ROM ENABLE), erlaubt wieder die Interrupts und springt in den Ready-Modus des Basic-Interpreters; - diese liegt beim CPC-464 an der Adresse &C064, bei den Modellen CPC-664 und CPC-6128 hingegen bei &C058. Damit finden Sie sich nach einem Programmabbruch im Basic-Editor wieder.

Drücken Sie stattdessen eine andere Taste, stellt der Computer alle Prozessor-Register wieder her, erlaubt Interrupts und kehrt zum aufrufenden Betriebssystem zurück.

Das Beispielprogramm TESTER gibt endlos den ASCII-Code 42 (Sternchen) aus und kann nur durch einen Reset oder durch den MCode-Breaker abgebrochen werden.

## 17. DEFAULTS setzt den Computer zurück

Der Schneider-CPC bietet zahlreiche Möglichkeiten, das Betriebssystem den eigenen Bedürfnissen anzupassen.

So lassen sich die Bildschirmfarben verändern, die Tastaturbelegungen und Funktionstasten anpassen und zum Beispiel auch Änderungen am Text- und Grafikbildschirm durchführen.

Wenn Sie ein Programm unterbrechen, das intensiv davon Gebrauch macht, kann es sein, daß Sie nichts mehr sehen, weil die Pen-Farbe gleich der Paper-Farbe ist oder über die Tastatur nichts mehr eingeben können, weil etwa die Tasten unbelegt wurden oder die Anschlagsgeschwindigkeit der Tastatur mit SPEED KEY auf ein unmögliches Maß hochgeschraubt ist.

Sie können jetzt versuchen, einzeln alle Parameter auf die Standardwerte zurückzusetzen, etwa so:

```
INK 1,24:INK 0,1:PEN 1:PAPER 0:BORDER 1:MODE 1 ...
```

Es dürfte aber nahezu unmöglich sein, sich das alles zu merken und wirklich alle Systemdaten zurückzusetzen. Stattdessen könnten Sie die Initialisierungsroutinen der "System-Packs" des Betriebssystems aufrufen:

```
CALL &BB00:CALL &BB4E:CALL &BBBA:CALL &BBFF ...
```

Das ist auch nicht gerade das Wahre, oder? Eine falsche Hexziffer, und der Computer kann abstürzen oder sonstige merkwürdigen Reaktionen zeigen.

Verwenden Sie doch das Programm "DEFAULTS"! Dieses erweitert das Betriebssystem des Schneider-CPC um einen RSX-Befehl, der alle diese Initialisierungen automatisch durchführt.

"DEFAULTS" können Sie entweder als Quellcode eingeben und dann assemblieren oder gleich den Basic-Loader eintippen. Der Basic-Loader erzeugt verschiebbaren Maschinencode und legt das Programm unter HIMEM ab.

Er führt auch selbsttätig die Installation der RSX-Erweiterung durch. Zuerst der Quellcode.

### 17.1 Assemblerlisting: DEFAULTS

```
; *****
; *
; *          DEFAULTS - setzt den Computer weitmöglichst zurück      *
; *
; *
; *****
```

```

                ORG      &A000      ; Mit dem Basic-Loader relocatibel
;
; Definitionen *****
; -----
KM_INITIALISE   EQU      &BB00      ; Initialisiert Keyboard-Manager
TXT_INITIALISE  EQU      &BB4E      ; Initialisiert Text-Manager
GRA_INITIALISE  EQU      &BBBA      ; Initialisiert Graphics-Manager
SCR_INITIALISE  EQU      &BBFF      ; Initialisiert Screen-Manager
CAS_INITIALISE  EQU      &BC65      ; Initialisiert Cassette-Manager
SOUND_RESET     EQU      &BCA7      ; Setzt Tonverwaltung zurück
MC_RESET_PRINT  EQU      &BD28      ; Setzt Drucker-Indirection zurück
JUMP_RESTORE    EQU      &BD37      ; Sprungvektoren zurücksetzen
TXT_SET_PEN     EQU      &BB90      ; Pen-Farbe setzen
KL_LOG_EXTERNAL EQU      &BCD1      ; RSX-Erweiterung installieren
;
; RSX-Einbindung *****
; -----
RSX_INIT        LD        BC,JUMPTABLE ; Zeiger auf Sprungtabelle
                LD        HL,KERNEL_RAM ; Zeiger auf Hilfsspeicher
                CALL      KL_LOG_EXTERNAL ; RSX installieren
                RET        ; Basic-Rücksprung
;
; Daten zur Verwaltung der RSX-Erweiterung *****
; -----
JUMPTABLE       DEFW      NAMETABLE   ; Vektor auf Namenstabelle
                JP        DEFAULTS    ; Sprung ins Hauptprogramm

NAMETABLE       DEFW      "DEFAULT"   ; RSX-Name "DEFAULTS"
                DEFB      "S"&80      ;
                DEFB      &00         ; Tabellensende

KERNEL_RAM      DEFS      &04         ; 4 Bytes für interne Verwaltung
;
; Eigentlicher Programmcode *****
; -----
DEFAULTS        EQU      $           ; Einsprung ins Programm

RESET_PALETTE   DI        ; Interrupts verbieten
                LD        BC,&DF00    ; An die ROM-Select-Adresse
                OUT        (C),C      ; Null schreiben

RESET_PERIPH     LD        BC,&F8FF    ; Ein &FF auf dem Port &F800
                OUT        (C),C      ; setzt die Peripherie zurück

RESET_VECTORS    CALL      JUMP_RESTORE ; KL JUMP RESTORE
                CALL      MC_RESET_PRINT ; MC RESET PRINTER
                CALL      KM_INITIALISE ; KM INITIALISE
                CALL      TXT_INITIALISE ; TXT INITIALISE

```

```

CALL      GRA_INITIALISE ; GRA INITIALISE
CALL      SCR_INITIALISE ; SCR INITIALISE
CALL      CAS_INITIALISE ; CAS INITIALISE
CALL      SOUND_RESET    ; SOUND RESET
EI                      ; Interrupts zulassen

RESET_PEN LD      A,1      ; Die Pen-Nummer auf den PEN
CALL      TXT_SET_PEN    ; 1 setzen
RET                      ; Rucksprung nach Basic

END
; -----

```

Und so sieht der Basic-Loader aus:

## 17.2 Listing: DEFAULTS.BAS

```

100 ' *****
110 ' *****
120 ' *
130 ' *          DEFAULTS          *
140 ' *
150 ' *****
160 ' *****
170 '
180 DEF FNmsb(a)=255 AND INT(a/256)
190 DEF FNlsb(a)=255 AND UNT(a)
200 SYMBOL AFTER 254
210 SYMBOL 255,&66,&0,&66,&66,&66,&3E,&0
220 ' Titelbild und Benutzer-Informationen -----
230 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
240 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
250 LOCATE 1,1:PRINT STRING$(40,210);
260 LOCATE 1,2
270 PRINT CHR$(24)+SPACE$(16)+"DEFAULTS"+SPACE$(16)+CHR$(24)
280 LOCATE 10,5:PRINT CHR$(164);" Martin Kotulla 1986"
290 PRINT:PRINT STRING$(40,210):PRINT
300 PEN 1:PRINT:PRINT " Die RSX-Erweiterung !DEFAULTS setzt"
310 PRINT:PRINT " die System-Komponenten des Computers"
320 PRINT:PRINT " zur"+CHR$(255)+"ck."
330 PRINT:PEN 3:PRINT STRING$(40,210):PRINT
340 PEN 2:PRINT TAB(16);"!DEFAULTS"
345 ' Einlesen und Pruefen des Maschinenprogramms -----
350 FOR i=1 TO 74:READ a:sum=sum+a:NEXT i
360 IF sum=-89133 THEN 390
370 PRINT CHR$(7);" * Fehler in den DATA-Zeilen!":PRINT
380 LIST 490:END
390 RESTORE:MEMORY HIMEM-74:start=HIMEM+1
400 FOR i=start TO start+69:READ a:POKE i,a:NEXT i
405 ' Anpassung des Programms an die Ladeadresse -----
410 FOR i=1 TO 4:READ a:a=a-40960+start
420 value=PEEK(a)+PEEK(a+1)*256-40960+start
430 POKE a,FNlsb(value):POKE a+1,FNmsb(value)
440 NEXT i

```



```

450 CALL start ' RSX-Befehl installieren
460 ' Demoprogramm -----
470 FOR i=1 TO 6000:NEXT i
480 :DEFAULTS:END
490 ' DATAs fuer das Maschinenprogramm -----
500 DATA &01,&0A,&A0,&21,&18,&A0,&CD,&D1,&BC,&C9,&0F,&A0,&C3,&1C,&A0,&44
510 DATA &45,&46,&41,&55,&4C,&54,&D3,&00,&00,&00,&00,&F3,&01,&00,&DF
520 DATA &ED,&49,&01,&FF,&F8,&ED,&49,&CD,&37,&BD,&CD,&28,&BD,&CD,&00,&BB
530 DATA &CD,&4E,&BB,&CD,&BA,&BB,&CD,&FF,&BB,&CD,&65,&BC,&CD,&A7,&BC,&FB
540 DATA &3E,&01,&CD,&90,&BB,&C9
550 ' DATAs zum Verschieben des Programms -----
560 DATA &A001,&A004,&A00A,&A00D
570 END ' -----

```

Nach dem Programmstart und der Initialisierung der RSX-Erweiterung besitzen Sie einen neuen Befehl:

:DEFAULTS

Folgende Systemkomponenten werden von DEFAULTS auf die normalen Standardwerte gesetzt, die sie nach dem Einschalten des Computers besitzen:

- Die Tastaturverwaltung
- Der Textbildschirm
- Der Grafikbildschirm
- Das Screen-Paket
- Die Cassetten-Verwaltung
- Die Tongenerator-Verwaltung
- Der Druckervektor
- Die Tabelle der Sprungvektoren
- Eventuell angeschlossene Peripheriegeräte

Das Programm ist nicht allzu kompliziert und kann deshalb recht leicht verstanden werden. Es setzt sich zusammen aus den Definitionen, der RSX-Initialisierung und dem eigentlichen Hauptprogramm, das beim Label "DEFAULTS" beginnt.

"DEFAULTS" ist folgendermaßen aufgebaut:

- RESET\_PALETTE gibt den Wert &00 auf den Port &DF00. Dieser Befehl setzt den "Palette-Pointer" des Computers zurück.
- RESET\_PERIPH schickt den Wert &FF auf den Port &F800. Alle angeschlossenen Peripheriegeräte sollten, sofern sie richtig an die Schneider-Hardware angepaßt sind, sich angesprochen fühlen und dieses Signal auswerten. Daraufhin können die Geräte für sich selbst einen Reset durchführen.
- RESET\_VECTORS setzt alle Betriebssystem-Komponenten auf ihre Originalwerte:
  - \* JUMP RESTORE (&BD37) legt auf die Vektoren im Bereich ab &BB00 die Sprünge, die dort beim Systemstart standen. Diese Vektoren sind Eintrittsadressen für alle Routinen des Betriebssystems und können, da im RAM liegend, beliebig verän-

dert werden. JUMP RESTORE macht diese Änderungen wieder rückgängig.

Von dieser Möglichkeit, die Vektoren "umzupatchen" macht zum Beispiel das Floppy-Betriebssystem Gebrauch und ersetzt die Zeiger auf Cassettenroutinen durch solche auf deren Disketten-Äquivalente. Das bedeutet auch, daß Sie -falls Sie Floppy-Besitzer sind-, nach der RSX :DEFAULTS gleich :DISC eingeben sollten.

- \* MC RESET PRINTER (&BD28) trägt im Indirection-Vektor IND MC WAIT PRINTER (&BDF1) wieder den Zeiger auf die ROM-Routine ein.
- \* KM INITIALISE (&BB00) setzt die Tastaturverwaltung zurück. Im einzelnen unternimmt die Routine die folgenden Aktionen:

Die Indirection KM TEST BREAK wird auf den Standardeinsprung gesetzt.

Der Tastaturpuffer wird gelöscht.

Die Belegung der Funktionstasten wird gelöscht. Auf dem Zehnerblock finden Sie wieder die normalen Zahlentasten.

Die Tastaturübersetzungstabellen, die sich zum Beispiel mit dem Basic-Befehl KEY DEF manipulieren lassen, werden wieder auf ihre Normalwerte gesetzt.

Die Tabelle der Tasten mit Wiederholfunktion wird initialisiert.

CAPS LOCK und SHIFT LOCK werden gelöscht.

Die ESC-Taste wird blockiert. Daher ist eine Benutzung der RSX :DEFAULTS - von Spezialfällen einmal abgesehen - nur im Direktmodus zu empfehlen.

- \* TXT INITIALISE (&BB4E) setzt den Text-Bildschirm zurück:

Alle Text-Indirections werden wiederhergestellt.

Die Sprungtabelle der Controlcodes wird initialisiert.

Paper wird 0, die Pen-Farbe 1.

Das Window #0 erstreckt sich über den gesamten Bildschirm.

TAGOFF wird gewählt.

Der Cursor wird in der linken oberen Bildschirmcke positioniert.

- \* GRA INITIALISE (&BBBA) setzt den Grafikbildschirm zurück:

Alle Grafik-Indirections werden wiederhergestellt.

GRAFIK-PAPER ist 0, GRAFIK-PEN wird 1.

ORIGIN setzt das Betriebssystem auf die Koordinaten XPOS=0 und YPOS=0.

Das Grafikfenster erstreckt sich über den ganzen Bildschirm.

- \* SCR INITIALISE (&BBFF) setzt das Screenpack zurück und übernimmt die Aufgaben, die nicht schon von TXT INITIALISE und SCR INITIALISE durchgeführt wurden:

Die SCR-Indirections werden wiederhergestellt.

INK und SPEED INK werden normalisiert.

Der Bildschirm-MODE 1 wird gewählt.

Der Bildschirmspeicher startet bei &C0000.

Der Bildschirm wird gelöscht.

- \* CAS INITIALISE (&BC65) setzt die Cassettenverwaltung zurück:

Die Cassettendateien werden geschlossen.

SPEED WRITE wird auf den Wert 0 gesetzt.

- \* SOUND RESET (&BCA7) setzt die Verwaltung des Tonchips zurück:

Alle Sound-Warteschlangen werden gelöscht.

Der Tonchip AY-3-8912 wird initialisiert.

- RESET\_PEN setzt die Pen-Farbe auf Eins. Manchmal macht der Computer dies nämlich seltsamerweise nicht automatisch, obwohl TXT INITIALISE diese Aufgabe erledigen sollte.

Wenn Sie die RSX-Erweiterung !DEFAULTS benutzen wollen, kann es ganz nützlich sein, den Befehl auf eine Funktionstaste zu legen:

KEY 139,"!DEFAULTS"+CHR\$(13)

Wenn Sie später einmal die Enter-Taste im Zehnerblock der Tastatur drücken, leitet der Computer die Ausführung von !DEFAULTS ein; und zwar unabhängig von der SPEED KEY-Geschwindigkeit!

## 18. RSX-Befehle intern

Das Schneider-Basic läßt sich bekanntlich um sogenannte RSX-Befehle (RSX steht für "Resident System Extension") erweitern. Diese werden ähnlich den normalen Basic-Befehlen aufgerufen, der Benutzer muß sich also nicht mit den komplizierten und auch fehlerträchtigen CALL-Aufrufen herumplagen.

Doch wer im System oder in fremden Programmen forschen will, muß mit Unterstützung von Maschinensprache-Monitoren recht mühsam die Startadresse eines RSX-Moduls herausfinden. Erst dann kann man es disassemblieren, weil man erst ab diesem Zeitpunkt weiß, wo das Programm im Speicher liegt.

Erheblich einfacher wird das mit den Programmen RSXLIST und GETADDR.

RSXLIST gibt alle RSX-Namen eines spezifizierten Erweiterungs-ROMs aus, GETADDR die Startadresse und den ROM-Status einer beliebigen RSX, also sowohl im RAM- als auch im ROM-Speicher.

Beide Maschinenprogramme sind selbst RSXen und können entweder als Quellcode eingetippt und assembliert oder als Basic-Loader abgeschrieben werden.

Beginnen wir mit GETADDR: Diese RSX verlangt den Namen eines RSX-Befehls und liefert die Startadresse des zugehörigen Maschinenprogramms und den ROM-Select-Status.

### 18.1 Assemblerlisting: GETADDR

```
; *****
; *
; *          RSX-Erweiterung GETADDR
; *
; *          Meldet Startadresse und ROM-Select für einen RSX-Befehl
; *
; *
; *****

          ORG      &A000          ; Mit dem Lader verschiebbar
          JP       INIT_RSX      ; Sprung zur RSX-Initialisierung

;
; Definitionen und Deklarationen *****
; -----

KL_UROM_ENABLE EQU      &B900          ; KL U ROM ENABLE
KL_LOG_EXTERNAL EQU     &BCD1          ; KL LOG EXTERNAL
KL_FIND_COMMAND EQU     &BCD4          ; KL FIND COMMAND
BAS_ERR        EQU      &CA94          ; CPC-864 = &CB50, 6128 = &CB4D

;
; Einbindung der RSX-Erweiterung *****
; -----

INIT_RSX    LD      BC,JUMPTABLE      ; Zeiger auf Sprungtabelle
```

```

LD      HL,KERNEL_RAM    ; Zeiger auf Hilfsp Speicher
CALL    KL_LOG_EXTERNAL  ; RSX einbinden
RET                                           ; Rücksprung nach Basic

;
; Daten zur Verarbeitung der RSX *****
; -----

JUMPTABLE   DEFW    NAMETABLE      ; Vektor auf Namenstabelle
            JP      GETADDR        ; Sprung nach GETADDR

NAMETABLE   DEFW    "GETADD"       ; RSX-Name "GETADDR"
            DEFB    "R"+&80        ; Wortende
            DEFB    &00            ; Tabellenende

KERNEL_RAM  DEFS    &04            ; 4 Bytes zur Verwaltung

;
; GETADDR - das ist die Befehlsweiterung *****
; -----

GETADDR     EQU      $

CHECK_PARAMS CP      3              ; Drei Argumente aus Basic?
            JR      NZ,BASIC_ERROR ; Nein - Fehlermeldung

GET_RSXSTRING LD      H,(IX+5)      ; Descriptor lowbyte 1. Parameter
            LD      L,(IX+4)      ; Descriptor highbyte 1. Parameter
            LD      C,(HL)        ; Stringlänge lesen
            LD      B,0           ; BC=C, B wird gelöscht
            INC     HL            ; Zeiger auf Adressenangabe
            LD      E,(HL)        ; Lowbyte lesen
            INC     HL            ; Zeiger erhöhen
            LD      D,(HL)        ; Highbyte lesen
            EX      DE,HL         ; HL=DE
            PUSH    HL            ; HL-Register sichern
            ADD     HL,BC         ; Zeiger auf Stringende+1
            DEC     HL            ; Zeiger auf Stringende
            LD      A,(HL)        ; Dortiges Byte lesen
            OR      &80           ; 7. Bit setzen
            LD      (HL),A        ; Wieder abspeichern
            POP     HL            ; HL-Register wiederherstellen

SEARCH_RSX  CALL     KL_FIND_COMMAND ; RSX-Kommando suchen
            JR      C,ASSIGN_ADDR  ; Carry gesetzt - Gefunden!

NOT_FOUND   LD      HL,0          ; Sonst Adresse=0
            LD      C,0           ; " ROM-Status=0

ASSIGN_ADDR EX      DE,HL         ; DE und HL tauschen
            LD      H,(IX+3)      ; Adressen-Descriptor (Low)
            LD      L,(IX+2)      ; Adressen-Descriptor (High)
            LD      (HL),E        ; Dort RSX-Adresse (Low) laden
            INC     HL            ; Zeiger erhöhen
            LD      (HL),D        ; RSX-Adresse (High) laden

ASSIGN_ROMSTAT LD     H,(IX+1)    ; ROM-Status-Descriptor (Low)

```

```

LD      L,(IX+0)      ; ROM-Status-Descriptor (High)
LD      (HL),C        ; ROM-Status abspeichern
INC     HL            ; Zeiger erhöhen
LD      (HL),0        ; Highbyte auf Null setzen
RET                                ; Rücksprung nach Basic

;
; BASIC_ERROR gibt eine Basic-Fehlermeldung aus *****
; -----

BASIC_ERROR  CALL      KL_UROM_ENABLE  ; Basic-ROM einschalten
              LD        E,22          ; "Operand Missing"
              JP        BAS_ERR       ; Sprung zur Fehlerausgabe

              END

; -----

```

Das zugehörige Basic-Programm, das das Maschinencode-Programm in DATA-Zeilen enthält, folgt hier:

## 18.2 Listing: GETADDR.BAS

```

100 ' *****
110 ' *****
120 ' *
130 ' *          GETADDR          *
140 ' *
150 ' *****
160 ' *****
170 '
180 DEF FNmsb(a)=255 AND INT(a/256)
190 DEF FNlsb(a)=255 AND UNT(a)
200 SYMBOL AFTER 254
210 SYMBOL 255,&66,&0,&66,&66,&66,&3E,&0
220 ' Titelbild und Informationen -----
230 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
240 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
250 LOCATE 1,1:PRINT STRING$(40,210);
260 LOCATE 1,2
270 PRINT CHR$(24)+SPACE$(16)+"GETADDR"+SPACE$(17)+CHR$(24)
280 LOCATE 10,5:PRINT CHR$(164);" Martin Kotulla 1986"
290 PRINT:PRINT STRING$(40,210):PRINT
300 PEN 1:PRINT:PRINT " GETADDR liefert f"+CHR$(255)+"r alle RSX-Befehle"
310 PRINT:PRINT " im RAM oder ROM deren Startadresse und"
320 PRINT:PRINT " ROM-Status."
330 PRINT:PEN 3:PRINT STRING$(40,210):PRINT
340 PEN 2:PRINT TAB(5);"addr%:rom%:rsx%="+CHR$(34)+"RSXNAME"+CHR$(34)
350 PRINT:PRINT TAB(5);"GETADDR,@rsx%,@addr%,@rom%"
355 ' DATA-Zeilen prüfen -----
360 FOR i=1 TO 101:READ a:sum=sum+a:NEXT i
370 IF sum=-113472 THEN 400
380 PRINT CHR$(7);" * Fehler in den DATA-Zeilen!"
390 LIST 650-:END
395 ' Maschinenprogramm einPOKEN -----
400 RESTORE:MEMORY HIMEM-100:start=HIMEM+1

```

```

410 FOR i=start TO start+95:READ a:POKE i,a:NEXT i
415 ' An die Ladeadresse anpassen -----
420 FOR i=1 TO 5:READ a:a=a-40960+start
430 value=PEEK(a)+PEEK(a+1)*256-40960+start
440 POKE a,FNlsb(value):POKE a+1,FNmsb(value)
450 NEXT i
460 CALL start ' RSX-Befehl installieren
470 ' Computertyp feststellen -----
480 version$=CHR$(&CD)+CHR$(&0)+CHR$(&B9)+CHR$(&3A)+CHR$(&2)+CHR$(&C0)
490 version$=version$+CHR$(&32)+CHR$(&66)+CHR$(&1)+CHR$(&C9)
500 CALL PEEK(@version+1)+PEEK(@version+2)*256 ' Welcher Computertyp?
510 version=PEEK(&166)
520 IF version=0 THEN 560
530 POKE start+&5E,&CB
540 IF version=1 THEN POKE start+&5D,&50 ELSE POKE start+&5D,&4D
550 ' Demoprogramm -----
560 FOR i=1 TO 6000:NEXT i
570 MODE 1:PRINT:PRINT TAB(8);" D E M O N S T R A T I O N":PRINT
580 PRINT " ";STRING$(38,"-"):PRINT:PRINT
590 addr%=0:rom%=0:rsx$="GETADDR"+"
600 !GETADDR,@rsx$,@addr%,@rom%
610 PRINT " Die RSX !GETADDR startet bei &";HEX$(addr%);".
620 PRINT:PRINT " Der ROM-Status ist &";HEX$(rom%,2);".
630 PRINT:PRINT:PEN 1:END
640 ' DATAs fuer das Maschinenprogramm -----
650 DATA &C3,&03,&A0,&01,&0D,&A0,&21,&1A,&A0,&CD,&D1,&BC,&C9,&12,&A0,&C3
660 DATA &1E,&A0,&47,&45,&54,&41,&44,&44,&D2,&00,&00,&00,&00,&FE,&03
670 DATA &20,&35,&DD,&66,&05,&DD,&6E,&04,&4E,&06,&00,&23,&5E,&23,&56,&EB
680 DATA &E5,&09,&2B,&7E,&F6,&08,&77,&E1,&CD,&D4,&BC,&38,&05,&21,&00,&00
690 DATA &0E,&00,&EB,&DD,&66,&03,&DD,&6E,&02,&73,&23,&72,&DD,&66,&01,&DD
700 DATA &6E,&00,&71,&23,&36,&00,&C9,&CD,&00,&B9,&1E,&16,&C3,&94,&CA,&00c
710 ' DATAs zum Verschieben des Programms -----
720 DATA &A001,&A004,&A007,&A00D,&A010
730 END ' #####

```

Die Parameterübergabe und -Übernahme erfolgt beim RSX-Befehl GET-ADDR über den Variablenpointer von Basic-Variablen.

Falls Sie es noch nicht wissen: Mit a\$="TEXT":PRINT @a\$ erhalten Sie statt der Zeichenkette einen Zeiger auf den String-Descriptor.

Dieser Befehl steht nicht im Benutzerhandbuch des Computers!

Der Aufruf von GETADDR kann etwa so erfolgen:

```

a$="USER"
adr%=0
rom%=0
!GETADDR,@a$,@adr%,@rom%
PRINT adr%;rom%

```

In der Variablen adr% befindet sich nach dem Abarbeitung der RSX die Startadresse des RSX-Programms, in rom% der RAM/ROM-Status. Wenn der Computer keine passende RSX-Erweiterung finden konnte,

Ist `adr%=0` und `rom%=0`. Denken Sie aber daran, immer nur Integervariablen anzugeben! Diese können entweder durch ein Prozentzeichen im Variablennamen oder durch `DEFINT` vereinbart werden:

```
a%=0
DEFINT b:beispielvariable=&0030
```

Auch muß die Variable bereits vor dem Aufruf von `GETADDR` schon einmal benutzt worden sein. Sonst erhalten Sie die Fehlermeldung "Improper Argument". Definieren Sie also zum Beispiel:

```
adr%=0
rom%=ASC("Irgendein Text")
```

Das Ende eines RSX-Namens wird intern vom Computer durch ein gesetztes siebtes Bit im letzten Buchstaben angezeigt.

Von Assemblern wird das etwa so verarbeitet:

```
DEFM "BASI"
DEFB "C"+128
```

Sie müssen aber nicht unbedingt das siebte Bit setzen. `GETADDR` nimmt Ihnen diese Arbeit ab. Es verändert den String automatisch so, daß das siebte Bit auf 1 gesetzt ist:

```
a$="DIR":a%=1:b%=1
PRINT ASC(RIGHT$(a$,1))   liefert den Wert 82
:GETADDR,@a$,@a%,@b%
PRINT ASC(RIGHT$(a$,1))   liefert 82+128=210
```

Wenn Sie den String in einem Programm direkt angeben, kann es passieren, daß `GETADDR` den String unmittelbar im Programmcode ändert. Dies können Sie durch eine einfache Stringmanipulation verhindern:

```
Statt      10 a$="CPM ROM"
Besser     10 a$="CPM ROM"+" "
```

Eine Ausnahme gilt es bei der Benutzung von `GETADDR` zu beachten: Die RSX "BASIC", die sich bereits im eingebauten ROM auf der Hauptplatine befindet, befindet sich in einem sogenannten "Vordergrund-ROM" und zwingt den Computer, die Initialisierungs-Routine des Basic-ROMs aufzurufen. `GETADDR` kehrt deswegen bei der RSX "BASIC" nicht zum Programm zurück. Damit sind aber alle Daten und Programme gelöscht - also Vorsicht!

Vergessen Sie einen Parameter oder geben Sie mehr als die gestatteten drei Argumente an, meldet das Programm auf dem CPC-464

"Operand missing",

auf dem CPC-664 und CPC-6128 hingegen

"Improper Argument".



Wenn Sie sich neben der reinen Anwendung auch für die Funktionsweise von GETADDR interessieren, können Sie hier weiterlesen:

Das Hauptprogramm finden Sie ab dem Label GETADDR. Es prüft zuerst, ob Sie wirklich drei Argumente aus dem Basic heraus übergeben.

Trifft dies nicht zu, springt der Computer zur Marke BASIC\_ERROR und gibt eine passende Fehlermeldung aus.

GET\_RSXSTRING holt den String, in dem der RSX-Name steht, ins Programm:

```
LD  H,(IX+5)    bringt einen Zeiger auf den String-Descriptor
LD  L,(IX+4)    ins HL-Register.
```

Dieser Descriptor ist immer gleich aufgebaut:

1. Byte: Stringlänge
2. Byte: Lowbyte der Startadresse des Strings
3. Byte: Highbyte der Startadresse

```
LD  C,(HL)      setzt das BC-Register
LD  B,0         auf die Stringlänge.
```

```
INC  HL         lädt das DE-Register mit einem Zeiger
LD   E,(HL)     auf den tatsächlichen String,
INC  HL         EX DE,HL sorgt dafür, daß dieser Pointer
LD   D,(HL)     ins HL-Register gebracht wird.
```

```
Der Programmcode  PUSH HL
                  ADD  HL,BC
                  DEC  HL
                  LD   A,(HL)
                  OR   &80
                  LD   (HL),A
                  POP  HL
```

dient dazu, das Highbit im letzten Zeichen des RSX-Namens zu setzen. Das zeigt - wie bereits gesagt - dem Computer das Ende des Namens an.

SEARCH\_RSX verwendet die Betriebssystem-Routine KL FIND COMMAND, um die Startadresse des RSX-Programms herauszufinden.

KL FIND COMMAND erwartet im HL-Register einen Zeiger auf die Zeichenkette, die den RSX-Namen repräsentiert. Nach deren Abarbeitung ist das Carry-Flag gesetzt oder gelöscht - je nachdem, ob die Suche erfolgreich war: Wenn die RSX nicht gefunden wurde, ist das Carry-Flag gelöscht, ansonsten ist es gesetzt.

Das Carry-Flag wird mit JR C,ASSIGN\_ADDR auch von GETADDR ausgewertet. Wenn es gelöscht ist, wird auch die Adresse und der ROM-Status auf Null gesetzt (LD HL,0:LD C,0). Andernfalls springt das

Programm zum Label ASSIGN\_ADDR: EX DE,HL bringt die Adresse der Routine ins DE-Register, die Zeilen

```
LD    H,(IX+3)
LD    L,(IX+2)
LD    (HL),E
INC   HL
LD    (HL),D
```

besorgen die Übergabe des Parameters an die Basic-Variable.

ASSIGN\_ROMSTAT weist mit den Zeilen

```
LD    H,(IX+1)
LD    L,(IX+0)
LD    (HL),C
INC   HL
```

der dritten beim RSX-Aufruf angegebenen Variablen den Wert des C-Registers zu. Da der ROM-Status nur Werte zwischen 0 und 255 annehmen kann, muß das High-Byte der Variablen ausdrücklich gelöscht werden:

```
LD (HL),0
```

Dieser ROM-Status repräsentiert eines der 252 anschließbaren Erweiterungs-ROMs. Bisher vergebene Nummern sind zum Beispiel 7 für das Floppy-ROM und 5 für den MAXAM-Assembler in der ROM-Version. Mit den Werten 252 bis 255 hat es eine besondere Bewandnis: Sie werden nur für im RAM-Arbeitsspeicher stehende RSX-Befehle gemeldet und geben die RAM-ROM-Auswahl an:

```
rom%=252: Basic-ROM ab &C000 eingeblendet
          Betriebssystem-ROM ab &0000 eingeblendet

rom%=253: Basic-ROM ab &C000 eingeblendet
          RAM ab &0000 eingeblendet

rom%=254: Video-RAM ab &C000 eingeblendet
          Betriebssystem-ROM ab &0000 eingeblendet

rom%=255: Video-RAM ab &C000 eingeblendet
          RAM ab &0000 eingeblendet
```

Ganz interessant mag es auch für Programmierer sein, wie sich von Maschinenprogrammen aus Basic-Fehlermeldungen erzeugen lassen: Mit CALL KL U ROM ENABLE (&B900) muß dazu erst einmal das Basic-ROM in den Adreßbereich zwischen &C000 und &FFFF eingeschaltet werden. Als nächstes ist das E-Register mit dem Fehlercode zu laden. Dieser Code ist identisch mit dem vom Basic-Befehl ERROR erzeugten. Einige der auch für Maschinenprogramme sinnvollen Meldungen lauten zum Beispiel:

```
2 - Syntax Error
5 - Improper Argument
6 - Overflow
13 - Type Mismatch
```

```

22 - Operand Missing
24 - EOF Met
25 - File Type Error
27 - File Already Open

```

Beim nun folgenden Aufruf der Routine, die den Fehler ausgibt, scheiden sich die (Computer-) Geister: Jedes Modell verlangt eine andere Adresse und erwartet den Fehlercode in einem anderen Register!

Beim CPC-464 liegt die Startadresse bei &CA94, der Fehlercode wird im E-Register übergeben. Der CPC-6128 verlangt hingegen den Code im Akku und will als Startadresse &CB55 sehen. Alternativ können Sie auch - wie bei GETADDR geschehen - &CB50 (CPC-664) oder &CB4D (CPC-6128) aufrufen. Dann meldet der Computer aber immer nur "Improper Argument".

Testen können Sie die Möglichkeiten zur Fehlerausgabe auf dem CPC-464 mit einem kleinen Maschinenprogramm:

```

&A000 &CD &00 &B9      CALL &B900
&A003 &3A &60 &01      LD  A,(&0160)
&A006 &5F                LD  E,A
&A007 &C3 &94 &CA      JP   &CA94

```

Mit POKE &160,FEHLERCODE und CALL &A000 wird der gewünschte Fehler ausgegeben.

Auf dem CPC-6128 muß das etwas anders aussehen:

```

&A000 &CD &00 &B9      CALL &B900
&A003 &3A &60 &01      LD  A,(&0160)
&A006 &C3 &55 &CB      JP   &CB55

```

Der Aufruf ist dennoch identisch:

```
POKE &160,FEHLERCODE und CALL &A000.
```

Jetzt aber zum zweiten Programm, "RSXLIST". Dieses hat die Aufgabe, sämtliche RSX-Befehle eines Erweiterungs-ROMs zusammen mit ihrer hexadezimalen Startadresse auf dem Bildschirm anzuzeigen.

### 18.3 Assemblerlisting: RSXLIST

```

; *****
; *
; *          RSX-Erweiterung RSXLIST
; *
; *      Druckt sämtliche RSX-Kommandos eines Erweiterungs-ROMs aus
; *
; *****

ORG      &A000          ; Mit dem Lader verschiebbar
JP       INIT_RSX      ; Sprung zur RSX-Initialisierung

```

```

;
; Definitionen und Deklarationen *****
; -----

ROMSTATUS      DEFS      2           ; Speicher für ROM-Status
KL_UROM_ENABLE EQU      &B900       ; KL U ROM ENABLE
KL_ROM_SELECT  EQU      &B90F       ; KL ROM SELECT
KL_ROM_DESELECT EQU      &B918       ; KL ROM DESELECT
KM_WAIT_KEY    EQU      &BB18       ; KM WAIT KEY
KM_READ_KEY    EQU      &BB1B       ; KM READ KEY
TXT_OUTPUT     EQU      &BB5A       ; TXT OUTPUT
TXT_WR_CHAR    EQU      &BB5D       ; TXT WR CHAR
TXT_SET_COLUMN EQU      &BB6F       ; TXT SET COLUMN
KL_LOG_EXTERNAL EQU      &BCD1       ; KL LOG EXTERNAL
BAS_ERR        EQU      &CA94       ; CPC-664 = &CB50, 6128 = &CB4D

CRLFSPACES     DEFB      13,10,32,32,32,32,0
;
; Einbindung der RSX-Erweiterung *****
; -----

INIT_RSX        LD        BC,JUMPTABLE ; Zeiger auf Sprungtabelle
                LD        HL,KERNEL_RAM ; Zeiger auf Hilfsppeicher
                CALL      KL_LOG_EXTERNAL ; RSX einbinden
                RET        ; Rücksprung nach Basic

JUMPTABLE       DEFW      NAMETABLE   ; Vektor auf Namenstabelle
                JP        RSXLIST     ; Sprung nach RSXLIST

NAMETABLE        DEFM      "RSXLIST"   ; RSX-Name "RSXLIST"
                DEFB      "T"+&80      ; Wortende
                DEFB      &00          ; Tabellenende
KERNEL_RAM       DEFS      &04        ; 4 Bytes zur Verwaltung
; RSXLIST - die eigentliche Befehlserweiterung *****
; -----

RSXLIST          EQU      $

CHECK_PARAMS     CP        1           ; Ein Argument aus Basic?
                JR        NZ,BASIC_ERROR ; Nein - Fehlermeldung

GET_ROM_NUMBER   LD        C,(IX+0)    ; ROM-Nummer aus Basic
                CALL      KL_ROM_SELECT ; ROM einblenden
                LD        (ROMSTATUS),BC ; Alten ROM-Status merken

SHOW_RSXNAMES    CALL      CR_LF       ; CR & LF ausgeben
                LD        HL,(&C004)    ; Start der Namenstabelle
                LD        DE,&C006      ; Start der Sprungtabelle

RSX_SHOW_LOOP    LD        A,(HL)      ; Namensbyte lesen
                CP        0            ; &00 = Tabellenende
                JR        Z,LAST_RSX    ; Wenn Ende erreicht
                CP        128          ; 7.Bit gesetzt?
                PUSH      AF           ; Zero-Flag sichern
                AND        &7F         ; 7.Bit löschen
                PUSH      HL           ; Namenszeiger sichern

```

```

PUSH    DE                ; Sprungzeiger sichern
CALL    TXT_WR_CHAR       ; Zeichen ausdrucken
POP     DE                ; DE wiederherstellen
POP     HL                ; HL wiederherstellen
INC     HL                ; Namenszeiger erhöhen
POP     AF                ; Flags wiederherstellen
JR      C,RSX_SHOW_LOOP   ; Carry=0? Weitermachen!

PUSH    HL                ; HL-Register sichern
LD      A,23              ; PRINT TAB(23);
CALL    TXT_SET_COLUMN    ; Cursor setzen

LD      A,"&"             ; Hexadezimal-"&"
CALL    TXT_OUTPUT        ; Ausgeben

INC     DE                ; Sprungzeiger +1
INC     DE                ; Sprungzeiger +1
EX      DE,HL             ; HL=DE
LD      C,(HL)            ; Sprung-Highbyte lesen
CALL    HEX_OUTPUT        ; Hexadezimal ausgeben
DEC     HL                ; Sprungzeiger -1
LD      C,(HL)            ; Sprung-Lowbyte lesen
CALL    HEX_OUTPUT        ; Hexadezimal ausgeben
INC     HL                ; Sprungzeiger +1
INC     HL                ; Sprungzeiger +1
EX      DE,HL             ; DE=HL
CALL    CR_LF             ; CR, LF und Spaces
POP     HL                ; HL wiederherstellen

KEY_PRESSED CALL    KM_READ_KEY   ; Taste gedrückt?
JR      NC,RSX_SHOW_LOOP ; Nein - weitermachen
CALL    KM_WAIT_KEY       ; Ja - auf Taste warten
JR      RSX_SHOW_LOOP     ; Wieder in die Schleife

LAST_RSX LD      BC,(ROMSTATUS) ; ROM-Status laden
CALL    KL_ROM_DESELECT   ; Wiederherstellen
RET                                ; Zurück nach Basic

;
; CR_LF druckt Wagenrücklauf, Zeilenvorschub und vier Spaces aus *****
; -----

CR_LF    LD      HL,CRLFSPACES ; Zeiger auf String

;
; PRINTSTRING druckt Speicher ab (HL) bis (HL)=0 *****
; -----

PRINTSTRING PUSH    AF                ; Akku und Flags sichern

PRINTSTRLOOP LD      A,(HL)           ; Zeichen lesen
CP            0                      ; Ende erreicht?
JR      Z,PRINTRETURN ; Ja - zurück
CALL    TXT_OUTPUT    ; Zeichenausgabe
INC     HL            ; Zeiger erhöhen
JR      PRINTSTRLOOP  ; In die Schleife

```

```

PRINTRETURN    POP      AF          ; AF wiederherstellen
                RET          ; Rücksprung nach Basic

;
; BASIC_ERROR gibt eine Basic-Fehlermeldung aus *****
; -----
BASIC_ERROR     CALL     KL_UROM_ENABLE ; Basic-ROM einschalten
                LD       E,22          ; "Operand Missing"
                JP       BAS_ERR       ; Sprung zur Fehlerausgabe

;
; HEX_OUTPUT druckt die Hexziffer im C-Register aus *****
; -----
HEX_OUTPUT      LD       A,C          ; Hexziffer in den Akku
                CALL     RIGHTSHIFT    ; Rechts-Shift & Ausgabe
                LD       A,C          ; Hexziffer in den Akku
                CALL     SHOWHEX       ; Ausgabe des Lownibbles
                RET          ; Rücksprung zum Programm

RIGHTSHIFT      RRA                  ; Den Akkumulator viermal
                RRA                  ; nach rechts um jeweils
                RRA                  ; eine Bitposition shiften
                RRA                  ; Highnibble wird Lownibble

SHOWHEX         AND      15          ; Linke 4 Bits ausblenden
                ADD      "0"         ; Aus Binär mach' ASCII
                CP       "9"+1       ; Größer als 9?
                JR       C,DISPLAY    ; Nein - ausgeben
                ADD      7            ; Sonst als Buchstaben

DISPLAY         CALL     TXT_OUTPUT   ; Bildschirmausgabe
                RET          ; Rücksprung

                END

; -----

```

Den zugehörigen Basic-Loader wollen wir auch nicht vergessen.

#### 18.4 Listing: RSXLIST.BAS

```

100 ' *****
110 ' *****
120 ' *
130 ' *          RSXLIST          *
140 ' *
150 ' *****
160 ' *****
170 '
180 DEF FNmsb(a)=255 AND INT(a/256)
190 DEF FNlsb(a)=255 AND UNT(a)
200 ' Titelbild und Informationen -----
210 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1

```

```

220 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
230 LOCATE 1,1:PRINT STRING$(40,210);
240 LOCATE 1,2
250 PRINT CHR$(24)+SPACE$(16)+"RSXLIST"+SPACE$(17)+CHR$(24)
260 LOCATE 10,5:PRINT CHR$(164);" Martin Kotulla 1986"
270 PRINT:PRINT STRING$(40,210):PRINT
280 PEN 1:PRINT:PRINT " Diese RSX-Befehlserweiterung bietet"
290 PRINT:PRINT " ein neues RSX-Kommando, mit dem sich"
300 PRINT:PRINT " alle Befehle eines bestimmten Erwei-"
310 PRINT:PRINT " terungs-ROMs auflisten lassen."
320 PRINT:PEN 3:PRINT STRING$(40,210):PRINT
330 PEN 2:PRINT TAB(13);"RSXLIST,romnummer":PEN 1
335 ' Pruefsumme der DATA-Zeilen erstellen -----
340 FOR i=1 TO 196:READ a:sum=sum+a:NEXT i
350 IF sum=-296873 THEN 380
360 PRINT CHR$(7);" * Fehler in den DATA-Zeilen!"
370 LIST 350-:END
375 ' Maschinenprogramm einPOKEn -----
380 RESTORE:MEMORY HIMEM-190:start=HIMEM+1
390 FOR i=start TO start+182:READ a:POKE i,a:NEXT i
400 FOR i=1 TO 14:READ a:a=a-40960+start
410 value=PEEK(a)+PEEK(a+1)*256-40960+start
420 POKE a,FNlsb(value):POKE a+1,FNmsb(value)
430 NEXT i
440 CALL start
450 ' Computertyp feststellen -----
460 version$=CHR$(&CD)+CHR$(&0)+CHR$(&B9)+CHR$(&3A)+CHR$(2)+CHR$(&C0)
470 version$=version$+CHR$(&32)+CHR$(&66)+CHR$(1)+CHR$(&C9)
480 CALL PEEK(@version+1)+PEEK(@version+2)*256 ' Welcher Computertyp?
490 version=PEEK(&166)
500 IF version=0 THEN 540
510 POKE start+&9B,&CB
520 IF version=1 THEN POKE start+&9A,&50 ELSE POKE start+&9A,&4D
530 ' Demoprogramm -----
540 ' (Nur einfüegen, wenn Floppy-Controller vorhanden!)
550 MODE 1:RSXLIST,7:END
560 ' DATAs fuer das Maschinenprogramm -----
570 DATA &C3,&0C,&A0,&00,&00,&0D,&0A,&20,&20,&20,&20,&00,&01,&16,&A0,&21
580 DATA &23,&A0,&CD,&D1,&BC,&C9,&1B,&A0,&C3,&27,&A0,&52,&53,&58,&4C,&49
590 DATA &53,&D4,&00,&00,&00,&00,&00,&FE,&01,&20,&69,&DD,&4E,&00,&CD,&0F
600 DATA &B9,&ED,&43,&03,&A0,&CD,&83,&A0,&2A,&04,&C0,&11,&06,&C0,&7E,&FE
610 DATA &00,&28,&30,&FE,&80,&F5,&E6,&7F,&E5,&D5,&CD,&5D,&BB,&D1,&E1,&23
620 DATA &F1,&30,&EB,&E5,&3E,&17,&CD,&6F,&BB,&3E,&26,&CD,&5A,&BB,&13,&13
630 DATA &EB,&4E,&CD,&9C,&A0,&2B,&4E,&CD,&9C,&A0,&23,&23,&EB,&CD,&63,&A0
640 DATA &E1,&CD,&1B,&BB,&30,&C8,&CD,&10,&BB,&18,&C3,&ED,&4B,&03,&A0,&CD
650 DATA &18,&B9,&C9,&21,&05,&A0,&F5,&7E,&FE,&00,&28,&06,&CD,&5A,&BB,&23
660 DATA &18,&F5,&F1,&C9,&CD,&00,&BB,&1E,&16,&C3,&94,&CA,&79,&CD,&A5,&A0
670 DATA &79,&CD,&A9,&A0,&C9,&1F,&1F,&1F,&1F,&E6,&0F,&C6,&30,&FE,&3A,&30
680 DATA &02,&C6,&07,&CD,&5A,&BB,&C9
690 ' DATAs zum Verschieben des Programms -----
700 DATA &A001,&A00D,&A010,&A016,&A019,&A033,&A036,&A063,&A068,&A06E
710 DATA &A07D,&A0B4,&A09E,&A0A2
720 END ' *****

```

Die Befehlssyntax ist erheblich einfacher als die des GETADDR-Kommandos:

```
!RSXLIST,romnummer
```

Zum Beispiel:

```
:RSXLIST,7    für das AMSDOS-Floppy-ROM
!RSXLIST,5    für das MAXAM-ROM
```

Die Bildschirmausgabe für die RSX-Liste des AMSDOS-ROMs wird in etwa so aussehen:

```
CPM ROM      &C1BC
CPM          &C1B2
DISC         &CCD1
DISC.IN      &CCD5
DISC.OUT     &CCE4
TAPE         &CCFD
TAPE.IN      &CD01
TAPE.OUT     &CD1B
A            &CDDA
B            &CDDD
DRIVE        &CDE4
USER         &CDFE
DIR          &D42E
ERA          &D48A
REN          &D4C4

             &CA72
             &C60D
             &C581
             &C666
             &C64E
             &C652
             &C763
             &C630
             &C603
```

RSXLIST gibt also nicht die Speicheradresse des Eintrags in der Sprungtabelle aus - diese kann man ja ganz leicht errechnen -, sondern das Ziel dieses Sprungs. An dieser Zieladresse beginnt die tatsächliche zugehörige Maschinenroutine.

Der erste RSX-Name, "CPM ROM" ist von den Computerentwicklern bei Amstrad nicht zur allgemeinen Verwendung gedacht worden, er wird nur beim Systemstart als Initialisierungsroutine gestartet.

Die letzten neun ausgedruckten RSX-Namen sehen aus wie Bildschirmsteuerzeichen: ihre Namen heißen CHR\$(1), CHR\$(2), CHR\$(3) bis CHR\$(9). Aus Basic lassen sie sich nicht aufrufen, wohl aber von Maschinenprogrammen. Sie erledigen niedere Aufgaben des Floppy-Betriebssystems, zum Beispiel das Lesen, Schreiben und Formatieren von Sektoren. Im einzelnen führen sie folgende Aufgaben durch:



### RSX 1 - Meldungen des Floppy-Systems an- und ausschalten

Meldungen wie "Disc missing - Retry, ignore, Cancel" lassen sich ausschalten, wenn der Akku den Wert 255 enthält. Ist der Akku gleich Null, werden die Meldungen wieder eingeschaltet.

### RSX 2 - Laufwerks-Parameter festlegen

Mit diesem RSX-Befehl lassen sich bestimmte Daten, die die Kommunikation mit der Floppy-Hardware regeln, festlegen. Im HL-Register muß dazu ein Zeiger auf eine Tabelle stehen. Diese Tabelle hat folgendes Aussehen:

- 1. und 2. Byte: Anlaufzeit des Floppy-Motors, bevor Daten übermittelt werden.  
Normalwert: &0032
- 3. und 4. Byte: Nachlaufzeit des Floppy-Motors, nachdem Daten übermittelt wurden.  
Normalwert: &00FA
- 5. Byte: Beim Formatieren als Warteschleife benötigter Wert. Hier steht nach dem Systemstart das Byte &AF. Eine Änderung ist zwar möglich, erscheint aber nicht sehr sinnvoll.
- 6. und 7. Byte: Spurwechselzeit, das ist die Zeit, die der Computer dem Laufwerk geben muß, damit es die Spur (Track) wechseln kann. Normalwert: &0C0F
- 8. Byte: Head Unload Time  
Zeit, die gewartet wird, bis der Schreib/Lese-Kopf von der Diskette abgehoben wird. Normalwert: &01
- 9. Byte: Head Load Time  
Zeit, die der Floppycontroller FDC765 wartet, bis er nach der Übermittlung des Befehls, den Schreib/Lese-Kopf aufzusetzen, mit der Datenübertragung beginnt. Normalwert: &03

### RSX 3 - Diskettenformat-Parameter festlegen

Diese RSX benötigt im Akkumulator des Z80-Prozessors den Code für das Diskettenformat, das beim folgenden Zugriff auf die Diskette angenommen werden soll. Für das CP/M-Systemformat ist das &40, für das Data-Only-Format &C0 und für das IBM-Format &00. Da das Vendor-Format eine Abart des CP/M-Systemformats ist (die System- und Boot-Spuren sind nicht gespeichert), gilt auch für dieses Format der Code &40. Damit können die später beschriebenen Routinen zum Lesen und Schreiben von Sektoren mit allen drei Diskettenformaten arbeiten.

## RSX 4 - Sektor lesen

Diese RSX liest den spezifizierten Sektor der Diskette mit einer Länge von 512 Bytes in den Arbeitsspeicher. Dazu müssen einige Register mit Werten geladen werden:

- Das E-Register mit der Laufwerksnummer (A=0, B=1)
- Das D-Register mit der Spurnummer (0 bis 39)
- Das C-Register mit der codierten Sektornummer (&01 bis &0B, &41 bis &49 oder &C1 bis &C9)
- Das HL-Register mit der Adresse, ab der der Sektor im Speicher abgelegt werden soll.

## RSX 5 - Sektor schreiben

Die RSX 5 schreibt einen Sektor aus dem RAM-Arbeitsspeicher auf die Diskette. Die Registerbelegung ist ähnlich der der RSX 4:

- Im E-Register die Laufwerksnummer
- Im D-Register die Tracknummer (Spurnummer)
- Im C-Register die Sektornummer
- Im HL-Register einen Zeiger auf den 512 Bytes großen Speicherbereich, der gesichert werden soll.

## RSX 6 - Spur formatieren

Diese RSX formatiert eine komplette aus acht (IBM-Format) beziehungsweise neun (Systemformat und Data-Only) Sektoren bestehende Spur. Dazu steht im E-Register die Laufwerksnummer, eben 0 für das Laufwerk A und 1 für das Laufwerk B, im C-Register die Nummer des ersten Sektors der Spur, im D-Register die Spurnummer und im HL-Register ein Zeiger auf die Formatierungstabelle.

Diese Formatierungstabelle ist 36 Bytes lang - für jeden Sektor sind vier Bytes reserviert, nämlich die Spurnummer, die Kopfnummer, die Sektornummer und die codierte Sektorgröße. Um beispielsweise den zehnten Track als Data-Only zu formatieren, sollte die Tabelle so aussehen:

```
DEFB 10,0,&C1,2
DEFB 10,0,&C3,2
DEFB 10,0,&C5,2
DEFB 10,0,&C7,2
DEFB 10,0,&C9,2
DEFB 10,0,&C2,2
DEFB 10,0,&C4,2
DEFB 10,0,&C6,2
DEFB 10,0,&C8,2
```

Die "2" als Sektorlänge kommt vom Highbyte der Sektorlänge: 512 Bytes sind &0200 Bytes. Das Highbyte von &0200 ist eben 2.

### RSX 7 - Spur aufsuchen

Eigentlich ist dieser RSX-Befehl recht unnütz, da alle Diskettenoperationen von sich aus die passende Spur aufsuchen. Aber bitte: Im E-Register steht der Laufwerkscode, im D-Register die Spurnummer.

### RSX 8 - Laufwerk abfragen

Hier läßt sich feststellen, ob ein bestimmtes Diskettenlaufwerk (A: oder B:) angeschlossen ist. Im Akkumulator muß dazu der Laufwerkscode (A=0, B=1) stehen. Wenn nach Aufruf der Routine der Akku für einen Laufwerkstest für A: gleich 0 beziehungsweise für B: gleich 1 ist, wurde das Laufwerk gefunden. Alle anderen Werte deuten auf einen Fehler hin.

### RSX 9 - Zahl der Schreib/Lese-Versuche festlegen

Wenn sich ein Sektor nicht lesen oder beschreiben läßt, zum Beispiel, weil er von einem dejustierten Laufwerk geschrieben oder formatiert wurde, versucht das Floppy-Betriebssystem insgesamt zehn Mal, auf den Sektor zuzugreifen. Erst dann gibt es eine Fehlermeldung aus. Wenn Sie aber mehr oder weniger Leseversuche wünschen, können Sie der RSX 9 im Akku den entsprechenden Wert übergeben.

RSXLIST ist auch von der Programmierung her recht interessant und lehrreich:

CHECK\_PARAMS prüft, ob auch wirklich genau ein Parameter aus Basic übergeben wird, und gibt nötigenfalls eine Fehlermeldung aus.

GET\_ROM\_NUMBER holt sich den Parameter aus Basic ins C-Register, blendet das passende ROM ein und speichert den alten ROM-Status, der am Programmende zur Wiederherstellung der RAM/ROM-Konfiguration gebraucht wird.

SHOW\_RSXNAMES gibt einen Wagenrücklauf und einen Zeilenvorschub an den Bildschirm aus und holt sich aus der Adresse &C004 einen Zeiger auf den Beginn der Namenstabelle. Das DE-Register wird mit einem Zeiger auf den Beginn der Sprungtabelle geladen.

RSX\_SHOW\_LOOP ist die Hauptschleife des Programms. LD A, (HL) holt ein Zeichen des Namens in den Akku, CP 0 und JR Z, LAST\_RSX brechen die Ausgabe der Namen ab, wenn das Tabellenende erreicht ist. CP 128 prüft, ob das siebte Bit des Zeichens gesetzt ist. Dies deutet darauf hin, daß das Ende eines Namens erreicht wurde. Der Computer reagiert aber nicht unmittelbar darauf, sondern sichert das Flag-Register auf dem Stack, löscht das siebte Bit mit AND &7F und gibt den Buchstaben mit TXT WR CHAR auf dem Bildschirm aus. Erst danach wird entschieden, ob die Schleife bereits zu Ende ist. Sofern dies der Fall ist, setzt das Programm den Cursor auf die Spalte 23 und gibt das Zeichen "&" als Beginn einer Hexadezimalzahl aus.

```

Mit den Befehlen INC DE
                  INC DE
                  EX DE,HL
                  LD C,(HL)
                  CALL HEX_OUTPUT

                  DEC HL
                  LD C,(HL)
                  CALL HEX_OUTPUT

                  INC HL
                  INC HL
                  EX DE,HL
                  CALL CR_LF
                  POP HL

```

gibt RSXLIST die Startadresse jeder RSX-Routine hexadezimal aus. Das Unterprogramm HEX\_OUTPUT erwartet eine 8-Bit-Zahl im C-Register und wird später noch ausführlich erläutert.

KEY\_PRESSED stellt fest, ob der Benutzer eine Taste gedrückt hat. Ist dies der Fall, wartet KEY\_PRESSED auf den Druck einer weiteren Taste, bevor es weitermacht.

LAST\_RSX ist die Abschlußroutine von RSXLIST. Es stellt den ROM-Status wieder her und kehrt nach Basic zurück.

Die Ausgabe einer hexadezimalen Zahl ist in den verschiedensten Programmen immer wieder einmal nötig, deshalb erläutern wir sie besonders ausführlich. HEX\_OUTPUT verlangt die auszugebende Zahl im C-Register. Es lädt sie dann in den Akku und springt zur Marke RIGHTSHIFT. Diese blendet die vier niederwertigen Bits des Registerinhalts aus und schiebt die vier höherwertigen in die niederwertigen Bits. Aus der Binärzahl 0111 0110 würde also 0000 0111. Im zweiten Durchgang springt das Programm die Marke RIGHTSHIFT gar nicht an, sondern setzt die Abarbeitung des Programms bei SHOWHEX fort.

Dort wird sichergestellt, daß nur die vier niederwertigen Bits stehenbleiben (AND 15). ADD "0" addiert zum Akkuinhalt den ASCII-Offset. Schließlich ist zum Beispiel "1" gleich 1+"0"=1+48 und "5" gleich 5+"0"=5+48. Ist die auszugebende Ziffer größer als 9, muß noch der Abstand zwischen Zahlen und Buchstaben im ASCII-Code berücksichtigt werden: ADD 7. DISPLAY gibt dann die Hexziffer aus und kehrt ins Hauptprogramm zurück.

## 19. Es gibt nicht nur einen Bildschirmspeicher!

Wie Sie sicher wissen, liegt der Bildschirmspeicher des Schneider-CPC im Adreßbereich zwischen &C000 (49152) und &FFFF (65535). Doch er ist nicht auf diesen Bereich festgelegt! Man kann ihn auch verschieben. Damit lassen sich sehr interessante Programmeffekte erzielen. Denken Sie etwa an Spiele, die auf blitzschnelle Grafik angewiesen sind.

Wenn Sie zum Beispiel von der Diskette oder Cassette zwei Grafikbildschirme geladen haben, können Sie mit neuen RSX-Befehlen zwischen diesen hin- und herschalten. Auch sind in gewissem Rahmen Manipulationen desjenigen Bildschirms möglich, der nicht angezeigt wird. So bemerkt der Benutzer nichts vom Zugriff des Programms auf den jeweils anderen Bildschirmspeicher.

Mit der Routine SCR SET BASE des Betriebssystems (Adresse &BC08) läßt sich die Startadresse bestimmen. Der Video-RAM darf immer nur an 16-Kilobyte-Grenzen beginnen. Er könnte also in die folgenden Speicherblöcke gelegt werden:

- |                    |                    |
|--------------------|--------------------|
| 1- &0000 bis &3FFF | 2- &4000 bis &7FFF |
| 3- &8000 bis &BFFF | 4- &C000 bis &FFFF |

Die Möglichkeit -1- scheidet von vornherein aus, ebenso der Bereich ab &8000: Zwischen &0000 und &0040 liegen die sogenannten Restart-Vektoren des 286-Prozessors. Diese Unterprogramme besorgen die richtige Auswahl der RAM- und ROM-Bänke. Eine Veränderung der Daten in diesem Bereich hätte katastrophale Folgen, da die Routinen ständig benötigt werden: mehrmals in der Sekunde!

Im Bereich &AC00 bis &BFFF liegen weitere wichtige Systeminformationen: Speicherstellen für das Betriebssystem und den Basic-Interpreter, alle Sprungvektoren für Routinen des Betriebssystems und nicht zuletzt auch der 286-Stack (Stapelspeicher). Also ist auch die Möglichkeit -3- nicht brauchbar.

Bleibt die Wahl zwischen -2- und -4-: Bei &C000 liegt der Video-RAM ja ohnehin schon, der zweite Bildschirmspeicher kann somit bei &4000 beginnen. Damit verkleinert sich allerdings der für Basic-Programme verfügbare Speicherbereich auf 16 Kilobytes.

Das RSX-Paket "SCREENUTIL" stellt Ihnen fünf neue Befehle zur Verfügung, mit denen die Arbeit mit zwei Bildschirmspeichern zum Kinderspiel wird. Der Video-RAM ab &4000 wird mit der Nummer 1 bezeichnet, derjenige ab &C000 mit der Nummer 2:

- !SCREENPAGE,x wählt, welcher Bildschirmspeicher angezeigt werden soll. !SCREENPAGE,1 verlegt den Bildschirmspeicher nach &4000, alle anderen Werte wie !SCREENPAGE,2 oder !SCREENPAGE,3 nach &C000. Beim Umschalten werden die Video-RAMs nicht gelöscht.

Sie sollten aber auf ein Scrollen des Bildschirms verzichten, da der andere Bildschirm sonst nach dem Umschalten völlig verzerrt dargestellt würde.

- !SCREENMOVE,x,y kopiert den Inhalt des einen Bildschirmspeichers in den anderen. !SCREENPAGE,1,2 lädt den Bildschirm von &4000 nach &C000, !SCREENPAGE,2,1 den Video-RAM von &C000 nach &4000.

- !SCREENSWAP tauscht den Inhalt der beiden Video-RAMs aus: Der Bildschirm von &C000 wird nach &4000 kopiert, der &4000-Bildschirm nach &C000. Der Aufruf erfolgt als !SCREENSWAP, und der RSX-Befehl benötigt keine Argumente.

- !SCREENCLEAR,x löscht den entsprechenden Grafikbildschirm, ohne daß er - wie bei CLS - gerade angezeigt werden mußte. So kann ein Bildschirm auch "im Hintergrund" gelöscht werden, ohne daß der Benutzer etwas davon bemerkt. !SCREENCLEAR,1 macht den Video-RAM zwischen &4000 und &7FFF frei, !SCREENCLEAR,2 den Speicher zwischen &C000 und &FFFF.

- !SCREENFILL,x,y füllt den angegebenen Video-RAM mit einem beliebigen Bitmuster auf. !SCREENFILL,1,0 füllt den Bildschirm 1 mit lauter Nullbytes, macht also dasselbe wie !SCREENCLEAR,1. Die anderen Bitwerte hängen vom gewählten Bildschirm-MODE ab. In den Bytes, die die Zeichen im Video-RAM darstellen, ist nämlich auch noch die PEN-Farbe codiert.

Am einfachsten ist die Sache im MODE 2: Jedes gesetzte Bit des angegebenen Bytes wird als gesetzter Punkt auf dem Monitor sichtbar, zum Beispiel füllt !SCREENFILL,2,255 den Bildschirm 2 vollständig mit der Farbe von PEN 1. Mit etwas Rechnerei kann man mit SCREENFILL den Basic-Befehl CLG-"farbe" für beide Bildschirmspeicher simulieren.

Denken Sie aber immer daran, mit MEMORY &3FFF die Obergrenze des verfügbaren Basic-Speichers unter den Bildschirm-RAM zu legen. Sonst kann es zu recht merkwürdigen Reaktionen des Computers kommen.

Hier zuerst das Sourcelisting des Maschinenprogramms und direkt danach der Basic-Loader.

### 19.1 Assemblerlisting: SCREENUTIL

```
; *****
; *
; *   SCREENUTIL - Utilities zum Arbeiten mit zwei Grafik-Bildschirmen
; *
; *****
```

```
ORG      &A000      ; Mit dem Basic-Lader verschiebbar!
```

```
;
; Definitionen *****
; -----
```

```
KL_UR0M_ENABLE EQU    &B900      ; KL U ROM ENABLE
```

```

TXT_OUTPUT      EQU      &BB5A      ; TXT OUTPUT
SCR_SET_OFFSET  EQU      &BC05      ; SCR SET OFFSET
SCR_SET_BASE    EQU      &BC08      ; SCR SET BASE
SCR_GET_LOCATN  EQU      &BC0B      ; SCR GET LOCATION
KL_LOG_EXT      EQU      &BCD1      ; KL LOG EXT
BASIC_ERROR     EQU      &CA94      ; CPC-664=&CB50, CPC-6128=&CB4D

;
; RSX-Erweiterung einbinden *****
; -----

INIT_UTILITRXX LD      BC,JUMPTABLE ; Zeiger auf Sprungtabelle
                LD      HL,KERNEL_RAM ; Zeiger auf Hilfsppeicher
                CALL    KL_LOG_EXT    ; RSX-Erweiterung initialisieren
                RET      ; Rücksprung nach Basic

;
; Daten zur Verwaltung der SCREENUTIL-RSXen *****
; -----

JUMPTABLE       DEFW     NAMETABLE   ; Zeiger auf Namenstabelle
                JP      SCREENPAGE   ; Sprung nach SCREENPAGE
                JP      SCREENMOVE   ; Sprung nach SCREENMOVE
                JP      SCREENSWAP   ; Sprung nach SCREENSWAP
                JP      SCREENCLEAR  ; Sprung nach SCREENCLEAR
                JP      SCREENFILL   ; Sprung nach SCREENFILL

NAMETABLE        DEFW     "SCREENPAG" ; RSX-Name "SCREENPAGE"
                DEFB     "E"+&80
                DEFW     "SCREENMOV"  ; RSX-Name "SCREENMOVE"
                DEFB     "E"+&80
                DEFW     "SCREENSWA"  ; RSX-Name "SCREENSWAP"
                DEFB     "P"+&80
                DEFW     "SCREENCLEA" ; RSX-Name "SCREENCLEAR"
                DEFB     "R"+&80
                DEFW     "SCREENFIL"  ; RSX-Name "SCREENFILL"
                DEFB     "L"+&80
                DEFB     &80         ; Tabellenende

KERNEL_RAM       DEFS     &04        ; Zur internen Verwaltung

; *****
;
; SCREENPAGE legt den Beginn des Bildschirmspeichers fest
; -----
;
; !SCREENPAGE,1   Der Bildschirm befindet sich ab &4000
; !SCREENPAGE,2   Der Bildschirm befindet sich ab &C000
;
; *****

SCREENPAGE        CP      1          ; Genau ein Argument aus Basic?
                JP      NZ,ERROR     ; Nein - Fehler!
                LD      A,(IX+0)     ; Argument in den Akku

SCREEN_4000        CP      1          ; Akku=1
                JR      NZ,SCREEN_C000 ; Nein, Bildschirm ab &C000
                LD      A,&4000      ; Ja, Bildschirm ab &4000

```

```

CALL      SCR_SET_BASE    ; Ans Betriebssystem melden
RET      ; Rücksprung nach Basic

SCREEN_C000 LD      A,&C0    ; Akku=2, Bildschirmspeicher
CALL      SCR_SET_BASE    ; liegt ab &C000
RET      ; Rücksprung nach Basic

; *****
;
; SCREENMOVE kopiert einen Grafikbildschirm in den anderen hinein
; -----
;
; !SCREENMOVE,1,2  Der Bildschirm wird von &4000 nach &C000 kopiert
; !SCREENMOVE,2,1  Der Bildschirm wird von &C000 nach &4000
kopiert
;
; *****

SCREENMOVE CP      2      ; Genau zwei Argumente aus Basic?
JP      NZ,ERROR    ; Nein, Fehler!

MOVE_SOURCE LD      A,(IX+2) ; Erstes Argument lesen
CP      1      ; Akku=1?
JR      NZ,M_SRC_C000

M_SRC_4000 LD      HL,&4000    ; Ja - Quell-Bildschirm &4000
JR      MOVE_DESTIN

M_SRC_C000 LD      HL,&C000    ; Nein - Quell-Bildschirm &C000

MOVE_DESTIN LD      A,(IX+0) ; Zweites Argument lesen
CP      1      ; Akku=1?
JR      NZ,M_DEST_C000

M_DEST_4000 LD      DE,&4000    ; Ja - Ziel-Bildschirm ab &4000
JR      MOVE_IT

M_DEST_C000 LD      DE,&C000    ; Nein - Zielbildschirm ab &C000

MOVE_IT LD      BC,&4000    ; Video-RAM hat 16 Kilobytes
LDIR    ; Speicher verschieben
RET      ; Basic-Rücksprung

; *****
;
; SCREENSWAP tauscht die Inhalte der beiden Grafikbildschirme aus
; -----
;
; !SCREENSWAP
;
; *****

SCREENSWAP CP      0      ; Kein Argument aus Basic erlaubt!
JR      NZ,ERROR    ; Sonst Fehlermeldung!

SWAP_IT LD      HL,&C000    ; Bildschirm 1 ab &C000
LD      DE,&4000    ; Bildschirm 2 ab &4000

```



```

SWAP_LOOP      LD      A,(DE)      ; Ein Byte lesen
               PUSH    AF          ; Sichern
               LD      A,(HL)      ; Auf dem anderen Schirm lesen
               LD      (DE),A      ; Auf den ersten Schirm schreiben
               POP     AF          ; Altes Byte wiederherstellen
               LD      (HL),A      ; Auf den zweiten Schirm schreiben

SWAP_NEXT      INC      HL          ; Zeiger auf Bildschirm 2 erhöhen
               INC      DE          ; Zeiger auf Bildschirm 1 erhöhen
               PUSH    DE          ; Register sichern
               LD      DE,&FFFF     ; Ende des zweiten Bildschirms
               PUSH    HL          ; Ersetzt CP HL,DE
               OR      A           ; ""
               SBC     HL,DE       ; ""
               POP     HL          ; ""
               POP     DE          ; DE-Register wiederherstellen
               JR      NZ,SWAP_LOOP ; Weitermachen, wenn nicht fertig
               RET                ; Sonst Basic-Rücksprung

```

```

; *****
;
; SCREENCLEAR löscht einen der beiden Grafikbildschirme
; -----
;
; !SCREENCLEAR,1   löscht den Bildschirm ab &4000
; !SCREENCLEAR,2   löscht den Bildschirm ab &C000
;
; *****

```

```

SCREENCLEAR    CP      1           ; Genau ein Argument ist erlaubt
               JR      NZ,ERROR    ; Wenn nicht korrekt, dann Fehler
               LD      A,(IX+0)    ; Argument lesen
               CP      1           ; Ist Akku=1?
               JR      NZ,CLS_C000

```

```

CLS_4000       LD      HL,&4000    ; Mit LDIR den Speicher löschen
               LD      DE,&4001
               LD      BC,&3FFF     ; BC=Zähler
               LD      (HL),0      ; Null ins erste Byte schreiben
               LDIR              ; Löscht den Speicher
               RET                ; Rücksprung nach Basic

```

```

CLS_C000       LD      HL,&C000    ; Mit LDIR den Speicher löschen
               LD      DE,&C001
               LD      BC,&3FFF     ; BC=Zähler
               LD      (HL),0      ; Null ins erste Byte schreiben
               LDIR              ; Löscht den Speicher
               RET                ; Rücksprung nach Basic

```

```

; *****
;
; SCREENFILL füllt einen der Bildschirme mit einem Bytewert
; -----
;
; Zum Beispiel !SCREENFILL,1,255
; !SCREENFILL,2,125
;
; *****

```

```

;
; *****
SCREENFILL    CP      2           ; Genau zwei Argumente
              JR      NZ,ERROR    ; Wenn nicht korrekt, dann Fehler
              LD      A,(IX+2)    ; Argument lesen
              CP      1           ; Ist Akku=1?
              JR      NZ,FILL_C000

FILL_4000     LD      A,(IX+0)    ; Fullbyte lesen
              LD      HL,&4000    ; Mit LDIR den Speicher füllen
              LD      DE,&4001
              LD      BC,&3FFF    ; BC=Zähler
              LD      (HL),A      ; Fullbyte ins erste Byte schreiben
              LDIR              ; Z80 ausgetrickst
              RET                ; Rücksprung nach Basic

FILL_C000     LD      A,(IX+0)    ; Fullbyte lesen
              LD      HL,&C000    ; Mit LDIR den Speicher füllen
              LD      DE,&C001
              LD      BC,&3FFF    ; BC=Zähler
              LD      (HL),A      ; Fullbyte ins erste Byte schreiben
              LDIR              ; Z80-Programmiertrick
              RET                ; Rücksprung nach Basic

;
; Fehlerausgaberroutine *****
; -----

ERROR         CALL    KL_UROM_ENABLE ; Basic-ROM einschalten
              LD      E,5         ; "Improper Argument"
              JP      BASIC_ERROR  ; Ausgeben

              END

; -----

```

## 19.2 Listing: SCREENU.BAS

```

100 ' *****
110 ' *****
120 ' *
130 ' *      SCREEN UTILITIES      *
140 ' *
150 ' *****
160 ' *****
170 '
180 DEF FNmsb(a)=255 AND INT(a/256)
190 DEF FNlsb(a)=255 AND UNT(a)
200 ' Titelbild und Benutzer-Informationen -----
210 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
220 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
230 LOCATE 1,1:PRINT STRING$(40,210);
240 LOCATE 1,2
250 PRINT CHR$(24)+SPACE$(12)+"SCREEN UTILITIES"+SPACE$(12)+CHR$(24)
260 LOCATE 10,5:PRINT CHR$(164);" Martin Kotulla 1986"
270 PRINT:PRINT STRING$(40,210)
280 PEN 1:PRINT:PRINT " Das RSX-Paket SCREEN UTILITIES bietet"

```

```

290 PRINT:PRINT " eine Reihe von Befehlen, die die"
300 PRINT:PRINT " Arbeit mit zwei getrennten Bild-"
310 PRINT:PRINT " schirmspeichern erlauben:"
320 PEN 3:PRINT STRING$(40,210):PRINT
330 PEN 2:PRINT " SCREENPAGE - SCREENMOVE - SCREENSWAP"
340 PRINT:PRINT TAB(9);"SCREENCLEAR - SCREENFILL":PEN 1
350 FOR i=1 TO 280:READ a:sum=sum+a:NEXT i
360 IF sum=-218120 THEN 390
370 PRINT:PRINT CHR$(7);" * Fehler in den DATA-Zeilen!":PRINT
380 LIST 540-:END
390 RESTORE:MEMORY HIMEM-274:start=HIMEM+1
400 FOR i=start TO start+269:READ a:POKE i,a:NEXT i
410 FOR i=1 TO 10:READ a:a=a-40960+start
420 value=PEEK(a)+PEEK(a+1)*256-40960+start
430 POKE a,FN!sb(value):POKE a+1,FN!sb(value)
440 NEXT i
450 CALL start ' RSX-Befehle installieren
460 ' Programm an den Computertyp anpassen -----
470 version$=CHR$(&CD)+CHR$(&D)+CHR$(&B9)+CHR$(&3A)+CHR$(2)+CHR$(&C0)
480 version$=version$+CHR$(&32)+CHR$(&66)+CHR$(1)+CHR$(&C9)
490 CALL PEEK(0version$+1)+PEEK(0version$+2)*256
500 version=PEEK(&166)
510 IF version=0 THEN END
520 POKE start+&10C,&CB
530 IF version=1 THEN POKE start+&10B,&50 ELSE POKE start+&10B,&4D
540 ' DATAs fuer das Maschinenprogramm -----
550 DATA &01,&0A,&A0,&21,&4F,&A0,&CD,&D1,&BC,&C9,&1B,&A0,&C3,&53,&A0,&C3
560 DATA &6B,&A0,&C3,&94,&A0,&C3,&B3,&A0,&C3,&DA,&A0,&53,&43,&52,&45,&45
570 DATA &4E,&50,&41,&47,&C5,&53,&43,&52,&45,&45,&4E,&4D,&4F,&56,&C5,&53
580 DATA &43,&52,&45,&45,&4E,&53,&57,&41,&D0,&53,&43,&52,&45,&45,&4E,&43
590 DATA &4C,&45,&41,&D2,&53,&43,&52,&45,&45,&4E,&46,&49,&4C,&CC,&00,&00
600 DATA &00,&00,&00,&FE,&01,&C2,&05,&A1,&DD,&7E,&00,&FE,&01,&20,&06,&3E
610 DATA &40,&CD,&08,&BC,&C9,&3E,&C0,&CD,&08,&BC,&C9,&FE,&02,&C2,&05,&A1
620 DATA &DD,&7E,&02,&FE,&01,&20,&05,&21,&00,&40,&18,&03,&21,&00,&C0,&DD
630 DATA &7E,&00,&FE,&01,&20,&05,&11,&00,&40,&18,&03,&11,&00,&C0,&01,&00
640 DATA &40,&ED,&B0,&C9,&FE,&00,&20,&6D,&21,&00,&C0,&11,&00,&40,&1A,&F5
650 DATA &7E,&12,&F1,&77,&23,&13,&D5,&11,&FF,&FF,&E5,&B7,&ED,&52,&E1,&D1
660 DATA &20,&EC,&C9,&FE,&01,&20,&4E,&DD,&7E,&00,&FE,&01,&20,&0E,&21,&00
670 DATA &40,&11,&01,&40,&01,&FF,&3F,&36,&00,&ED,&B0,&C9,&21,&00,&C0,&11
680 DATA &01,&C0,&01,&FF,&3F,&36,&00,&ED,&B0,&C9,&FE,&02,&20,&27,&DD,&7E
690 DATA &02,&FE,&01,&20,&10,&DD,&7E,&00,&21,&00,&40,&11,&01,&40,&01,&FF
700 DATA &3F,&77,&ED,&B0,&C9,&DD,&7E,&00,&21,&00,&C0,&11,&01,&C0,&01,&FF
710 DATA &3F,&77,&ED,&B0,&C9,&CD,&00,&B9,&1E,&05,&C3,&94,&CA,&00
720 ' DATAs zum Verschieben des Maschinencodes -----
730 DATA &A001,&A004,&A00A,&A00D,&A010,&A013,&A016,&A019,&A056,&A06E
740 END ' -----

```

Für alle, die auch daran interessiert sind, wie SCREENUTIL funktioniert, sind die folgenden Zeilen gedacht:

- SCREENPAGE prüft zuerst, ob ein einzelner Parameter übergeben wurde. Trifft dies nicht zu, gibt das Programm eine Fehlermeldung "Improper Argument" aus. SCREEN\_40000 stellt fest, ob der Bildschirmspeicher ab &40000 eingeschaltet werden soll. Wenn dies zutrifft, lädt es den Akku mit &40, dem Highbyte von &40000, und ruft die ROM-Routine SCR SET BASE (&BC08) auf. Andernfalls kommt

SCREEN\_C000 zum Zuge und blendet den Bildschirmspeicher ab &C000 ein.

- SCREENMOVE benötigt zwei Parameter (woher? wohin?). Geben Sie nicht genau diese zwei Daten an, kommt es zu einer Fehlermeldung. Mit LD A,(IX+2) liest das Programm die Bildschirmquelle und lädt das HL-Register abhängig vom Wert mit dem Beginn des Video-RAMs, der als Quelle dient - entweder bei M\_SRC\_4000 mit &4000 oder bei M\_SRC\_C000 mit &C000. Ähnlich verfährt MOVE\_DESTIN mit dem zweiten Parameter aus dem RSX-Aufruf: Das DE-Register enthält entweder &4000 oder &C000. MOVE\_IT benutzt den LDIR-Befehl des Z80-Prozessors, um den Bildschirmspeicher umzukopieren. LDIR verlangt folgende Registerwerte:

- HL-Register: Quelle der Daten
- DE-Register: Ziel der Daten
- BC-Register: Zahl der zu kopierenden Bytes

- SCREENSWAP erwartet keinen Parameter und gibt deshalb auch ein "Improper Argument" aus, wenn Sie partout einen oder sogar mehrere übergeben wollen. SWAP\_IT lädt das HL-Register mit einem Zeiger auf den Anfang von Bildschirm 2 (&C000), das DE-Register zeigt auf den Anfang von Bildschirm 1 (&4000). SWAP\_LOOP hat die Aufgabe, jedes Byte des einen Speicherbereichs gegen das Byte des anderen auszutauschen:

```
LD  A,(DE)  liest ein Byte aus dem ersten Bildschirm.
PUSH AF    sichert es auf dem Stack.
LD  A,(HL)  liest ein Byte aus dem zweiten Bildschirm.
LD  (DE),A  überträgt es in den ersten Bildschirm.
POP  AF    holt das alte Byte zurück.
LD  (HL),A  überträgt es in den zweiten Bildschirmspeicher.
```

In SWAP\_NEXT werden die beiden Zeiger inkrementiert. Da der Z80-Prozessor keinen Befehl wie CP HL,&FFFF besitzt, muß das Ende des Austauschvorgangs mit einigen Maschinenbefehlen festgestellt werden:

```
PUSH DE    rettet das DE-Register.
LD  DE,&FFFF enthält das Vergleichsobjekt.
PUSH HL    rettet das HL-Register.
OR  A      löscht Carry-Flag.
SBC HL,DE  subtrahiert DE von HL und setzt Flags.
POP  HL    stellt das HL-Register wieder her.
POP  DE    stellt das DE-Register wieder her.
```

Der darauf folgende bedingte Sprungbefehl JR NZ,SWAP\_LOOP wird nur ausgeführt, solange der Inhalt des HL-Registers kleiner als &FFFF ist.

- SCREENCLEAR verlangt die gewünschte Bildschirmnummer als Argument aus Basic. Geben Sie hier keinen oder zuviele Parameter an, führt das unweigerlich zu einer Fehlermeldung. LD A,(IX+0) bestimmt, welche Bildschirmnummer gemeint ist. Abhängig davon springt das Programm zur Marke CLS\_4000 oder zu CLS\_C000. Das Löschen des Bildschirms geschieht auf eine etwas ungewöhnliche Art: Durch einen zweckentfremdeten LDIR-Befehl!

Normalerweise muß der Programmierer bei sich überlappenden Speicherbereichen überlegen, ob er LDIR oder LDDR zum Kopieren benutzen will. LDIR kopiert von unten nach oben, LDDR von oben nach unten. Auf diese Art können auch sich überlappende Speicherbereiche korrekt übertragen werden. Genau die umgekehrte Überlegung führt zur Lösung: Wenn die Speicherbereiche sich überlappen und nur ein Byte auseinanderliegen, kopiert LDIR den Inhalt der ersten Speicherzelle in alle anderen! Auf diese Art kann der Bildschirm mit Höchstgeschwindigkeit geleert werden:

```
LD    HL,Beginn      Zeiger auf erste Speicherstelle
LD    DE,Beginn+1    Zeiger auf zweite Speicherstelle
LD    BC,Länge-1     Zahl der zu kopierenden Bytes -1
LD    (HL),0         Gewünschtes Löschbyte eintragen
LDIR                                     Löschen
```

- SCREENFILL funktioniert auf ganz ähnliche Art. Es verwendet nur nicht die Null zum Füllen des Speichers, sondern den Parameter, der im RSX-Aufruf steht.

Damit wissen Sie alles, was Sie zum Arbeiten und zum Verstehen von SCREENUTIL brauchen. Falls Ihnen der Speicherplatz einmal knapp werden sollte, können Sie mit ein paar Tricks mehrere Kilobytes wieder herausholen. Der Speicherbereich von &8000 bis zum alten HIMEM bei circa 42500 ist nämlich gar nicht belegt!

Sie können dorthin die Zeichentabellen, die Pufferspeicher für Cassetten- und Diskettendateien und sogar Maschinenprogramme legen.

Fall 1: Die Zeichentabelle soll in den unbenutzten Speicher gebracht werden:

```
SYMBOL AFTER 0
MEMORY &3FFF
```

Hier wird vorausgesetzt, daß HIMEM vorher bei dem Wert lag, den er schon beim Einschalten des Computers hatte.

Fall 2: Der Dateipuffer soll im unbenutzten Speicher liegen:

```
OPENOUT "DUMMY.***"
MEMORY HIMEM-1
CLOSEOUT
MEMORY &3FFF
```

Fall 3: Es wird sowohl die Zeichentabelle als auch der Ausgabepuffer benötigt:

```
SYMBOL AFTER 256
OPENOUT "DUMMY.***"
MEMORY HIMEM-1
CLOSEOUT
```

SYMBOL AFTER Ø  
MEMORY &3FFF

Fall 4: Platz für Maschinenprogramme mit oder ohne Zeichentabellen und Ausgabepuffer: Verwenden Sie die Befehle der Beispiele 1 bis 3, aber lassen Sie den MEMORY &3FFF-Befehl vorerst weg.

PRINT HIMEM liefert die letzte benutzbare Speicheradresse für ihre Maschinenprogramme. Sie können also im Bereich von &8000 bis HIMEM Maschinencode unterbringen. Geben Sie aber vor dem EinPOKEN des Programms ein MEMORY &7FFF oder MEMORY &3FFF ein.

Und noch ein ganz heißer Tip für 6128-Besitzer: Auch ohne den Bank-Manager BANKMAN können Sie mit SCREENUTIL insgesamt sechs Bildschirme benutzen: Den normalen ab der Adresse &C000, den zweiten ab &4000 und vier weitere in der zweiten 64K-Speicherbank.

Dazu müssen Sie nur wissen, daß sich mit OUT-Befehlen auch in Basic vier 16-Kilobyte-Blöcke einblenden lassen - glücklicherweise gerade in den Bereich &4000 bis &7FFF!

- OUT &7F00,&C0 benutzt den normalen 16K-Block im 64K-Hauptspeicher.
- OUT &7F00,&C4 benutzt den ersten 16K-Block in der zweiten Speicherbank.
- OUT &7F00,&C5 benutzt den zweiten 16K-Block in der zweiten Speicherbank.
- OUT &7F00,&C6 benutzt den dritten 16K-Block in der zweiten Speicherbank.
- OUT &7F00,&C7 benutzt den vierten 16K-Block in der zweiten Speicherbank.

Wenn Sie den Bildschirmspeicher nach &4000 verlegt haben und die OUT-Befehle verwenden, werden Sie feststellen, daß Sie den Computer nicht dazu bringen können, einen Speicherblock anzuzeigen. Sie können in der zweiten RAM-Bank aber sehr wohl Bildschirme "deponieren" und später mit SCREENMOVE zurückholen!

## 20. Blitzschnell eingeblendet

Bei "perfekten" Spielprogrammen soll der Benutzer meistens nicht merken, wie der Grafikbildschirm aufgebaut wird. Dazu müßte man irgendwie den Bildschirm abschalten können.

Mit zwei neuen RSX-Befehlen, um die das Programm HIDE & SHOW den Basic-Interpreter erweitert, läßt sich das ganz einfach machen.

Das Maschinenprogramm kann als Quellcode eingetippt und dann von einem Assembler übersetzt werden:

### 20.1 Assemblerlisting: HIDESHOW

```
; *****
; *
; * RSX-Erweiterung - Blitzschnelles An- und Abschalten des Bildschirms *
; *
; *****

                ORG      &A000          ; Mit dem Basic-Lader verschiebbar

;
; Definitionen und Deklarationen *****
; -----

KL_UROM_ENABLE EQU      &B900          ; KL U ROM ENABLE
SCR_SET_INK    EQU      &BC32          ; SCR SET INK
SCR_GET_INK    EQU      &BC35          ; SCR GET INK
KL_LOG_EXT     EQU      &BCD1          ; KL LOG EXT
BASIC_ERROR    EQU      &CA94          ; CPC-664=&CB50, CPC-6128=&CB4D

INKBUFFER      DEFS     36              ; Speichertabelle für Inks

;
; Einbindung der RSX-Befehle *****
; -----

INIT_RSX       LD        BC,JUMPTABLE  ; Zeiger auf Sprungtabelle
                LD        HL,KERNEL_RAM ; Zeiger auf Hilfsspeicher
                CALL      KL_LOG_EXT    ; RSX ins Betriebssystem einbinden
                RET              ; Rücksprung nach Basic

;
; Daten zur Verwaltung der neuen RSX-Befehle *****
; -----

JUMPTABLE      DEFW      NAMETABLE     ; Zeiger auf Namenstabelle
                JP        HIDE
                JP        SHOW

NAMETABLE       DEFW      "HID"         ; RSX-Name "HIDE"
                DEFB      "E"&80
                DEFW      "SHO"        ; RSX-Name "SHOW"
                DEFB      "W"&80
```

```

DEFB      &00      ; Tabellenende

KERNEL_RAM  DEFS      &04      ; 4 Bytes fürs Betriebssystem

; RSX-Befehl HIDE -----
;
; !HIDE,ink1,ink2  !läßt Bildschirm verschwinden und
;                  setzt alle Inks auf ink1,ink2
;
; -----

HIDE        CP        2          ; Zwei Argumente zulassen
            JR        NZ,ERROR    ; Sonst Fehler
            LD        B,16        ; 16 Inks zu sichern
            LD        HL,INKBUFFER ; Zeiger auf Speichertabelle

GET_INKS_LOOP LD      A,B          ; Akku=Inknummer
            DEC      A            ; Intern dekrementieren
            PUSH     BC           ; BC-Register sichern
            PUSH     HL           ; HL-Register sichern
            CALL     SCR_GET_INK  ; Ink-Farbe setzen
            POP      HL           ; HL-Register wiederherstellen
            LD       (HL),B       ; Erste Ink abspeichern
            INC      HL           ; Zeiger erhöhen
            LD       (HL),C       ; Zweite Ink abspeichern
            INC      HL           ; Zeiger erhöhen
            POP      BC           ; BC-Register wiederherstellen
            DJNZ     GET_INKS_LOOP ; Weiter, bis alle Inks kopiert

GET_INK_PARAM LD      B,(IX+2)     ; Ink1 holen
            LD      C,(IX+0)     ; Ink2 holen
            SUB      A           ; Pen=0

INK_PAR_LP   PUSH     AF          ; Akku & Flags sichern
            PUSH     BC          ; BC-Register sichern
            CALL     SCR_SET_INK  ; Ink setzen
            POP      BC          ; BC-Register wiederherstellen
            POP      AF          ; Akku & Flags wiederherstellen
            INC      A           ; Pen-Nummer erhöhen
            CP       16          ; Letzten Pen erreicht?
            JR       NZ,INK_PAR_LP ; Weitermachen, bis fertig
            RET                ; Rücksprung nach Basic

; RSX-Befehl SHOW -----
;
; !SHOW zeigt den Grafikbildschirm wieder an
; und stellt die alten Ink-Farben wieder her
;
; -----

SHOW        CP        0          ; Kein Parameter erlaubt!
            JR       NZ,ERROR    ; Sonst Fehler
            LD       B,16        ; 16 Inks zu laden
            LD       HL,INKBUFFER ; Zeiger auf Speichertabelle

SET_INKS_LOOP LD      A,B          ; Akku=Pen

```



```

DEC      A          ; Zur internen Verwaltung -1
PUSH     BC         ; BC-Register sichern
LD       B,(HL)     ; InK1 holen
INC      HL         ; Zeiger erhöhen
LD       C,(HL)     ; InK2 holen
INC      HL         ; Zeiger erhöhen
PUSH     HL         ; HL-Register sichern
CALL     SCR_SET_INK ; Beide Inks setzen
POP      HL         ; HL-Register wiederherstellen
POP      BC         ; BC-Register wiederherstellen
DJNZ     SET_INKS_LOOP ; Weitermachen bis fertig
RET      ; Rücksprung nach Basic

```

```

;
; BASIC_ERROR - "Improper Argument" ausgeben lassen *****
; -----

```

```

ERROR      CALL     KL_UROM_ENABLE ; Basic-ROM einschalten
LD         E,5       ; "Improper Argument"
JP         BASIC_ERROR ; Fehler ausgeben

```

```

END

```

```

; *****

```

Einfacher geht es wohl, wenn Sie gleich den Basic-Loader eingeben. Dieser legt HIDE & SHOW direkt unter HIMEM ab und zeigt auch eine kleine Demonstration der neuen Befehle:

## 20.2 Listing: HIDESHOW.BAS

```

100 ' *****
110 ' *
120 ' *      HIDE & SHOW      *
130 ' *
140 ' *****
150 '
160 DEF FNmsb(a)=255 AND INT(a/256)
170 DEF FNlsb(a)=255 AND UNT(a)
180 ' Titelbild und Benutzer-Informationen -----
190 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
200 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
210 LOCATE 1,1:PRINT STRING$(40,210);
220 LOCATE 1,2
230 PRINT CHR$(24)+SPACE$(14)+"HIDE & SHOW"+SPACE$(15)+CHR$(24)
240 PRINT:PRINT STRING$(40,210)
250 PEN 1:PRINT:PRINT " HIDE & SHOW sind zwei neue Befehle,"
260 PRINT:PRINT " die das blitzschnelle Abschalten"
270 PRINT:PRINT " und Einschalten des Grafikbild-"
280 PRINT:PRINT " schirms erlauben:"
290 PEN 3:PRINT:PRINT STRING$(40,210):PRINT
300 PEN 2:PRINT " !HIDE,newink1,newink2"
310 PRINT:PRINT " !SHOW":PRINT:PEN 1
320 FOR i=1 TO 151:READ a:sum=sum+a:NEXT i
330 IF sum=-159293 THEN 360
340 PRINT:PRINT CHR$(7);" * Fehler in den DATA-Zeilen!":PRINT

```

```

350 LIST 630-:END
360 RESTORE:MEMORY HIMEM-148:start=HIMEM+1
370 FOR i=start TO start+143:READ a:POKE i,a:NEXT i
380 FOR i=1 TO 7:READ a:a=40960+start
390 value=PEEK(a)+PEEK(a+1)*256-40960+start
400 POKE a,FNlsb(value):POKE a+1,FNmsb(value)
410 NEXT i
420 CALL start+24 ' RSX-Befehle installieren
430 ' Programm an den Computertyp anpassen -----
440 version$=CHR$(&CD)+CHR$(&0)+CHR$(&B9)+CHR$(&3A)+CHR$(2)+CHR$(&C0)
450 version$=version$+CHR$(&32)+CHR$(&66)+CHR$(1)+CHR$(&C9)
460 CALL PEEK(@version$+1)+PEEK(@version$+2)*256
470 version=PEEK(&166)
480 IF version=0 THEN 520
490 POKE start+8F,&CB
500 IF version=1 THEN POKE start+8E,&50 ELSE POKE start+8E,&4D
510 ' Demoprogramm -----
520 MODE 0 ' Auf 16-Farben-Modus umschalten
530 FOR i=0 TO 15 ' Alle 16 Pens auf Zufallswerte
540 INK i,RND*27 ' setzen
550 PEN i:PRINT STRING$(20,143); ' und auf dem Bildschirm ausgeben
560 NEXT i
570 FOR i=1 TO 100 ' 100mal die Demonstration durchlaufen
580 :HIDE,0,0:BORDER 0 ' Bildschirm auf schwarz setzen
590 FOR j=1 TO 400:NEXT j ' Warteschleife
600 :SHOW ' Wieder anzeigen
610 FOR j=1 TO 400:NEXT j ' Warteschleife
620 NEXT i
630 ' DATAs fuer das Maschinenprogramm -----
640 DATA &00,&00,&00,&00,&00,&00,&00,&00,&00,&00,&00,&00,&00,&00,&00,&00
650 DATA &00,&00,&00,&00,&00,&00,&00,&00,&00,&00,&00,&00,&00,&00,&00,&00
660 DATA &00,&00,&00,&00,&01,&2E,&A0,&21,&3F,&A0,&CD,&D1,&BC,&C9,&36,&A0
670 DATA &C3,&43,&A0,&C3,&6F,&A0,&48,&49,&44,&C5,&53,&48,&4F,&D7,&00,&00
680 DATA &00,&00,&00,&FE,&02,&20,&41,&06,&10,&21,&00,&A0,&78,&3D,&C5,&E5
690 DATA &CD,&35,&BC,&E1,&70,&23,&71,&23,&C1,&10,&F1,&DD,&46,&02,&DD,&4E
700 DATA &00,&97,&F5,&C5,&CD,&32,&BC,&C1,&F1,&3C,&FE,&10,&20,&F4,&C9,&FE
710 DATA &00,&20,&15,&06,&10,&21,&00,&A0,&78,&3D,&C5,&46,&23,&4E,&23,&E5
720 DATA &CD,&32,&BC,&E1,&C1,&10,&F1,&C9,&CD,&00,&B9,&1E,&05,&C3,&94,&CA
740 ' DATAs, die das Verschieben des Maschinencodes moeglich machen ----
750 DATA &A025,&A028,&A02E,&A031,&A034,&A04A,&A076
760 END ' *****

```

Die beiden neuen RSX-Kommandos haben folgende Syntax:

```

:HIDE,ink1,ink2
:SHOW

```

HIDE laßt alle Grafiken und Texte auf dem Bildschirm verschwinden, indem es alle PENS auf die beiden Werte setzt, die beim Aufruf angegeben werden:

```

:HIDE,0,0 macht den Bildschirm schwarz.
:HIDE,0,26 laßt ihn schwarz und weiß blinken.

```

Die Blinkgeschwindigkeit können Sie ganz normal mit dem Basic-Befehl SPEED INK bestimmen. Die BORDER-Farbe wird von HIDE & SHOW

nicht beeinflusst.

So können Sie ganz reizvolle Effekte erzielen, wenn Sie den Bildschirmrand und die Bildschirmfläche im Takt blinken lassen:

```
:HIDE,3,18:BORDER 18,3
```

SHOW zeigt auf einen Schlag den gesamten Bildschirm wieder an. Die alten PEN-Farben wurden von HIDE intern gespeichert und erscheinen jetzt wieder in voller Pracht.

Ein kleines Programmbeispiel soll die Anwendung verdeutlichen:

```
100 MODE 0
110 FOR I=0 TO 15
120 INK I,RND*27
130 NEXT I
140 :HIDE,0,0:BORDER 0
150 FOR I=1 TO 1000
160 PLOT RND*640,RND*400,RND*15
170 NEXT I
180 :SHOW:END
```

Und hier finden Sie noch Erläuterungen zum Programmaufbau und zur Funktionsweise von HIDE & SHOW:

- HIDE prüft, ob zwei Argumente mit dem RSX-Befehl angegeben wurden - die beiden INK-Farben, auf die sämtliche PENs gesetzt werden sollen. Sollten Sie einen Fehler gemacht haben, quittiert das Programm diesen mit einer Fehlermeldung "Improper Argument". Andernfalls verwendet HIDE das B-Register als Schleifenzähler zum Abspeichern sämtlicher INK-Farben. Das HL-Register zeigt auf den Beginn des verwendeten Pufferspeichers.

- GET\_INKS\_LOOP holt den Schleifenzähler in den Akku und dekrementiert den Wert, da die Systemroutine SCR GET INK die Zählung der Farben bei Null beginnt, nicht bei Eins. Die letzte PEN-Nummer ist für SCR GET INK auch nicht 16, sondern 15. Da die Routine den Inhalt des BC- und des HL-Registers zerstört, muß dieser mit PUSH BC und PUSH HL auf dem Stack gesichert werden. CALL SCR GET INK holt dann das Farbenpaar ins BC-Register. POP HL stellt den Zeiger auf den INKBUFFER wieder her, und die vier folgenden Befehle tragen die Farbe in diesen Pufferspeicher ein. POP BC holt den Inhalt des Schleifenzählers ins B-Register zurück.

DJNZ GET\_INKS\_LOOP springt solange in die Schleife zurück, wie noch nicht alle Farben gespeichert sind.

- GET\_INK\_PARAM holt sich mit Hilfe des Indexregisters IX die beiden Farbwerte aus dem Basic-Aufruf ins BC-Register und setzt die PEN-Nummer im Akkumulator auf &00 (SUB A).

- in INK\_PAR\_LP müssen zuerst der Akkumulator, das Flag-Register und das BC-Doppelregister auf den Stack gebracht werden. Erst dann kann die INK-Farbe einer PEN-Nummer zugewiesen werden. POP BC und POP AF stellen die Registerwerte wieder her. INC A erhöht

die PEN-Nummer um Eins. Sobald der Vergleich CP 16 positiv ausfällt, verläßt der Computer die Programmschleife.

- Bei SHOW dürfen keine Parameter im RSX-Befehl übermittelt werden. Deshalb prüft das Programm mit CP 0, ob alles korrekt ist. LD B,16 dient wieder als Schleifenzähler - diesmal für eine Schleife zum Wiederherstellen der INK-Farben. HL zeigt auf den Speicher für die Farben.

- SET\_INKS\_LOOP lädt erst einmal die PEN-Nummer in den Akku und dekrementiert den Wert für die interne Verwaltung. PUSH BC rettet den Schleifenzähler auf den Stapelspeicher des Prozessors. Die Befehle LD B,(HL):INC HL:LD C,(HL):INC HL holen ein INK-Farbenpaar ins BC-Register. PUSH HL schickt den Inhalt des HL-Registers auf den Stack, worauf dann unbesorgt CALL SCR SET INK aufgerufen werden kann, das eine INK einer PEN-Nummer zuordnet. POP HL:POP BC bringen die alten Registerinhalte zurück, und DJNZ wird wieder als Endbedingung der Programmschleife verwendet.

## 21. Der Schneider versteht auch fremde Diskettenformate

Schneider und Amstrad machen bei der Diskettenstation DD1-1 und bei den Modellen CPC-664 und CPC-6128, in die ein Laufwerk ja bereits eingebaut ist, Werbung mit der CP/M-Fähigkeit der Geräte. So sollen die Anwender auf das riesige Potential von CP/M-Programmen zurückgreifen können. Was von dieser Aussage zu halten ist, wird im folgenden noch eingehend beleuchtet werden.

Am Schluß werden Sie in der Lage sein, Disketten fremder Computer Ihrem Schneider-CPC "mundgerecht" zu servieren! Der CPC kann dann fremde Diskettenformate lesen und beschreiben.

Zuallererst stehen Sie natürlich vor dem Problem der verschiedenen Diskettengrößen. Der größte Teil der CP/M-Software ist auf 5.25-Zoll- und 8-Zoll-Scheiben lieferbar. Manche Programme gibt es auch im 3.5-Zoll-Format. Aber 3 Zoll? Bisher hat sich außer Amstrad (und damit Schneider) praktisch kein Computer-Hersteller für dieses Format sonderlich begeistern können. Entsprechend mager sieht es auch mit Software in diesem ungewöhnlichen Format aus.

Doch auch mit den Preisen der runden Scheiben können die Schneider-Disketten nicht sonderlich prahlen. Sie kosten immer noch ein Vielfaches ihrer Brüder mit 5 1/4 Zoll. Allein schon aus diesem Grund sollten wirtschaftlich rechnende CPC-Freaks die Anschaffung eines Zweitlaufwerks im 5.25-Zoll-Format zumindest erwägen.

Wenn Sie sich also - aus welchem der angegebenen Gründe auch immer - zur Anschaffung einer 5.25-Zoll-Floppy entschlossen haben, können Sie mit Hilfe unserer abgedruckten Programme Fremdformate lesen. Natürlich bleibt es Ihnen überlassen, sich auch für ein 8-Zoll-Laufwerk zu entscheiden. Falls Sie es tatsächlich tun, dieses anzuschließen, müssen Sie aber auch bedenken, daß die 8-Zoll-Disketten praktisch keinen Vorteil gegenüber den Minifloppies (5.25-Zoll-Disketten nennt man auch so, die 3- und 3.5-Zoll-Disketten heißen folglich "Microdisketten") besitzen.

Die meisten der verfügbaren Zweitlaufwerke arbeiten unter AMSDOS und CP/M und bieten ebenso wie die Erstlingsstation 180 Kilobytes Speicherkapazität.

Das heißt, daß sie einseitig mit doppelter Dichte und 40 Spuren arbeiten.

Das Lesen von 80-Spur-Disketten scheidet damit aus Hardware-Gründen von vornherein aus. Von zweiseitigen Disketten kann aus einsichtigen Gründen nur die erste Seite gelesen werden. Doppelseitige Disketten dürfen also nur bis etwa zur Hälfte beschrieben werden, wenn Sie auf dem Schneider-CPC alle Daten lesen wollen.

Doch nun an die Arbeit: Disketten müssen vom Computer irgendwie "adressiert" werden können. Dazu teilt er sie beim Formatieren in 40 Spuren (Tracks) ein. Jeder dieser Tracks besteht beim Schneider-CPC aus neun Sektoren (Sectors), von denen jeder einzelne 512 Bytes umfaßt. Rechnen wir nach: 512 Bytes \* 9 Sektoren \* 40 Tracks = 184320 Bytes. Diese 184 Kilobytes sind der gesamte frei

verfügbare Speicherplatz auf der Diskette.

"Frei verfügbar" sind sie aber nur für das System, nicht für den Benutzer. Das Betriebssystem braucht ja schließlich noch einige Sektoren für das Inhaltsverzeichnis der Diskette und bei CP/M-Disketten einen System- und einen Boot-Track. Diese beiden Spuren erlauben das Starten von CP/M 2.2.

Somit bleiben Ihnen noch zwischen 169 (System- oder Vendorformat) und 178 Kilobytes (Data-Only-Format) für Ihre eigenen Zwecke. Das IBM-CP/M-86-Format, das der Schneider-CPC ebenso anbietet, soll hier allerdings ausgeklammert werden, da es nicht allzu nutzbringend ist und nur rund 150 Kilobytes an Speicherplatz zuläßt. Dies kommt daher, daß beim IBM-Format nur acht Sektoren jeder Spur formatiert werden. Der neunte Sektor geht verloren.

Die oben gemachten Angaben (40 Tracks, 9 Sektoren, Sektorlänge 512 Bytes) sind aber nur für den Schneider-CPC zutreffend. Andere CP/M-Computer verwenden völlig andere Formate, die somit für Sie nicht ohne weiteres lesbar sind. So ist das Format der Osborne-Z80-Computer so zu charakterisieren: 40 Tracks, 5 Sektoren pro Spur, Sektorlänge 1024 Bytes.

Nachdem es aber für verschiedene CP/M-Computer bereits Programme gibt, die Fremdformate lesen können (etwa für den Osborne-PC mit dem sinnreichen Namen "Osmosis"), muß man das ja auf irgendeine Art auch beim Schneider-CPC durchführen können. Doch dahin ist es noch ein weiter Weg, wie sich zeigen wird.

Fast alle Programme, die wir für diesen Zweck schreiben werden, arbeiten unter AMSDOS. Dies hat zwei einfache Gründe: So können die Programme in Basic mit kurzen Maschinencode-Routinen geschrieben werden, und so ist auch ein wirklich interaktives Arbeiten möglich.

Stellen Sie sich nur vor, Sie müßten ständig Hilfsprogramme unter CP/M assemblieren oder compilieren und linkern!

Glücklicherweise ist der Aufbau der Diskettenverwaltung unter AMSDOS und CP/M recht ähnlich. Größere Unterschiede sind von vornherein ausgeschlossen, da AMSDOS und CP/M beliebig Dateien austauschen können.

Begeben wir uns also als "Forscher" ins Betriebssystem des Schneider-CPC: Das Floppy-Betriebssystem ist als Erweiterungs-ROM in den Adreßbereich von &C000 bis &FFFF parallel zum Bildschirm-RAM und Basic-Interpreter-ROM einblendbar. Da AMSDOS sich auch intern Daten merken muß (zum Beispiel Namen für geöffnete Dateien und Pufferspeicher), reserviert es sich beim Einschalten des Computers den Speicher zwischen &BE40 (dezimal 48704) und &BE7F (48767). Dort findet das Basic-Betriebssystem Ticker-Blöcke zur Interrupt-Steuerung des Floppy-Motors und einige andere Daten. Da dieser Speicher aber bei weitem nicht ausreicht, beansprucht die Floppy 1280 Bytes des normalen RAM-Speichers. Wie das genau funktioniert, können Sie in dem Kapitel (12) nachlesen, in dem wir zeigen, wie sich dieser RAM-Bereich verschieben läßt.

Einen Zeiger auf diesen dynamisch verwalteten Bereich schreibt das AMSDOS in die Adressen &BE7D und &BE7E. Normalerweise ergibt ein Auslesen dieser Adressen den Wert &A700:

```
PRINT HEX$(PEEK(&BE7D)+PEEK(&BE7E)*256)
```

Dieser Bereich ist für unser Ziel schon wesentlich hilfreicher, denn im Speicherbereich &A890 bis &A8A5 (Laufwerk A:) beziehungsweise ab &A8D0 für das Laufwerk B: stehen alle wichtigen Informationen, die ein Diskettenformat beschreiben:

&A890	&A8D0	SPT	128-Byte-Sektoren pro Track (Lowbyte)
&A891	&A8D1	SPT	128-Byte-Sektoren pro Track (Highbyte)
&A892	&A8D2	BSH	Block Shift
&A893	&A8D3	BLM	Block Mask
&A894	&A8D4	EXM	Extent Mask
&A895	&A8D5	DSM	Zahl der Blöcke -1 (Lowbyte)
&A896	&A8D6	DSM	Zahl der Blöcke -1 (Highbyte)
&A897	&A8D7	DRM	Maximale Directory-Einträge -1 (Low)
&A898	&A8D8	DRM	Maximale Directory-Einträge -1 (High)
&A899	&A8D9	AL0	Zahl der Directory-Blöcke (Highbyte)
&A89A	&A8DA	AL1	Zahl der Directory-Blöcke (Lowbyte)
&A89B	&A8DB	CKS	Zahl der geprüften Directory-Einträge
&A89C	&A8DC	CKS	Zahl der geprüften Directory-Einträge
&A89D	&A8DD	OFF	Offset reservierter Systemspuren (Low)
&A89E	&A8DE	OFF	Offset reservierter Systemspuren (High)

Bis hierher sind die Einträge CP/M-kompatibel. Der Speicherblock heißt in der CP/M-Terminologie "DPB". Das steht für "Disk Parameter Block". Die restlichen Bytes sind AMSDOS-spezifische Erweiterungen. Dementsprechend haben wir auch neue Abkürzungen erfunden, die möglichst treffend die Bedeutung dieser Speicherzeilen ausdrücken sollen:

&A89F	&A8DF	FSC	Erster Sektor jedes Tracks
&A8A0	&A8E0	NUM	Zahl der Sektoren pro Track
&A8A1	&A8E1	GRW	Gap-Länge bei Read & Write von Sektoren
&A8A2	&A8E2	GAF	Gap-Länge beim Formatieren von Sektoren
&A8A3	&A8E3	FIL	Fullbyte beim Formatieren von Sektoren
&A8A4	&A8E4	BPS	Bytes pro Sektor (codiert)
&A8A5	&A8E5	RPS	Records pro Sektor

Wahrscheinlich werden Sie mit diesen Namen und Abkürzungen noch nicht allzuviel anfangen können. Wir werden sie aber ausführlich besprechen.

Zuerst zum Begriff "Record": Records sind 128-Byte-Sektoren. Wie bitte? Sie dachten, es gäbe beim Schneider-CPC nur 512-Byte-Sektoren? Das ist schon richtig, 512 Bytes sind die Sektoren lang, die AMSDOS und das CP/M-BIOS (der hardwareabhängige Teil von CP/M) lesen und schreiben können. Doch ein standardisiertes Betriebssystem wie CP/M kann es natürlich nicht zulassen, daß der eine Computer 128, der andere vielleicht 1024 Bytes bei einem Lesebefehl übermittelt. So hat sich Digital Research, die "Erfinderfirma" von CP/M, auf 128 Bytes lange Sektoren festgelegt. Es ist nun die Aufgabe des BIOS (Basic Input/Output System) der einzelnen Computer, die vom Hersteller festgelegten langen Sektoren

in die CP/M-gerechte Darstellung zu verwandeln.

Zur deutlicheren Unterscheidung nennt man die 128-Bytes-Bereiche "logische Sektoren" oder eben "Records". Die hardwareabhängige interne Größe nennt sich "physikalischer Sektor".

Die Speicherstelle SPT gibt an, wieviele Records in einem Track (einer Spur) untergebracht werden können. Das ist eine wirklich sehr einfache Rechnung: 9 Sektoren mit je 512 Bytes in einem Track entsprechend  $9 \times (512/128) = 9 \times 4 = 36$  logischen Sektoren mit 128 Bytes. SPT muß also beim Schneider-Computer den Wert 36 besitzen.

Die Variablen BSH und BLM sind etwas vertrackt, da sie recht kompliziert zu berechnen sind. CP/M und AMSDOS verwenden intern auch noch den Begriff des "Blocks". Ein Block ist ein 1 Kilobyte großer Ausschnitt der Diskette. Der Wert von BSH gibt damit indirekt Auskunft über die Diskettengröße.

BSH gibt an, wieviele Records in einem Block zu finden sind. 1024 (die Blockgröße) geteilt durch 128 (die Recordgröße) ergibt die Zahl 8. BSH ist der 2er-Logarithmus (!) der Zahl der Records im Block. Da 2 hoch 3 die Acht ergibt, steht in BSH eine 3.

Andersherum kann man das auch berechnen:  $\log_2(8) = \log(8)/\log(2) = 3$ .

BLM gibt nun die um eins verminderte Anzahl der Records in einem Block an. Aus 8-1 errechnet sich der BLM-Wert 7.

EXM ("Extent Mask") ist wieder ein Maß für die Diskettengröße. CP/M und AMSDOS unterteilen intern Dateien, die größer als 16 Kilobytes sind, in einzelne Dateien, die allerdings nicht im Directory (Inhaltsverzeichnis der Diskette) erscheinen, da sie speziell markiert sind. Der reine Anwender merkt also davon nichts, sehr wohl aber alle, die tiefer ins System einsteigen wollen.

Die Aufteilung in 16K-Blöcke, sogenannte Extents, ist notwendig, da im 32 Bytes großen Verzeichniseintrag im Directory, der für jede Datei angelegt wird, nicht genügend Platz ist, die von einer Datei belegten Sektoren zu markieren. Da die Schneider-Disketten zu den kleineren Magnetplatten zu zählen sind (denken Sie etwa auch an Festplatten mit mehreren Megabytes!) und gleichzeitig die Blockgröße 1 Kilobyte ist, betrifft Sie EXM nicht. Der Computer trägt hier den Wert 0 ein.

DSM zeigt dem Betriebssystem die Anzahl der freien Blöcke einer Diskette zuzüglich der Blöcke, die für das Diskettendirectory Verwendung finden. Beim CP/M-Systemformat der CPC-Reihe gilt der Wert 171: 169 Kilobytes plus vierundsechzig 32-Byte-Einträge ergeben 169 plus 2 Blöcke, eben 171. Der Wert von DSM ist konstant um eins geringer als die berechnete Zahl, da der Computer die Zählung der Blöcke bei Null beginnt: DSM ist 170.

DRM sagt dem Computer, wieviele Einträge im Disketten-Directory untergebracht werden können. Die Amstrad-Programmierer waren in dieser Hinsicht recht knickrig und haben der Diskette nur 64 Einträge spendiert. Andere Computer bieten meist 128, manche sogar



256 Einträge. Für DRM ist dieser Wert 64 wieder um eins zu vermindern, somit ist DRM 63.

AL0 und AL1 gehören zusammen. Sie bestimmen, wieviele Blöcke der Diskette für das Inhaltsverzeichnis freigehalten werden müssen. Dazu sind die beiden Werte binär codiert. Der Schneider verwendet AL0=&C0 und AL1=&00. Im Binärsystem kann man das so darstellen:

AL0= bin 11000000      AL1= bin 00000000

Damit sind die ersten beiden der verfügbaren Diskettenblöcke für das Directory reserviert und können nicht von Daten überschrieben werden. Sollten es etwa die ersten vier Blöcke sein, müßten AL0 und AL1 folgende Werte annehmen:

AL0= bin 11110000      AL1= bin 00000000

Hexadezimal entspräche das &F0 und &00.

CKS gibt die Zahl der Directory-Einträge an, die bei jedem Diskettenzugriff geprüft werden. Wechselt nämlich der Benutzer einfach die Diskette im Laufwerk, ohne dies dem Betriebssystem mitzuteilen, könnte es passieren, daß auf der alten Diskette eine Datei noch geöffnet ist und auf der neuen Diskette Daten in eine nicht-existente Datei geschrieben würden. Also muß das Betriebssystem irgendwie einen Diskettenwechsel erkennen. Dazu prüft es bei jedem Zugriff auf die Diskette die in CKS angegebenen Einträge im Directory.

Entdeckt der Computer einen Unterschied zwischen dem alten und dem neuen Directory, hat ein Diskettenwechsel stattgefunden, und das Betriebssystem kann darauf reagieren. CP/M gibt beim nächsten Schreibzugriff auf die gewechselte Diskette einen BDOS-Error aus, AMSDOS meldet "Drive x: Disc changed, closing file". Natürlich ist dieser Schutzmechanismus nicht hundertprozentig sicher. Sind nämlich alle Directory-Einträge der beiden Disketten identisch, kann der Computer einen Wechsel nicht erkennen.

Dieser Fall dürfte aber nur sehr selten eintreten. Je größer der Wert für CKS gewählt wird, desto sicherer wird das Verfahren, weil eine größere Zahl von Einträgen geprüft wird. Allerdings dauert das auch entsprechend länger, und alle Diskettenoperationen würden bei einem zu großen Wert spürbar gebremst.

OFF bestimmt die Zahl der vom Betriebssystem belegten Spuren. Beim CP/M-Systemformat benutzt CP/M 2.2 die ersten beiden Tracks, indem es dort das gesamte Betriebssystem ablegt. Somit ist dort OFF=2. Die Data-Only-Disketten bieten aus eben diesem Grund mehr Speicherplatz, denn sie besitzen keinen reservierten Platz für CP/M. Somit ist dort OFF=0.

Die übrigen Eintragungen sind Schneider-spezifisch und nicht von CP/M vorgegeben:

FSC repräsentiert die Nummer des ersten Sektors jedes Tracks. Es ist nämlich keineswegs so, daß die Zählung immer bei 0 oder 1 beginnen müßte. Anhand dieser Sektornummern unterscheidet das

AMSDOS-System sogar die verschiedenen zulässigen Diskettenformate! Das CP/M-Systemformat numeriert die Sektoren von &41 bis &49 durch, das Data-Only-Format von &C1 bis &C9 und das IBM-CP/M-86-Format von &01 bis &08. Also stehen je nach Format in FSC die Zahlen &41, &C1 oder &01.

Durch NUM wird angegeben, wieviele physikalischen Sektoren in einem Track zu finden sind. Bis auf das IBM-Format (NUM=8) gilt der Wert 9.

GRW und GAF sind Daten, die vom FDC (Floppy Disc Controller) abhängig sind.

Der FDC ist der IC-Baustein, der die Floppy-Stationen ansteuert, von ihnen Daten liest, an sie Daten schreibt und auch Tracks formatiert.

Gaps sind "Lücken", die beim Lesen, Schreiben und Formatieren zwischen den einzelnen Sektoren und Spuren freigelassen werden, um Toleranzen der Fertigung der Station und der Diskettenscheiben auszugleichen. Eine Änderung der Werte dürfte kaum Sinn machen.

FIL ist das Byte, das beim Formatieren einer Diskette in die neu entstandenen Sektoren geschrieben wird. CP/M- und CPC-typisch ist der Wert &E5 (229).

BPS ("Bytes per Sector") ist eine codierte Information darüber, wieviele Bytes in einem Sektor zu finden sind. BPS=2 entspricht den 512 Bytes eines Sektors der Schneider-Diskettenstationen.

RPS legt fest, wieviele Records in einem Sektor untergebracht werden können. RPS ist 512/128 Bytes und damit 4.

Da nun einmal alle Theorie "grau" ist, werden wir diese Informationen an zwei Programmbeispielen demonstrieren. Das Programm DPBSHOW.BAS läuft unter AMSDOS und ist in Basic geschrieben. DPBSHOW.COM und der zugehörige Quellcode DPBSHOW.ASM sind ein kleines Maschinenprogramm für CP/M.

DPBSHOW.BAS zeigt den Disk Parameter Block zusammen mit den AMSDOS-typischen Erweiterungen in hexadezimaler Schreibung auf dem Bildschirm an. Es wurde der DPB von Laufwerk B: gewählt, da so wieso nur dieses eine 5.25-Zoll-Floppy sein kann. DPBSHOW.BAS gibt die folgenden Ausgaben von sich:

Disk Parameter Block  
-----

```
SPT= &0024
BSH= &0003
BLM= &0007
EXM= &0000
DSM= &00AA
DRM= &003F
AL0= &00C0
AL1= &0000
CKS= &0010
```

```

OFF= &0002
FSC= &0041
NUM= &0009
GRW= &002A
GAF= &0052
FIL= &00E5
BPS= &0002
RPS= &0004

```

Der Einfachheit halber macht das Programm keinen Unterschied zwischen Einbyte- und Zweibyte-Speicherstellen.

Sie können diese aber ganz leicht an den vorhin gegebenen Informationen differenzieren.

Natürlich vergessen wir auch das Listing von DPBSHOW.BAS nicht.

## 21.1 Listing: DPBSHOW.BAS

```

100 ' *****
110 ' *
120 ' * DPBSHOW.BAS: Zeigt den DPB-Block von Laufwerk B: an *
130 ' *
140 ' *****
150 '
160 MODE 1:INK 1,0:INK 0,13:BORDER 10:PEN 1:PAPER 0:WINDOW 12,39,2,24
170 dpb.a=PEEK(&BE42)+PEEK(&BE43)*256 ' Start des DPB von Laufwerk A:
180 base=dpb.a+64 ' DPB fuer Laufwerk B: hat Offset von 64 Bytes
190 PRINT "Disk Parameter Block":PRINT STRING$(20,45):PRINT
200 PRINT "SPT= &";HEX$(PEEK(base)+PEEK(base+1)*256,4)
210 PRINT "BSH= &";HEX$(PEEK(base+2),4)
220 PRINT "BLM= &";HEX$(PEEK(base+3),4)
230 PRINT "EXM= &";HEX$(PEEK(base+4),4)
240 PRINT "DSM= &";HEX$(PEEK(base+5)+PEEK(base+6)*256,4)
250 PRINT "DRM= &";HEX$(PEEK(base+7)+PEEK(base+8)*256,4)
260 PRINT "AL0= &";HEX$(PEEK(base+9),4)
270 PRINT "AL1= &";HEX$(PEEK(base+A),4)
280 PRINT "CKS= &";HEX$(PEEK(base+B)+PEEK(base+C)*256,4)
290 PRINT "OFF= &";HEX$(PEEK(base+D)+PEEK(base+E)*256,4)
300 PRINT "FSC= &";HEX$(PEEK(base+F),4)
310 PRINT "NUM= &";HEX$(PEEK(base+10),4)
320 PRINT "GRW= &";HEX$(PEEK(base+11),4)
330 PRINT "GAF= &";HEX$(PEEK(base+12),4)
340 PRINT "FIL= &";HEX$(PEEK(base+13),4)
350 PRINT "BPS= &";HEX$(PEEK(base+14),4)
360 PRINT "RPS= &";HEX$(PEEK(base+15),4)
370 PRINT:END

```

Wäre noch die Funktionsweise von DPBSHOW.BAS zu betrachten: In den Adressen &BE42 und &BE43 legt das Floppy-Betriebssystem einen Zeiger auf den Disk Parameter Block des Laufwerks A: an. Natürlich können Sie die Adresse auch durch Auslesen der Speicherstellen &BE7D und &BE7E erhalten. Nach der Addition des Werts 400 ergibt das dieselbe Adresse wie die, die in &BE42 steht. 64 Bytes hinter dem DPB von A: beginnt der DPB für die Zweitfloppy. Durch

eine Vielzahl von PRINT-HEX\$-Befehlen gibt der Computer den Inhalt dieses DPB-Blocks aus.

Das CP/M-Programm DPBSHOW.COM arbeitet ganz ähnlich, hat aber den entscheidenden Vorteil, daß Sie es auch auf anderen CP/M-Rechnern starten können und anhand der ausgedruckten Informationen Ihr Schneider-Laufwerk auf deren Fremddisketten einstellen können. Die Bildschirmausgabe ist etwas anders aufgebaut, die gegebenen Informationen sind aber identisch:

A>DPBSHOW

Disk Parameter Block

-----

SPT lo = &24

SPT hi = &00

BSH = &03

BLM = &07

EXM = &00

DSM lo = &AA

DSM hi = &00

DRM lo = &3F

DRM hi = &00

AL0 = &C0

AL1 = &00

CKS lo = &10

CKS hi = &00

OFF lo = &02

OFF hi = &00

Das CP/M-Pendant gibt also jedes einzelne Byte aus; Speicherstellen, die aus zwei Bytes bestehen, werden in "lo" für das Lowbyte und "hi" für das Highbyte aufgesplittet.

Bevor Sie aber das Programm nutzen können, müssen Sie es erst einmal eintippen.

Das ist für alle, die nicht zu den "CP/M-Profis" gehören, nicht so ganz einfach.

Verwenden Sie am besten einen Editor, zum Beispiel WordStar im N-Modus ("Bearbeiten einer Programmdatei"). Der Name der zu bearbeitenden Datei muß mit ".ASM" enden.

Empfehlenswert ist "DPBSHOW.ASM". In diesen Editor geben Sie den Quellcode des CP/M-Programms ein.

Die Kommentare können Sie selbstverständlich weglassen:

## 21.2 Assemblerlisting: DPBSHOW.COM

```
*****
*
*      DPBSHOW.COM zeigt den DPB-Block unter CP/M an
*
*****
```

```

*
*   Deklarationen und Equates
*   -----
*
WARM    EQU    0           ; Warmstart
BDOS     EQU    5          ; BDOS-Einsprung
CONOUT   EQU    2          ; Konsolenausgabe
STRROUT  EQU    9          ; String ausgeben
GETDPB   EQU    31         ; Hole DPB-Zeiger nach HL
TPA      EQU    0100H      ; Beginn der TPA

*
*   Hauptprogramm
*   -----
*
        ORG     TPA

START    LXI     SP,STACK   ; Stackpointer laden

MESSAGE  MVI     C,STRROUT  ; Titelmeldung ausgeben
        LXI     D,TMESSAGE  ; ueber die BDOS-Funktion
        CALL    BDOS        ; 9 "STRROUT"

ADDR     MVI     C,GETDPB   ; Zeiger auf DPB-Tabelle ins
        CALL    BDOS        ; HL-Register holen
        LXI     D,NAMES     ; Zeiger auf Namenskuerzel
        MVI     B,15        ; 15 Daten auszugeben

LOOP     PUSH    B          ; BC-Register sichern
        PUSH    D          ; DE-Register sichern
        PUSH    H          ; HL-Register sichern
        MVI     C,STRROUT  ; Stringausgabe-Funktion
        CALL    BDOS        ; Aufrufen
        POP     H          ; HL-Register wiederherstellen
        POP     D          ; DE-Register wiederherstellen
        POP     B          ; BC-Register wiederherstellen
        PUSH    H          ; HL-Register sichern
        LXI     H,15        ; 15 Bytes ist jede Meldung lang
        DAD     D          ; Zum Zeiger addieren
        XCHG     ; ins DE-Register bringen
        POP     H          ; HL-Register wiederherstellen

        CALL    HEXBYTE    ; Hexbyte an (HL) ausgeben
        INC     H          ; Zeiger erhoehen

LASTLP   DCR     B          ; Schleifenzaehler vermindern
        JNZ     LOOP       ; Weitermachen, bis B=0

LAST     LXI     D,CRLF     ; Wagenruecklauf & Zeilenvorschub
        MVI     C,STRROUT  ; Stringausgabe ueber BDOS-Funktion
        CALL    BDOS        ; 9 "STRROUT"
        JMP     WARM        ; Danach Warmstart

*
*   Ausgabe eines Hexbytes (HL)

```

```

* -----
*
HEXBYTE MOV     A,M           ; Byte (HL) lesen
        CALL    SHIFT        ; Nach rechts shiften und ausgeben
        MOV     A,M           ; Byte (HL) lesen
        CALL    PRINT         ; Unveraendert ausgeben
        RET              ; Ruecksprung zum Hauptprogramm

SHIFT   RAR              ; Viermal nach rechts shiften.
        RAR              ; So wird das rechte Nibble
        RAR              ; ausgeblendet und das linke
        RAR              ; Nibble wird zum rechten.

PRINT   ANI     0FH         ; Highnibble ausblenden
        ADI     48          ; Aus 0 wird '0', aus 1 '1' etc.
        CPI     58          ; Groesser als '9'?
        JC      SHOW        ; Wenn nicht, unveraendert ausgeben
        ADI     7            ; Sonst 'A'-'9' addieren

SHOW    PUSH     B           ; BC-Register sichern
        PUSH     D           ; DE-Register sichern
        PUSH     H           ; HL-Register sichern
        MOV     E,A          ; Zeichen ins E-Register
        MVI     C,CONOUT     ; BDOS-Funktion 2 "Konsolenausgabe"
        CALL    BDOS         ; System aufrufen
        POP      H           ; HL-Register wiederherstellen
        POP      D           ; DE-Register wiederherstellen
        POP      B           ; BC-Register wiederherstellen
        RET              ; Ruecksprung zum Hauptprogramm

```

```

* -----
*      Titelmeldung beim Programmstart
* -----
*

```

```

MESSAGE DB      13,10,10
        DB      'Disk Parameter Block'
        DB      13,10
        DB      '-----'
        DB      13,10,10,'$'

```

```

* -----
*      Disk-Parameter-Namen
* -----
*

```

```

NAMES  DB      13,10,'SPT lo  = &$'
        DB      13,10,'SPT hi  = &$'
        DB      13,10,'BSH    = &$'
        DB      13,10,'BLM    = &$'
        DB      13,10,'EXM    = &$'
        DB      13,10,'DSM lo  = &$'
        DB      13,10,'DSM hi  = &$'
        DB      13,10,'DRM lo  = &$'
        DB      13,10,'DRM hi  = &$'
        DB      13,10,'AL0    = &$'

```

```

DB      13,10,'AL1'    =  &$'
DB      13,10,'CKS lo' =  &$'
DB      13,10,'CKS hi' =  &$'
DB      13,10,'OFF lo' =  &$'
DB      13,10,'OFF hi' =  &$'

CRLF    DB      13,10,10,'$'

        DS      20                      ; Programmstack
STACK   EQU     $
        END

```

Nachdem Sie das "END" in der letzten Zeile eingetippt haben, drücken Sie bitte nochmals die RETURN- beziehungsweise ENTER-Taste, so daß der Cursor in einer leeren Zeile nach dem letzten Befehl steht. Dies ist sehr wichtig, da sonst der Assembler während der Übersetzung "aussteigt"!

Wenn Sie den Text gespeichert haben (mit WordStar verwenden Sie dazu Control-KX), benötigen Sie ASM.COM und LOAD.COM von Ihrer CP/M-Systemdiskette. Gesetzt den Fall, daß sich ASM, LOAD und der Quellcode auf einer Diskette im Drive A: befinden und genug Platz für die entstehenden Dateien ist, tippen Sie:

A>ASM DPBSHOW.AAA

Das erste "A" der Extension des Dateinamens bestimmt das Laufwerk, auf dem die Quelldatei steht. Das folgende "A" legt das Ziellaufwerk der entstehenden Hexadezimal-Datei fest. ASM.COM erzeugt nämlich keine COM-Kommandodateien, sondern eine Zwischendatei mit der Extension ".HEX". Erst LOAD.COM macht daraus ein COM-File. Das dritte "A" sagt dem Assembler, daß die Listdatei auf das Laufwerk A: geschrieben wird. Diese Datei erhält automatisch den Namen "DPBSHOW.PRN" für "Print File". Auf der Diskette befinden sich damit die Programme DPBSHOW.ASM (der von Ihnen eingegebene Quellcode), DPBSHOW.HEX (die Intel-Hexdatei, so heißt diese Codierung) und DPBSHOW.PRN (die Listdatei). Geben Sie nun ein:

A>LOAD DPBSHOW

Aus der ".HEX"-Datei entsteht jetzt "DPBSHOW.COM". Wenn keines der beiden Programme einen Fehler bei der Übersetzung gemeldet hat, können Sie DPBSHOW.COM jetzt starten:

A>DPBSHOW

Es erscheint die oben angegebene Bildschirmmeldung. Auch hier wollen wir ausführlich auf die Funktionsweise des Programms eingehen:

DPBSHOW wird mit ORG TPA an die Adresse &0100 geladen. Der CP/M-Assembler versteht Hexadezimalzahlen aber nur, wenn sie zum Beispiel als 0100H geschrieben werden. Da CP/M auch mit den Prozessoren der Familie Intel 8080 und 8085 arbeitet, verwendet der Assembler ASM.COM deren Mnemonics. Diese unterscheiden sich doch

ziemlich von den Z80-Namen, weswegen wir im folgenden zusätzlich die Z80-Mnemonics angeben werden.

Am Label START wird mit LXI SP,STACK (LD SP,STACK) der Stackpointer auf einen privaten Prozessorstapel ausgerichtet.

Der Programmcode ab MESSAGE gibt die Titelmeldung "Disk Parameter Table" samt Unterstreichung aus. Dazu kann die BDOS-Funktion 9 ("String ausgeben") benutzt werden. Zu diesem Zweck lädt man das C-Register mit dem Wert 9 (MVI C,STROUT bzw. LD C,STROUT) und das DE-Doppelregister mit einem Zeiger auf den zu druckenden String. Das geht mit LXI D,TMESSAGE (LD DE,TMESSAGE).

Das Ende der Zeichenkette zeigt man dem Betriebssystem durch ein Dollarzeichen an. Eine Define-Byte-Zeile ließe sich so eintippen:

```
TEXT DB 'Dies ist eine Meldung$'
```

Durch CALL BDOS springt der Prozessor in die BDOS-Routine, in der die gewünschte Aktion ausgeführt wird.

BDOS repräsentiert die Adresse 5. In &0005 steht nämlich bei allen CP/M-Versionen der BDOS-Einsprung.

Ab der Marke ADDR holt sich der Computer mit der BDOS-Funktion 31 einen Zeiger auf den Disk-Parameter-Block ins HL-Register. Dazu findet der Code MVI C,GETDPB (LD C,GETDPB) und CALL BDOS Verwendung. Das DE-Register zeigt auf die Tabelle mit den Namenskurzeln wie SPT, DSM, ALØ etc. Da 15 Daten auszugeben sind, wird das B-Register mit diesem Wert geladen.

LOOP ist die Hauptschleife, in der die kurzen Texte und die Werte im DPB-Block ausgegeben werden. Da die BDOS-Routinen alle Registerinhalte zerstören können, sollten die Register mit PUSH auf den Stack geschoben werden. Der STROUT-Aufruf gibt ein Namenskurzel aus.

Mit

```
PUSH H      (PUSH HL)
LXI H,15    (LD HL,15)
DAD D       (ADD HL,DE)
XCHG        (EX DE,HL)
POP H       (POP HL)
```

sorgt der Computer dafür, daß der Inhalt des DE-Registers um 15 erhöht wird. Somit zeigt DE dann auf den nächsten Eintrag in der Namenstabelle. Mit dem Aufruf CALL HEXBYTE wird der Inhalt der Speicherstelle, auf die HL zeigt, hexadezimal ausgegeben. INX H (INC HL) erhöht den Zeiger auf das nächste Byte im Disk-Parameter-Block.

LASTLP vermindert den Schleifenzähler im B-Register um eins und springt wieder nach LOOP, wenn B ungleich 0 ist.

LAST gibt mit Unterstützung der BDOS-Funktion 9 einen Wagenrücklauf und einen Zeilenvorschub aus, worauf noch ein Warmstart ausgelöst wird.



Das Unterprogramm HEXBYTE ist nicht nur in DPBSHOW von Nutzen, sondern kann ganz allgemein in anderen Programmen auch Verwendung finden. Es gibt das Byte aus, auf dessen Adresse das HL-Register zeigt. Stattdessen können Sie auch einen Registerinhalt ausgeben lassen. Dazu müssen Sie nur die beiden MOV A,M durch MOV A,reg ersetzen, zum Beispiel MOV A,B oder MOV A,L.

Zur Funktionsweise von HEXBYTE: Mit MOV A,M wird das Byte in den Z80-Akkumulator geladen. Das Programm springt dann zur Marke SHIFT. Dort schiebt das Programm die vier linksstehenden Bits in die rechtsstehenden. Durch ANI 15 wird sichergestellt, daß nur das rechtsstehende Nibble des Bytes stehenbleibt. ADI 48 macht aus dem internen Binärkode des Computers ein ASCII-Zeichen. So ist ja der ASCII-Code von "0" 48. Andersherum: CHR\$(0+48)="0". Wenn das Zeichen kleiner oder gleich "9" und damit eine Ziffer ist, wird es von SHOW direkt ausgegeben. Sonst addiert der Computer noch die Zahl 7 und verarbeitet so die Buchstabencodes im Hexadezimalsystem, nämlich A, B, C, D, E und F. In einem zweiten Durchgang wird wieder der Akku mit dem Byte geladen. Die RRA-Befehle werden übersprungen. So bleibt das rechte Nibble an seinem Platz und ANI bewirkt nur, daß das linke Nibble gelöscht wird. Ab hier funktioniert alles wie bereits erklärt.

In den übrigen Programmzeilen stehen Texte und ein Define-Space (DS, bei anderen Assemblern DEFS oder RMEM), der Speicherplatz für den Systemstack reserviert.

Mit diesen beiden Programmen können Sie sich die DPB-Blöcke ausgeben lassen. Umgekehrt ist es auch möglich, in die DPBs neue Daten zu POKEn.

Wenn Sie die "richtigen" Daten einsetzen, versteht der Schneider-Computer auch Diskettenformate anderer Computer.

Wir beschränken uns hier auf CP/M-Computer, da zum Beispiel Commodore und Apple ein völlig anderes Aufzeichnungsverfahren verwenden, das sich ohne Hardwareänderungen nicht simulieren läßt. Für Fachleute: Die Disk-Controller beider Firmen benutzen GCR (Group Code Recording) im Gegensatz zum heute weit verbreiteten MF- und MFM-System.

Auch Disketten anderer Betriebssysteme, etwa MS-DOS oder UNIX, werden wir nicht lesen können, ganz einfach, weil die Daten dort anders organisiert sind. Wer schreibt beispielsweise vor, daß das Directory immer am Anfang der Diskette stehen muß? Doch es gibt noch genügend CP/M-Computer, deren Formate man übersetzen kann. Wir zeigen diese Methode an Disketten des Triumph-Adler Alphatronic-PC, des Bondwell-12/14 und des Osborne-Double-Density-Formats.

Der erste Schritt wird darin bestehen, diese Disketten sektorweise zu lesen. Wir verwenden dazu unsere Kenntnisse, die wir uns in vorangegangenen Kapiteln angeeignet haben.

Das Programm SREAD.ASM stellt eine sehr einfach gehaltene Routine zum Sektorlesen dar. "Einfach" deshalb, weil alle Daten von Basic durch direktes EinPOKEN Übergeben werden müssen:

## 21.3 Assemblerlisting: SREAD.ASM

```

; *****
; *
; *   SREAD.ASM -
; *
; *   Sektorweises Lesen von Disketten in Laufwerk B
; *
; *   *****
; *****

      ORG      &A000

SREAD  LD      E,1      ; Laufwerk B=1
      LD      D,0      ; Track
      LD      C,1      ; Sektor
      LD      HL,BUFFER ; Zeiger auf Sektorpuffer
      RST     3         ; RST3 in den Floppy-ROM
      DEFW    RSX       ; zur gewünschten RSX
      RET

RSX    DEFW    &C03C     ; ROM-Adresse READ SECTOR
      DEFB     &07      ; ROM-Auswahladresse

BUFFER DEFS    256      ; 256 Bytes Pufferspeicher

      END
; -----

```

Das Programm startet bei &A000, kann aber (falls es tatsächlich notwendig ist) durch Änderung der ORG-Direktive im Speicher verschoben werden. Die RSX "Read Sector" erwartet im E-Register die Laufwerksnummer. "1" steht für Drive B:, "0" würde A: repräsentieren. Im D-Register hat die Track-(Spur-)Nummer zu stehen.

Das C-Register enthält die Sektornummer und HL sollte auf einen Speicherbereich zeigen, in dem der Sektor abgelegt wird. Durch einen RST3 springt der Computer die RSX-Routine im AMSDOS/Floppy-ROM an. RET besorgt den Rücksprung nach Basic. Wenn Sie das Programm für die Adresse &A000 assemblieren, erhalten Sie folgende Speicheradressen:

Laufwerk	&A001
Spur	&A003
Sektor	&A005
Zeiger auf Pufferspeicher	&A007
Pufferspeicher	ab &A010

Da die Benutzung des Maschinenprogramms allein recht unkomfortabel wäre, schreiben wir uns ein Basic-Programm, das alle Sektoren aller Tracks einer Diskette einliest und auf dem Bildschirm anzeigt. Es arbeitet in der vorliegenden Version mit BONDWELL-Disketten, also nennen wir es SECREAD.BON.

## 21.4 Listing: SECREAD.BON ein Basicprogramm!

```

100 ' *****
110 ' *
120 ' *                               SECREAD.BON
130 ' *
140 ' *   Auslesen von Disk-Sektoren am Beispiel des Bondwell-Formats
150 ' *
160 ' *****
170 '
180 'B:MODE 2:INK 1,0:INK 0,13:BORDER 10
190 ' Berechnung der Startadresse vom DPB f)r das Laufwerk B: *****
200 dpb.drive.a=PEEK(&BE42)+PEEK(&BE43)*256
210 dpb.drive.b=dpb.drive.a+64
220 base=dpb.drive.b
230 ' Benennung der Bytes im DPB *****
240 SPT=base
250 BSH=base+2
260 BLM=base+3
270 EXM=base+4
280 DSM=base+5
290 DRM=base+7
300 AL0=base+9
310 AL1=base+&A
320 CKS=base+&B
330 OFF=base+&D
340 FSC=base+&F
350 NUM=base+&10
360 GRW=base+&11
370 GAF=base+&12
380 FIL=base+&13
390 BPS=base+&14
400 RPS=base+&15
410 ' Einlesen der Daten f)r das Bondwell-12-Format *****
420 POKE SPT,36: POKE SPT+1,0
430 POKE BSH,4
440 POKE BLM,15
450 POKE EXM,1
460 POKE DSM,84: POKE DSM+1,0
470 POKE DRM,127:POKE DRM+1,0
480 POKE AL0,&C0:POKE AL1,0
490 POKE CKS,32: POKE CKS+1,0
500 POKE OFF,2: POKE OFF+1,0
510 POKE FSC,1
520 POKE NUM,17
530 POKE GRW,&2A
540 POKE GAF,&52
550 POKE FIL,&E5
560 POKE BPS,1
570 POKE RPS,2
580 ' Maschinencode zum Lesen von Sektoren einPOKE n *****
590 MEMORY &9FFF
600 FOR i=&A000 TO &A00F:READ a:POKE i,a:NEXT i
610 DATA &1E,&01,&16,&00,&0E,&01,&21,&10,&A0,&DF,&0D,&A0,&C9,&3C,&C0,&07
620 ' Schleife zum Auslesen aller Sektoren einer Diskette *****
630 FOR trk=0 TO 39
640     POKE &A003,trk

```

```

650     FOR sec=0 TO 17
660         POKE &A005,sec
670         PRINT "Track";trk;"Sektor";sec;
680         PRINT STRING$(81-POS(#0),45)
690         CALL &A000
700         FOR i=&A010 TO &A10F
710             PRINT CHR$(1);CHR$(PEEK(i));
720         NEXT i
730         PRINT:PRINT
740     NEXT sec
750     PRINT:PRINT:PRINT:PRINT
760 NEXT trk
770 END
780 ' *****

```

SECREAD.BON berechnet auf bekannte Art die Adresse des DPB-Blocks für das B-Laufwerk. In den Zeilen 240 bis 400 werden diversen Variablen die Adressen der Einzelbytes des DPB zugewiesen.

Von 420 bis 570 POKet das Programm in den DPB die Werte, die für das Bondwell-Format typisch sind. Natürlich ließe sich das Programm mit einer FOR-NEXT-Schleife wesentlich einfacher und kürzer schreiben.

Es hätte aber den Nachteil, daß Sie nicht im Direktmodus mit Hilfe der Namen auf den DPB zugreifen könnten, sondern erst herumrechnen müßten. Mit der gewählten Art der Programmierung läßt sich nach Ablauf des Programms etwa eingeben:

```

PRINT PEEK(SPT)
X=PEEK(RPS)
PRINT PEEK(DSM)+PEEK(DSM+1)*256

```

In 590 bis 610 findet sich der Maschinencode von SREAD.ASM wieder. Mit den Befehlen in den Zeilen 620 bis 770 werden alle Sektoren der Diskette in den Speicher eingelesen und auf dem Bildschirm ausgegeben.

Das Bondwell-12-Format läßt sich so charakterisieren: Sektorlänge 256 Bytes, 171 Kilobytes Speicherkapazität, 18 Sektoren in jedem Track, die Diskette faßt 40 Tracks. Die Sektoren sind von 0 bis 17 numeriert. Die Diskette wird einseitig beschrieben, das Directory kann 128 Einträge verwalten. Diese Beschreibungen werden durch das Programm in maschinenlesbare Form gebracht.

Je nachdem, was sich auf der Diskette befindet, werden Daten, Programme oder lauter "Spielkarten"-Symbole ausgegeben.

Das Zeichen &E5 (229) wird ja verwendet, um leere Sektoren zu füllen.

Damit sehen wir, daß es technisch möglich ist, fremde Diskettenformate zu lesen. Falls Sie keine Bondwell-Diskette haben, wie wäre es mit einer im Format des TA-Alphatronic?

Das Programm SECREAD.TAL kann zu SECREAD.BON hinzugeMERGET werden und verarbeitet dann das neue Format. Da es beim MERGEN von Programmen zum Fehler "EOF met" kommen kann, sollten Sie das Programm im ASCII-Format SAVEN:

SAVE "SECREAD.TAL",A

So sehen die Ergänzungen und Anpassungen für das TA-Format aus:

```
410 ' SECREAD.TAL: Anpassung für Disketten im TA-Alphatronic-Format *****
420 POKE SPT,64:POKE SPT+1,0
430 POKE BSH,4
440 POKE BLM,&F
450 POKE EXM,1
460 POKE DSM,150:POKE DSM+1,0
470 POKE DRM,63:POKE DRM+1,0
480 POKE AL0,&80:POKE AL1,0
490 POKE CKS,16:POKE CKS+1,0
500 POKE OFF,2:POKE OFF+1,0
510 POKE FSC,1
520 POKE NUM,16
530 POKE GRW,&2A
540 POKE GAF,&52
550 POKE FIL,&E5
560 POKE BPS,1
570 POKE RPS,2
650      FOR sec=1 TO 16
```

Das Alphatronic-Format ist völlig anders aufgebaut als das Bondwell-Format. SECREAD.TAL greift auf diese Daten zurück:

```
SPT=64
BSH=4
BLM=15
EXM=1
DSM=150
DRM=63
AL0=&80
AL1=&00
CKS=16
OFF=2
```

Der Alphatronic-PC beschreibt seine Disketten auf beiden Seiten. Abhängig von der Schneider-Hardware können Sie Disketten nur einseitig lesen. Die zweite Seite ist also für Sie verloren. Wenn die Disketten auf dem Alphatronic-PC nur bis etwa zur Hälfte beschrieben werden, können Sie auf alle Daten zugreifen, weil zuerst die erste Diskettenseite gefüllt wird, bevor mit dem Beschreiben der zweiten Seite begonnen wird.

Wenn Sie selbst Fremdformate installieren wollen, benötigen Sie auch eine solche Tabelle. Am einfachsten lässt sich diese erstellen, wenn Sie das DPBSHOW.COM-Programm auf dem Rechner laufen lassen, dessen Diskformat Sie benötigen.

Sonst können Sie versuchen, anhand der Diskettengröße und der

Angaben von STAT DSK: hinter das Format zu kommen. Das ist allerdings ein recht mühsamer Weg. Ein Tip: Es ist keineswegs so, daß Sie von Anfang an wirklich alle Daten wissen müssen. Oft können Sie schon mit drei oder vier Werten die Diskette "notdürftig" lesen. Das beste Beispiel dafür ist SECREAD.TAL. Denn die Daten, die oben angegeben sind, stimmen eigentlich gar nicht. Vielmehr ist das Alphatronic-Format exakt so definiert:

```
SPT=64
BSH=4
BLM=15
EXM=1
DSM=151
DRM=127
AL0=&C0
AL1=&00
CKS=32
OFF=2
FSC=1
NUM=16
GRW=&2A
GAF=&52
FIL=&E5
BPS=1
RPS=2
```

Trotzdem erkennt der Schneider-CPC das Triumph-Format auch mit den falschen Daten an!

Sektorweise können wir die Disketten also nun lesen. Doch was passiert, wenn Sie etwa DIR, CAT oder OPENIN eingeben? Das Laufwerk rattert gar fürchterlich, und nach einiger Zeit erscheint ein "Read fail - Retry, Ignore, Cancel". Brechen Sie um Gottes willen (oder besser: um Ihrer Floppy willen) mit "C" die Leseversuche ab! Das Problem rührt einfach daher, daß AMSDOS bei jedem Diskettenzugriff ein "Log-In", ein Einloggen der Diskette durchführt und dabei immer auf die bereits vorhandenen Standardformate zurückgreift. Somit steht nach der Read-fail-Meldung immer wieder einer der alten DPB-Inhalte im Block.

Doch die AMSDOS-Programmierer haben diese mißliche Lage anscheinend schon vorhergesehen. Sie spendierten dem Betriebssystem zwei Speicherstellen, die als Flags dienen. Werden diese Speicherstellen auf einen Wert ungleich Null gesetzt, unterdrückt der Computer dieses "Log-In" und erkennt das fremde Diskettenformat an.

Diese Speicherstellen liegen 24 Bytes hinter dem Anfang des jeweiligen DPB-Blocks.

Unter AMSDOS startet der DPB normalerweise für das Laufwerk A: bei &A890, derjenige für B: bei &A8D0. Damit liegen die Flags bei &A8A0 und &A8E0. Durch POKE &A8E0,255 verhindert man damit das Log-In der Diskette. Allerdings hat das den Nachteil, daß auch die Schneider-eigenen Disketten nicht mehr eingeloggt werden. Da das fremde Diskettenformat aber sowieso nur für die Zeit der Übertragung der Daten und Programme auf das Laufwerk A: benutzt

werden dürfte und man danach die Speicherstelle zurücksetzen kann, sollten daraus eigentlich keine Probleme entstehen.

Das Programm FORMAT.TAL zeigt alle bisher besprochenen Programmier-techniken und verwandelt das Laufwerk B: in eine Triumph-Adler-Floppy. Nach all den bisherigen Programmen eine ungemein kompakte Lösung!

## 21.5 Listing: FORMAT.TAL

```

100 ' *****
110 ' *
120 ' *          FORMAT.TAL
130 ' *
140 ' *   Formateinstellung des B-Laufwerks fuer den TA-Alphatronic-PC
150 ' *
160 ' *****
170 '
180 DATA 64,0,4,15,1,151,0,127,0,192,0,32,0,2,0,1,16,&2A,&52,&E5,1,2
190 dpb.drive.a=PEEK(&BE42)+PEEK(&BE43)*256
200 dpb.drive.b=dpb.drive.a+64
210 base=dpb.drive.b
220 FOR i=0 TO 21:READ a:POKE base+i,a:NEXT i
230 POKE base+24,255
240 END ' -----

```

Völlig identisch ist das Programm FORMAT.BON aufgebaut, es verwendet nur andere DPB-Daten und macht damit Bondwell-12-Disketten lesbar:

## 21.6 Listing: FORMAT.BON

```

100 ' *****
110 ' *
120 ' *          FORMAT.BON
130 ' *
140 ' *   Formateinstellung des B-Laufwerks fuer den Bondwell-12
150 ' *
160 ' *****
170 '
180 DATA 36,0,4,15,1,84,0,127,0,192,0,32,0,2,0,1,16,&2A,&52,&E5,1,2
190 dpb.drive.a=PEEK(&BE42)+PEEK(&BE43)*256
200 dpb.drive.b=dpb.drive.a+64
210 base=dpb.drive.b
220 FOR i=0 TO 21:READ a:POKE base+i,a:NEXT i
230 POKE base+24,255
240 END ' -----

```

Wenn Sie eines der beiden Programme gestartet haben, können Sie alle Daten auf Disketten des jeweiligen Formats auslesen und sich auf dem Bildschirm anzeigen lassen.

Ein ziemlich ärgerliches Problem gibt es allerdings bei Formaten,

deren Sektorlänge über 512 Bytes liegt. Das ist zum Beispiel beim Osborne-Double-Density-Format der Fall. Dieses besitzt eine Sektorlänge von 1024 Zeichen. Denn der Floppy-Controller kann immer nur einen ganzen Sektor lesen, keine einzelnen Zeichen. Es ist nun die Aufgabe des AMSDOS-Systems, diesen Sektor intern abzulegen und daraus dem Benutzer einzelne Zeichen zur Verfügung zu stellen.

AMSDOS benutzt dazu einen Sektorpuffer zwischen den Adressen &A9B0 und &ABB0. Dieser ist 512 Bytes lang - entsprechend den Erfordernissen des Schneider-Diskettenformats.

Wenn aber eine Diskette längere Sektoren besitzt, überschreibt der Floppy-Controller ohne Rücksicht die 512 hinter dem Sektorpuffer liegenden Bytes.

Das führt in den allermeisten Fällen zum "Systemcrash". Denn ab &AC00 beginnt der Bereich, in dem der Basic-Interpreter seine Systemvariablen ablegt!

Durch die enorme Flexibilität des Betriebssystems ist aber auch hier Abhilfe in Sicht. Sie können ganz einfach die Routine "Floppy-RAM verschieben" benutzen, die auch den Sektorpuffer verschiebt.

Das Floppy-RAM muß nur in einen Bereich gelegt werden, in dem es keinen Schaden mehr anrichten kann.

Das Maschinenprogramm OSMOVE.ASM beziehungsweise dessen Basic-Lader OSMOVE.BAS verschiebt das Floppy-RAM in den Bereich hinter &9000.

## 21.7 Listing: OSMOVE.ASM

```
; *****
; *
; *   OSMOVE.ASM verschiebt das Floppy-RAM für das Osborne-Format   *
; *                                                                 *
; *****

                ORG    &9000

PCHL            EQU    &001B      ; KL FAR PCHL
BKINIT          EQU    &BCCE      ; KL INIT BACK
FINDCM          EQU    &BCD4      ; KL FIND COMMAND

MOVE            LD     HL,TAPE     ; Zeiger auf "TAPE"
                CALL  FINDCM      ; KL FIND COMMAND sucht RSX-Adresse
                RET  NC           ; Nicht gefunden, Basic-Ruecksprung
                SUB   A           ; Akku=0 - Keine Parameter uebergeben
                CALL  PCHL        ; RSX in Floppy-ROM aufrufen

DSINIT          LD     C,7         ; ROM-Auswahl-Adresse der Floppy
                LD     DE,&0040     ; Low-Adresse des Memory Pool
                LD     HL,&0000     ; Hoechste erlaubte Adresse fuer Floppy-RAM
```



```

CALL BKINIT      ; KL BACK INIT initialisiert Floppy nochmal
RET              ; Ruecksprung nach Basic

TAPE             DEFN "TAP"      ; RSX-Name ":TAPE"
                DEFB &C5

                END              ; *****

```

Dazu kann der Maschinencode mit CALL &9000 aufgerufen werden. Das Floppy-RAM wird dann von AMSDOS meist beginnend an der Adresse 39121 abgelegt. Erfahren können Sie diese Adresse durch:

```
PRINT PEEK(&BE42)+PEEK(&BE43)*256
```

Der Programmaufruf wird vom Basic-Loader OSMOVE.BAS sozusagen "nebenbei" miterledigt.

## 21.8 Listing: OSMOVE.BAS

```

100 ' *****
110 ' *
120 ' *      OSMOVE.BAS
130 ' *
140 ' *****
150 '
200 DATA &21,&17,&90,&CD,&D4,&BC,&D0,&97,&CD,&1B,&00
210 DATA &0E,&07,&11,&40,&00,&21,&40,&9C,&CD,&CE,&BC
220 DATA &C9,&54,&41,&50,&C5
230 MEMORY &8FFF
240 FOR i=&9000 TO &901A:READ a:POKE i,a:NEXT i
250 CALL &9000 ' Floppy-RAM verschieben
260 END ' -----

```

Wenn Sie danach das Programm FORMAT.OSB laufen lassen, wird das B-Laufwerk auf Osborne-DD-Disketten eingestellt:

## 21.9 Listing: FORMAT.OSB

```

100 ' *****
110 ' *
120 ' *      FORMAT.OSB
130 ' *
140 ' *      Formateinstellung des B-Laufwerks fuer Osborne (DD)
150 ' *
160 ' *****
170 '
180 DATA 40,0,3,7,0,184,0,63,0,192,0,16,0,3,0,1,5,&2A,&52,&E5,3,8
190 dpb.drive.a=PEEK(&BE42)+PEEK(&BE43)*256
200 dpb.drive.b=dpb.drive.a+64
210 base=dpb.drive.b
220 FOR i=0 TO 21:READ a:POKE base+i,a:NEXT i
230 POKE base+24,255
240 END ' -----

```

Einfacher wird es wohl, wenn Sie die beiden Programme OSMOVE.BAS und FORMAT.OSB zusammenfassen.

Dies ist beim Basic-Programm FORMAT.OS2 geschehen. Sie können es genauso anwenden wie FORMAT.TAL und FORMAT.BON.

#### 21.10 Listing: FORMAT.OS2

```

100 ' *****
110 ' *
120 ' *          FORMAT.OS2
130 ' *
140 ' *          Formateinstellung des B-Laufwerks fuer Osborne (DD)
150 ' *
160 ' *****
170 '
180 DATA &21,&17,&90,&CD,&D4,&BC,&D0,&97,&CD,&1B,&00
190 DATA &0E,&07,&11,&40,&00,&21,&40,&9C,&CD,&CE,&BC
200 DATA &C9,&54,&41,&50,&C5
210 MEMORY &8FFF
220 FOR i=&9000 TO &901A:READ a:POKE i,a:NEXT i
230 CALL &9000 ' Floppy-RAM verschieben
240 ' -----
250 DATA 40,0,3,7,0,184,0,63,0,192,0,16,0,3,0,1,5,&2A,&52,&E5,3,8
260 dpb.drive.a=PEEK(&BE42)+PEEK(&BE43)*256
270 dpb.drive.b=dpb.drive.a+64
280 base=dpb.drive.b
290 FOR i=0 TO 21:READ a:POKE base+i,a:NEXT i
300 POKE base+24,255
310 END ' -----

```

Damit sind wir einen ganz gewaltigen Schritt weitergekommen.

Der Schneider-CPC verarbeitet nun klaglos Disketten in anderen Formaten. Mit OPENIN können ASCII-Textdateien geöffnet, mit LINE INPUT #9 gelesen werden. OPENOUT, PRINT#9 und WRITE#9 schreiben Dateien auf die Diskette.

Doch Programmfiles sind immer noch nicht verwertbar. Unter "Programmfiles" sind CP/M-COM-Dateien und codierte Basic-Programme zu verstehen. Die Anfertigung von Kopien dieser Dateitypen wird aus zwei Gründen nahezu unmöglich gemacht: Manche Dateien melden bei OPENIN einen "File Type Error". Andere werden scheinbar korrekt übertragen, brechen aber mitten beim Einlesen mit "EOF met" ab. Das liegt daran, daß der Computer ein Control-Z-Zeichen (ASCII-Code 26) findet, das bei AMSDOS und CP/M als Dateiende-Markierung (EOF-Mark) von Textdateien verwendet wird.

Bei der internen Codierung kann Control-Z bei Basic-Programmen etwa in Zeilennummern vorkommen, und in Maschinenprogrammen unter CP/M kann dieser Code auch stecken: &1A entspricht dem Prozessorbefehl LDAX D beziehungsweise LD A,(DE).

Im Gegensatz zu den Basic-Befehlen OPENIN, INPUT#9 und LINE INPUT#9 ermöglichen die Maschinenroutinen CAS IN OPEN und CAS IN

CHAR die Bearbeitung aller Dateitypen.

So können Sie in Maschinensprache auch Basic-Programmfiles öffnen und lesen! Das Programm ACCESS.ASM stellt eine RSX-Erweiterung dar, mit der Sie auf solche Dateien zugreifen können. Die Befehle lauten im einzelnen:

```
A$="Dateiname":!OPENIN,@A$
A%=:!READCHAR,@A%
!CLOSEIN

A$="Dateiname":!OPENOUT,@A$
A%=:!WRITECHAR,@A%
!CLOSEOUT
```

Die Parameter-Übergabe erfolgt also in allen Fällen über den Variablenpointer "@". Dieser verlangt, daß alle verwendeten Variablen bereits definiert sein müssen. Dies kann durch eine Wertzuweisung geschehen: A%=0. Die Befehle !READCHAR und !WRITECHAR arbeiten nur mit Integervariablen. Diese sind durch ein folgendes Prozentzeichen oder durch eine vorherige DEFINT-Anweisung zu vereinbaren.

Es folgt der Quellcode von ACCESS.ASM.

#### 21.11 Assemblerlisting: ACCESS.ASM

```
; *****
; *
; *   ACCESS.ASM - Öffnen, Lesen und Schreiben aller Dateitypen   *
; *
; *****

; ***** Definitionen *****

                ORG        &A000

KL_UROM_ENABLE EQU    &B900        ; KL U ROM ENABLE
KL_LOG_EXTERNAL EQU    &BCD1        ; KL LOG EXTERNAL

CAS_IN_OPEN     EQU    &BC77        ; CAS IN OPEN
CAS_IN_CLOSE    EQU    &BC7A        ; CAS IN CLOSE
CAS_IN_CHAR     EQU    &BC80        ; CAS IN CHAR
CAS_OUT_OPEN    EQU    &BC8C        ; CAS OUT OPEN
CAS_OUT_CLOSE   EQU    &BC8F        ; CAS OUT CLOSE
CAS_OUT_CHAR    EQU    &BC95        ; CAS OUT CHAR

BASIC_ERROR     EQU    &CA94        ; Fehlerausgabe im Basic-ROM
GET_INBUFFER    EQU    &F632        ; Eingabepuffer reservieren
REL_INBUFFER    EQU    &F66D        ; Eingabepuffer freigeben
GET_OUTBUFFER   EQU    &F637        ; Ausgabepuffer reservieren
REL_OUTBUFFER   EQU    &F671        ; Ausgabepuffer freigeben

; ***** Initialisierung der RSX-Erweiterung *****

INIT_RSX        LD        BC,JUMPTABLE    ; Zeiger auf Sprungtabelle

330
```

```

LD      HL, KERNEL_RAM      ; Zeiger auf Hilfsspeicher
CALL    KL_LOG_EXTERNAL     ; RSXen ins System integrieren
RET     ; Rückkehr nach Basic

; ***** Daten zur RSX-Verwaltung *****

JUMPTABLE  DEFW  NAMETABLE      ; Zeiger auf Namenstabelle
JP        OPENIN               ; Sprung nach OPENIN
JP        OPENOUT              ; Sprung nach OPENOUT
JP        WRITECHAR            ; Sprung nach WRITECHAR
JP        READCHAR             ; Sprung nach READCHAR
JP        CLOSEIN              ; Sprung nach CLOSEIN
JP        CLOSEOUT             ; Sprung nach CLOSEOUT

NAMETABLE  DEFW  "OPENI"        ; RSX-Name "OPENIN"
          DEFB  "N"+&80
          DEFW  "OPENOU"       ; RSX-Name "OPENOUT"
          DEFB  "T"+&80
          DEFW  "WRITECHA"     ; RSX-Name "WRITECHAR"
          DEFB  "R"+&80
          DEFW  "READCHA"      ; RSX-Name "READCHAR"
          DEFB  "R"+&80
          DEFW  "CLOSEI"       ; RSX-Name "CLOSEIN"
          DEFB  "N"+&80
          DEFW  "CLOSEOU"      ; RSX-Name "CLOSEOUT"
          DEFB  "T"+&80
          DEFB  &00            ; Zeigt Tabellenende an

KERNEL_RAM  DEFS  &04          ; Hilfsspeicher für Kernel

; ***** OPENIN *****

OPENIN      CALL  GET_FILENAME   ; Dateinamen holen
          PUSH  BC              ; BC-Register sichern
          PUSH  HL              ; HL-Register sichern
          CALL  KL_UROM_ENABLE   ; Basic-ROM einschalten
          CALL  GET_INPBUFFER    ; Eingabepuffer reservieren
          POP   HL              ; HL-Register wiederherstellen
          POP   BC              ; BC-Register wiederherstellen
          CALL  CAS_IN_OPEN      ; Datei zum Lesen eröffnen
          RET     ; Rücksprung nach Basic

; ***** OPENOUT *****

OPENOUT     CALL  GET_FILENAME   ; Dateinamen holen
          PUSH  BC              ; BC-Register sichern
          PUSH  HL              ; HL-Register sichern
          CALL  KL_UROM_ENABLE   ; Basic-ROM einschalten
          CALL  GET_OUTBUFFER    ; Ausgabepuffer reservieren
          POP   HL              ; HL-Register wiederherstellen
          POP   BC              ; BC-Register wiederherstellen
          CALL  CAS_OUT_OPEN     ; Datei zum Schreiben öffnen
          RET     ; Rücksprung nach Basic

; ***** WRITECHAR *****

WRITECHAR   CP      1           ; Wird 1 Argument Übergeben?

```

```

JR      NZ, ARGUM_ERROR    ; Nein - Fehler ausgeben
LD      H, (IX+1)          ; Adresse der Variablen im
LD      L, (IX+0)          ; RSX-Aufruf holen
LD      A, (HL)            ; Variableninhalt lesen
CALL    CAS_OUT_CHAR       ; In die Datei übertragen
RET                                           ; Rücksprung nach Basic

; ***** READCHAR *****

READCHAR CP      1          ; Wird 1 Argument übergeben?
JR      NZ, ARGUM_ERROR    ; Falls nicht - Fehler ausgeben
CALL    CAS_IN_CHAR        ; Zeichen aus der Datei lesen
LD      H, (IX+1)          ; Variablenadresse holen
LD      L, (IX+0)          ; Ebenso das zweite Byte
LD      (HL), A            ; In die Variable übertragen
INC     HL                 ; Zeiger auf Variable erhöhen
LD      (HL), 0            ; Highbyte der Variablen löschen
RET                                           ; Rücksprung nach Basic

; ***** CLOSEIN *****

CLOSEIN CALL    CAS_IN_CLOSE ; Eingabedatei schließen
CALL    KL_UROM_ENABLE      ; Basic-ROM einschalten
CALL    REL_INBUFFER        ; Eingabepuffer freigeben
RET                                           ; Rücksprung nach Basic

; ***** CLOSEOUT *****

CLOSEOUT CALL    CAS_OUT_CLOSE ; Ausgabedatei schließen
CALL    KL_UROM_ENABLE       ; Basic-ROM einblenden
CALL    REL_OUTBUFFER        ; Ausgabepuffer freigeben
RET                                           ; Rücksprung nach Basic

; ***** Unterprogramme *****

GET_FILENAME CP      1          ; Ein Parameter im Aufruf?
JR      NZ, PARAM_ERROR    ; Nein - Fehler melden
LD      H, (IX+1)          ; Adresse des String-Descriptors
LD      L, (IX+0)          ; ins HL-Register holen
LD      B, (HL)            ; Stringlänge ins B-Register
INC     HL                 ; Zeiger erhöhen
LD      E, (HL)            ; Erstes Byte der Stringadresse
INC     HL                 ; Zeiger erhöhen
LD      D, (HL)            ; Zweites Byte der Stringadresse
EX      DE, HL             ; Adresse ins HL-Register
RET                                           ; Rücksprung nach Basic

PARAM_ERROR POP      HL      ; Stack korrigieren

ARGUM_ERROR CALL    KL_UROM_ENABLE ; Basic-ROM einschalten
LD      E, 5              ; Nummer für "Improper Argument"
JP      BASIC_ERROR       ; Fehleroutine im Basic-ROM
END
; -----

```

Das Ganze nun als Basic-Lader.

## 21.12 Listing: ACCESS.BAS

```

100 ' *****
110 ' *
120 ' * ACCESS.BAS *
130 ' *
140 ' *****
150 '
160 DEF FNlsb(a)=255 AND UNT(a)
170 DEF FNmsb(a)=255 AND INT(a/256)
180 MEMORY HIMEM-197
190 start=HIMEM+1
200 ' Initialisierung des Basic-Loaders *****
210 SYMBOL 253,&66,&0,&78,&C,&7C,&CC,&76,&0
220 SYMBOL 254,&66,&0,&3C,&66,&66,&66,&3C,&0
230 SYMBOL 255,&66,&0,&66,&66,&66,&66,&3E,&0
240 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
250 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
260 ' Titelbild und Benutzerinformationen *****
270 LOCATE 1,1:PRINT STRING$(40,210);
280 LOCATE 1,2
290 PRINT CHR$(24)+SPACE$(15)+"ACCESS.BAS"+SPACE$(15)+CHR$(24)
300 PRINT:PRINT STRING$(40,210):PRINT
310 PEN 1:PRINT " Diese Routinen erlauben das Lesen und"
320 PRINT:PRINT " Beschreiben aller Diskettendateien mit"
330 PRINT:PRINT " folgenden RSX-Befehlen:"
340 PRINT:PRINT " :OPENIN,@NAME$ :OPENOUT,@NAME$"
350 PRINT:PRINT " :READCHAR,@A% :WRITECHAR,@A%"
360 PRINT:PRINT " :CLOSEIN :CLOSEOUT"
370 PRINT:PRINT
380 PRINT:PEN 1
390 FOR i=&A000 TO &A0C1:READ a:sum=sum+a:NEXT i
400 IF sum=24515 THEN 440
410 PRINT CHR$(7);"* Fehler in den DATA-Zeilen!":PRINT
420 LIST 610:-END
430 ' Das Maschinenprogramm wird eingePOKET und verschoben *****
440 RESTORE:FOR i=start TO start+C1:READ a:POKE i,a:NEXT i
450 FOR i=1 TO 11:READ a:a=a-40960+start
460 wert=PEEK(a)+PEEK(a+1)*256-40960+start
470 POKE a,FNlsb(wert):POKE a+1,FNmsb(wert)
480 NEXT i
490 CALL start ' RSX-Erweiterung initialisieren
500 ' Computertyp erkennen und Programmcode anpassen *****
510 POKE &160,&CD:POKE &161,&0:POKE &162,&B9:POKE &163,&3A:POKE &164,&2
520 POKE &165,&C0:POKE &166,&32:POKE &167,&60:POKE &168,&1:POKE &169,&C9
530 CALL &160:cpcversion=PEEK(&160)
540 IF cpcversion=0 THEN END ' CPC-464: keine Anpassung!
550 POKE start+&59,&20:POKE start+&5A,&F7
560 POKE start+&6A,&25:POKE start+&6B,&F7
570 POKE start+&9A,&59:POKE start+&9B,&F7
580 POKE start+&A4,&5D:POKE start+&A5,&F7
590 POKE start+&C0,&CB
600 IF cpcversion=1 THEN POKE start+&BF,&55 ELSE POKE start+&BF,&4D
610 ' DATA-Zeilen fuer den Maschinencode *****

```

```

620 DATA &01,&0A,&A0,&21,&4C,&A0,&CD,&D1,&BC,&C9,&1E,&A0,&C3,&50,&A0,&C3
630 DATA &61,&A0,&C3,&72,&A0,&C3,&81,&A0,&C3,&93,&A0,&C3,&9D,&A0,&4F,&50
640 DATA &45,&4E,&49,&CE,&4F,&50,&45,&4E,&4F,&55,&D4,&57,&52,&49,&54,&45
650 DATA &43,&48,&41,&D2,&52,&45,&41,&44,&43,&48,&41,&D2,&43,&4C,&4F,&53
660 DATA &45,&49,&CE,&43,&4C,&4F,&53,&45,&4F,&55,&D4,&00,&00,&00,&00,&00
670 DATA &CD,&A7,&A0,&C5,&E5,&CD,&00,&B9,&CD,&32,&F6,&E1,&C1,&CD,&77,&BC
680 DATA &C9,&CD,&A7,&A0,&C5,&E5,&CD,&00,&B9,&CD,&37,&F6,&E1,&C1,&CD,&8C
690 DATA &BC,&C9,&FE,&01,&20,&43,&DD,&66,&01,&DD,&6E,&00,&7E,&CD,&95,&BC
700 DATA &C9,&FE,&01,&20,&34,&CD,&80,&BC,&DD,&66,&01,&DD,&6E,&00,&77,&23
710 DATA &36,&00,&C9,&CD,&7A,&BC,&CD,&00,&B9,&CD,&6D,&F6,&C9,&CD,&8F,&BC
720 DATA &CD,&00,&B9,&CD,&71,&F6,&C9,&FE,&01,&20,&0D,&DD,&66,&01,&DD,&6E
730 DATA &00,&46,&23,&5E,&23,&56,&EB,&C9,&E1,&CD,&00,&B9,&1E,&05,&C3,&94
740 DATA &CA,&00
750 ' DATAs zum Relokalisieren des Maschinencodes *****
760 DATA &A001,&A004,&A00A,&A00D,&A010,&A013,&A016,&A019,&A01C,&A051,&A062
770 END ' -----

```

!OPENIN öffnet eine Datei, und zwar unabhängig von ihrem Typ. Sie können also eingeben:

```

100 PRINT "Dies ist ein einfaches Basic-Programm!"
110 END

SAVE "PROG.BAS"

a$="PROG.BAS"
!OPENIN,@a$
c%=0
FOR i=1 TO 60: !READCHAR,@c%:PRINT CHR$(1);CHR$(c%);:NEXT i
!CLOSEIN

```

Das Programm "PROG.BAS" wird dann in der Form sichtbar, in der es intern codiert ist. CHR\$(1) sorgt dafür, daß auch Steuerzeichen, deren ASCII-Code zwischen 0 und 31 liegt, als Symbole ausgegeben werden und nicht als Steuerbefehle interpretiert werden. Auf diese Art lassen sich die Steuercodes auch auf dem Bildschirm sichtbar machen.

Leider können Sie nicht auf die Systemvariable EOF zurückgreifen, die bei ASCII-Dateien das Dateiende anzeigt. Aus oben genannten Gründen bricht nämlich dann die Übertragung ab, wenn ein Control-Z entdeckt wird. Das einfachste ist, mit CAT die Dateigröße in Erfahrung zu bringen. Leider ist diese Angabe nur auf einen Block genau. Bei 1K-Blocks wie beim Schneider-CPC kann es also sein, daß 1023 Bytes zuviel angezeigt werden, bei 4K-Blocks auch mal 4095 Bytes!

Sie sollten sich also die Datei auf dem Bildschirm anzeigen lassen, bis Sie über das Dateiende hinaus gelangt sind. Dies äußert sich darin, daß ab diesem Zeitpunkt alle Zeichen identisch sind. Sie entsprechen dem letzten Zeichen der Datei. Eine andere Möglichkeit ist auch, daß der Computer aus dem zwei Kilobytes großen Dateipuffer weitere Zeichen ausliest. Dann entsprechen diese den Daten, die vorher im Puffer standen.

!CLOSEIN schließt eine Eingabedatei wieder und gibt im Gegensatz

zum Basic-Befehl CLOSEIN auf jeden Fall den Pufferspeicher wieder frei.

!OPENOUT funktioniert analog zu !OPENIN und ist dazu gedacht, auf dem jeweils anderen Laufwerk eine Ausgabedatei zu öffnen, in die die Dateien auf das Schneider-Format umkopiert werden. !WRITECHAR schreibt ein einzelnes Zeichen in die Datei und !CLOSEOUT schließt sie wieder.

Kurz zur Arbeitsweise von ACCESS: Die RSX-Befehle greifen hauptsächlich auf bereits bestehende ROM-Routinen zurück, deren Aufruf von Basic aus unmöglich ist. So übernehmen !OPENIN und !OPENOUT einen Dateinamen aus Basic und öffnen dann entsprechend eine Eingabe- oder Ausgabedatei.

!READCHAR liest ein Zeichen aus der Datei und übermittelt es der angegebenen Basic-Variablen.

!WRITECHAR liest das Zeichen aus Basic und schreibt es in die Ausgabedatei. CLOSEIN und CLOSEOUT rufen direkt CAS IN CLOSE und CAS OUT CLOSE auf.

ACCESS.BAS ist übrigens nicht nur beim Lesen von fremden Diskettenformaten von Nutzen, es kann auch bei anderen Gelegenheiten Verwendung finden. Wenn Sie das Programm eingeben, können Sie entweder den Assembler-Quellcode abtippen und das Programm dann assemblieren oder stattdessen den Basic-Lader eingeben.

Dieser verschiebt das Maschinenprogramm automatisch unterhalb HIMEM und initialisiert die Erweiterung. Außerdem paßt er das Maschinenprogramm bei Bedarf selbsttätig an den CPC-664 oder CPC-6128 an. Dies muß gemacht werden, weil ACCESS auf ROM-Routinen im Basic-ROM zurückgreift. Das sind die Routinen, die Ein- und Ausgabepuffer reservieren beziehungsweise freigeben und eine Basic-Fehlermeldung erzeugen.

Da die Benutzung von Dateipuffern bei allen Cassetten- und Diskettenroutinen verlangt wird, finden Sie hier Hinweise zur Verwendung der ROM-Routinen:

Die Systemroutinen CAS IN OPEN und CAS OUT OPEN erwarten im DE-Register einen Zeiger auf einen Dateipuffer. In diesem 2 Kilobytes großen RAM-Speicherbereich werden alle Daten zwischendurch abgelegt.

Wenn ein Maschinenprogramm, das auf diese Routinen des Betriebssystems angewiesen ist, mit einem Basic-Programm zusammenarbeiten soll, wird es kompliziert. Woher soll man den Pufferspeicher nehmen? Dieser darf ja nicht in einem vom Basic-Programm beanspruchten Bereich liegen. Andererseits ist es auch nicht schön, HIMEM um diese 2048 Bytes zu vermindern, nur um für die Maschinenroutinen einen Pufferspeicher zu haben. Da bietet es sich natürlich an, das Basic-ROM nach passenden Routinen zu durchsuchen. Und hier sind sie:

Eingabepuffer reservieren	&F632	&F720
Eingabepuffer freigeben	&F66D	&F759



Ausgabepuffer reservieren	&F637	&F725
Ausgabepuffer freigeben	&F671	&F75D

Die ersten Adresse gelten für den CPC-464, die zweiten für die Modelle CPC-664 und CPC-6128.

Vor Aufruf der Routinen muß aber noch das Basic-ROM freigeschaltet werden. Diese Aufgabe übernimmt die Systemroutine KL U ROM ENABLE (Adresse &B900). Sie müssen nur die beiden CALL-Aufrufe kombinieren und erhalten dann im DE-Register einen Zeiger auf den Pufferspeicher. Der Inhalt des DE-Registers kann gleich an CAS IN OPEN beziehungsweise CAS OUT OPEN weitergereicht werden:

```
CALL U_ROM_ENABLE
CALL &F66D
LD B,Namenslänge
LD HL,Namensadresse
CALL CAS_IN_OPEN
```

Mit ACCESS können Sie jetzt Dateien von Fremddisketten auf das Schneider-Format übertragen. Folgende Reihenfolge der Arbeitsschritte hat sich bewährt:

- Legen Sie die Fremddiskette ins Laufwerk B:, im Laufwerk A: liegt eine Diskette mit ACCESS.BAS und FORMAT.xxx.

Wenn Sie eine Bondwell-Diskette lesen wollen, nehmen Sie logischerweise FORMAT.BON, für Triumph-Adler-Disketten FORMAT.TAL. Osborne-Disketten werden dem System durch FORMAT.OS2 vorgesetzt.

- Starten Sie die Umschaltung auf das Fremdformat mit RUN "A:FORMAT.xxx". Der Inhalt der Diskette im Laufwerk B: kann nun mit !DIR und CAT festgestellt werden.
- Merken Sie sich jetzt die Namen der zu kopierenden Dateien und deren ungefähre Größe.
- Laden Sie ACCESS mit RUN "A:ACCESS.BAS". Nehmen wir an, die Datei, die übertragen werden soll, heißt "BONDWELL.TXT". Sie geben dann die folgenden Befehle ein:

```
a$="B:BONDWELL.TXT"
c%=0
!OPENIN,@a$
FOR i=1 TO 2000000:READCHAR,@c%:PRINT CHR$(i);CHR$(c%);:NEXT i
```

Sobald Sie das Dateiende erkannt haben, drücken Sie zweimal die ESC-Taste. Nachdem "Break" erschienen ist, tippen Sie PRINT i und schließen Sie die Datei mit !CLOSEIN. Der Wert in der Variablen i gibt an, wie lang die Datei ist.

Jetzt können Sie die Datei wiederum öffnen und die Zeichen in eine Datei im Laufwerk A: schreiben:

```
a$="B:BONDWELL.TXT"
c%=0
!OPENIN,@a$
```

```

a$="A:COPY.TXT"
!OPENOUT,@a$
FOR j=1 TO 1: !READCHAR,@c%: !WRITECHAR,@c%: NEXT j
!CLOSEIN: !CLOSEOUT

```

Damit steht die Datei auf einer Diskette im normalen Schneider-Format und kann mit allen zur Verfügung stehenden Programmen und Befehlen bearbeitet werden.

Wir sind aber noch nicht ganz fertig. Wesentlich komfortabler wäre es doch, wenn man die Fremddisketten direkt unter CP/M 2.2 lesen und beschreiben könnte. Nichts einfacher als das! Denn unter CP/M sind die Disk-Parameter-Blöcke völlig identisch aufgebaut wie unter AMSDOS, und sogar das dringend benötigte "Log-in-Flag" ist verfügbar.

Wählen Sie das CP/M-Betriebssystem durch Aufruf des RSX-Kommandos !CPM und suchen Sie sich eine Diskette heraus, auf der sich der Systemdebugger und Maschinensprache-Monitor DDT.COM befindet. Diesen haben Sie beim Kauf Ihrer Diskettenstation auf der Systemdiskette geliefert bekommen. Dieser Monitor mit dem äußerst giftigen Namen ("DDT" steht hier allerdings für "Dynamic Debugging Tool") kann verwendet werden, um Maschinenprogramme auszutesten.

Wir werden ihn aber benutzen, um kleine Programme zu schreiben und sofort ablaufen zu lassen.

Laden Sie ihn durch Eingabe von DDT DDT.COM. Durch die Angabe des Dateinamens "DDT.COM" wird der Debugger gezwungen, diese Datei an die Adresse &0100, den Beginn des CP/M-TPA-Programmspeichers, zu laden und sich selbst im Speicher nach oben zu verschleiben. So besteht keine Gefahr, daß wir versehentlich Teile von DDT überschreiben und eventuell einen Systemabsturz provozieren.

Der Debugger meldet sich mit "DDT VERS 2.2" und den Angaben für NEXT und PC. Diese interessieren uns aber nicht. Wichtiger sind die Kommandos von DDT. Sie bestehen aus einzelnen Buchstaben und - je nach Befehl - darauf folgenden Hexadezimalzahlen. Hier finden Sie eine Aufstellung der DDT-Befehle:

- A = Assemble (Assembler aufrufen)  
Beispiel: A0100
- D = Dump (Speicherauszug ausgeben)  
Beispiel: D0100,0200
- F = Fill (Speicherbereich mit einem Byte füllen)  
Beispiel: F0100,0300,FF
- G = Goto (Maschinenroutine aufrufen)  
Beispiel: G0000
- H = Hex Arithmetic (Hexadezimale Addition und Subtraktion)  
Beispiel: H0001,0002
- I = Input File (Dateinamen zum Einlesen vorgeben)  
Beispiel: ICDPYDISC.COM

- L = List (Disassembliertes Speicherlisting)  
Beispiel: L0100,0120
- M = Move (Speicherbereich kopieren)  
Beispiel: M0100,0300,5500
- R = Read (Mit I vorbereitete Datei einlesen)  
Beispiel: R
- S = Store (Speicherbereich byteweise anzeigen und ändern)  
Beispiel: S0100
- T = Trace (Einzelschrittabarbeitung mit Registerausgabe)  
Beispiel: T (1 Schritt)  
          T5 (5 Schritte)
- U = Untrace (Einzelschritt ohne Registerausgabe)  
Beispiel: U (1 Schritt)  
          U8 (8 Schritte)
- X = Examine Register (Register anzeigen und abändern)  
Beispiel: X (Ausgabe aller Register)  
          XA, XB, XD, XH, XP, XS (Registeränderung)  
          XC (Änderung des Carry-Flags)  
          XZ (Änderung des Zero-Flags)  
          XM (Änderung des Minus-Flags)  
          XE (Änderung des Even-Parity-Flags)  
          XI (Änderung des Interdigit-Flags)

Der DPB-Block liegt in CP/M nicht gezwungenermaßen an der gleichen Stelle wie unter AMSDOS. Um die Startadresse zu bestimmen, kann die BDOS-Funktion 31 verwendet werden. Diese liefert im HL-Doppelregister die Adresse. RST 6 besorgt den Rücksprung in den DDT-Debugger. Geben Sie bitte das folgende kleine Maschinenprogramm ein:

```
A0100
0100 MVI C,1F
0102 CALL 5
0105 RST 6
0106 .
```

Mit G0100 wird das Programm aufgerufen. Sofort meldet sich der Debugger mit der Endadresse "\*0105" zurück. Tippen Sie "X" und <Enter>. Die Registerleiste wird angezeigt. Wenn Sie den Debugger vom Laufwerk A: aus gestartet haben, steht im HL-Doppelregister die Adresse &ADD8. Das ist die Startadresse des DPB von Laufwerk A:. Wenn Sie DADD8,AE00 eingeben, wird ein Speicher-Dump ausgegeben. Wie Sie unschwer erkennen können, ist das tatsächlich der DPB.

Um an die Adresse des zweiten DPBs zu gelangen, setzen wir mit der BDOS-Funktion 14 die Laufwerksnummer auf 1 und lassen uns dann wieder die DPB-Adresse berechnen:

```
A0100
0100 MVI E,1
0102 MVI C,E
```

```

0104 CALL 5
0107 MVI C,1F
0109 CALL 5
010C RST 6
010D .

```

Mit G0100 und X erscheint - wie gehabt - der Inhalt des HL-Registers. Die Adresse liegt jetzt bei &AE18. Ein DAE18,AE40 beweist wieder augenfällig, daß es sich um einen DPB-Block handelt. Die Differenz der beiden Adressen errechnet HAE18,ADD8 als &0040 entsprechend der Zahl 64 im dezimalen Zahlensystem. Merken wir uns also, daß die beiden DPBs (normalerweise) um 64 Bytes auseinander liegen.

Vorausgesetzt, Sie haben eine Bondwell-Diskette, können Sie mit dem S-Kommando entsprechend der folgenden Aufstellung die Daten in den DPB eingeben. Daneben sind die Daten für das Alphatronic-Format gelistet. Das Osborne-DD-Format ist unter CP/M kaum lesbar, weil der 1K-Sektorpuffer (Sektorlänge 1024 Bytes!) nicht zur Verfügung steht.

## Bondwell:

```

SAE18
AE18=24 24
AE19=00 00
AE1A=03 04
AE1B=07 0F
AE1C=00 01
AE1D=AA 54
AE1E=00 00
AE1F=3F 7F
AE20=00 00
AE21=C0 C0
AE22=00 00
AE23=10 20
AE24=00 00
AE25=02 02
AE26=00 00
AE27=41 00
AE28=09 11
AE29=2A 2A
AE2A=52 52
AE2B=E5 E5
AE2C=02 01
AE2D=04 02
AE2E=00 00
AE2F=00 00
AE30=00 FF

```

## Alphatronic:

```

SAE18
AE18=24 40
AE19=00 00
AE1A=03 04
AE1B=07 0F
AE1C=00 01
AE1D=AA 97
AE1E=00 00
AE1F=3F 7F
AE20=00 00
AE21=C0 C0
AE22=00 00
AE23=10 20
AE24=00 00
AE25=02 02
AE26=00 00
AE27=41 01
AE28=09 10
AE29=2A 2A
AE2A=52 52
AE2B=E5 E5
AE2C=02 01
AE2D=04 02
AE2E=00 00
AE2F=00 00
AE30=00 FF

```

Nachdem Sie Control-C gedrückt oder G0000 eingegeben haben, ist das Laufwerk B: auf das entsprechende Format ausgerichtet. Alle CP/M-Befehle und Programme können jetzt mit dem neuen Format arbeiten und auf die Dateien der Fremddiskette zugreifen. Am interessantesten dürften hier wohl DIR und PIP.COM sein. Wenn auf der Diskette im A-Laufwerk PIP.COM enthalten ist und auf der Diskette genügend Platz verbleibt, kopiert der folgende Programm-

aufruf alle Dateien von B: nach A:.

A>PIP A:=B:\*. \*

Natürlich kann man die Eingabe der Bytes mit DDT.COM nicht gerade als komfortabel bezeichnen. Wesentlich besser geeignet für eine regelmäßige Benutzung ist das Programm SF.COM, das wir schreiben werden. "SF" steht nicht etwa für "Science-Fiction", sondern ganz einfach für "Set Format".

Sie haben die Wahl zwischen einem Turbo-Pascal-Programm und einem echten Maschinenprogramm. Das Pascal-Programm SF.PAS bringt weniger Eintipparbeit und ist leichter zu verstehen, benötigt dafür aber 10 Kilobytes Speicherplatz auf der Diskette. Der Grund dafür ist, daß die Laufzeitbibliothek des Pascal-Compilers zum Code dazugebunden werden muß. Davon ist das eigentliche aus dem Pascal-Quellcode erzeugte Programm nur 1368 Bytes lang, die Daten 244 Bytes. Der ganze Rest von etwa acht Kilobytes wird von den Runtime-Routinen des Pascal-Systems eingenommen. Das Maschinenprogramm SF.ASM hat zwar mehr Zeilen zum Abtippen, dafür ist das COM-File dann auch nur ein Kilobyte lang.

Die beiden Programme unterscheiden sich nur minimal in ihren Bildschirmausgaben, in der Funktionsweise hingegen überhaupt nicht. Der Aufbau des Bildschirms ist beim Pascal-Programm so:

Fremdformate auf dem Schneider-CPC

-----

- 0 - Ende
- 1 - Bondwell-Format
- 2 - Alphatronic-PC
- 3 - Schneider-Systemformat

Bitte wählen Sie: \*

Das Maschinenprogramm arbeitet im Gegensatz dazu im MODE i und zeigt sich so:

Fremdformatleser fuer den CPC

-----

- 1- TA Alphatronic-PC
  - 2- Bondwell-12
  - 3- Schneider CP/M-Systemformat
- <ENTER> Programmabbruch

Wer sich in Pascal gut auskennt und von den Möglichkeiten von Turbo-Pascal weiß, wird sicher bald merken, daß das Programm keineswegs allzu elegant programmiert ist. Es zeigt die typische Programmierweise eines eingefleischten Basic-Fans.

Aber die Hauptsache ist ja wohl, daß das Programm korrekt arbeitet!

Um arbeitsfähige Programme zu erhalten, müssen Sie beim Maschinenprogramm so vorgehen:

- Abtippen des folgenden Quellcodes mit einem Texteditor in eine Datei SF.ASM:

### 21.13 Assemblerlisting: SF.COM

```
*****
*
*                               SF.COM - Set Format
*
*   Verarbeitet Alphatronic- und Bondwell-Disketten unter CP/M 2.2
*
*****

*
*   Definitionen und Equates
*   -----
*

WARM    EQU    0                ; Warmstart-Adresse für CP/M
BDOS    EQU    5                ; BDOS-Einsprung
CONIN   EQU    1                ; Konsoleneingabe
CONOUT  EQU    2                ; Konsolenausgabe
STROUT  EQU    9                ; Stringausgabe
SETDRV  EQU    14               ; Laufwerk festlegen
GETDPB  EQU    31               ; DPB-Adresse holen
TPA     EQU    0100H            ; Beginn der CP/M-TPA

                ORG    TPA        ; Programmzähler festsetzen

START   LXI    SP,STACK        ; Privaten Prozessorstack benutzen

DRIVE%A MVI    C,SETDRV        ; Laufwerk A: anwählen
        MVI    E,0
        CALL   BDOS

MENU    MVI    C,STROUT        ; Titelbild ausgeben
        LXI    D,MESSAG1
        CALL   BDOS

SELECT  MVI    C,CONIN         ; Tastatur abfragen
        CALL   BDOS

IF$CR   CPI    13              ; War es die Enter-Taste?
        JZ     TEXTLST        ; Ja, Programm abbrechen, Warmstart

IF$1    CPI    '1'             ; War es die '1'?
        JZ     SET$TA         ; Ja - Alphatronic-Format

IF$2    CPI    '2'             ; War es die '2'?
        JZ     SET$BO         ; Ja - Bondwell-Format

IF$3    CPI    '3'             ; War es die '3'?
        JZ     SET$CPC        ; Ja - Schneider-Format

BELL    PUSH   PSW             ; Zeichen sichern
        MVI    E,7            ; Keine der Tasten:
```

```

MVI    C,CONOUT      ; Warnton ausgeben
CALL   BDOS
POP     PSW           ; Zeichen wiederherstellen
CPI     32            ; Wenn Controlcode
JC      SELECT        ; Cursor nicht nach links

C$LEFT MVI    E,8      ; Backspace-Code
        MVI    C,CONOUT ; Konsolenausgabe
        CALL   BDOS     ; Cursor nach links
        JMP     SELECT   ; und Rücksprung in Abfrageschleife

SET$TA  LXI    H,TAPCTAB ; Zeiger auf Alphatronic-DPB
        JMP     CONT

SET$BO  LXI    H,BOND12  ; Zeiger auf Bondwell-DPB
        JMP     CONT

SET$CPC LXI    H,CPCTAB  ; Zeiger auf Schneider-DPB

CONT    PUSH   H         ; Zeiger sichern
        MVI    C,GETDPB  ; DPB-Adresse holen
        CALL   BDOS
        LXI    D,64      ; Der B:-DPB liegt 64 Bytes hinter
        DAD    D          ; dem DPB von Laufwerk A:, also muß
        XCHG          ; zur Adresse die Zahl 64 addiert werden
        POP     H         ; Zeiger wiederherstellen
        LXI    B,25      ; 25 Bytes umzukopieren
        DB     0EDH       ; Diese beiden Bytes simulieren den
        DB     0B0H       ; Z80-spezifischen Befehl "LDIR"

TEXTLST LXI    D,MESSAG2 ; Bildschirm löschen, MODE 2
        MVI    C,STROUT  ; wiederherstellen, das geht über die
        CALL   BDOS      ; Stringausgabe von Steuerzeichen
        JMP     WARM      ; Warmstart auslösen

;-----
;      Tabelle für das Format des Triumph-Adler Alphatronic-PC
;-----

TAPCTAB DW     64        ; SPT - 128-Byte-Sektoren je Track
        DB     4         ; BSH - Block Shift
        DB     15        ; BLM - Block Mask
        DB     1         ; EXM - Extent Mask
        DW     151       ; DSM - Maximale Blocknummer -1
        DW     127       ; DRM - Directory-Einträge -1
        DB     192       ; AL0 - Reservierung Dir-Blöcke/Byte 1
        DB     0         ; AL1 - Reservierung Dir-Blöcke/Byte 2
        DW     32        ; CKS - Bei Diskwechsel getestete Einträge
        DW     2         ; OFF - Spuroffset für Systemspuren

TAPCTA2 DB     0001H     ; FSC - Erster Sektor jedes Tracks
        DB     0010H     ; NUM - Anzahl der Sektoren je Track
        DB     002AH     ; GRW - Gap für den FDC bei Read/Write
        DB     0052H     ; GAF - Gap für den FDC beim Formatieren
        DB     00E5H     ; FIL - Fullbyte beim Formatieren
        DB     0001H     ; BPS - Bytes pro Sektor (=256 Bytes)
        DB     0002H     ; RPS - Records pro Sektor

```

```

DB      0000H      ; 1.Hilfsspeicher
DB      0000H      ; 2.Hilfsspeicher
DB      00FFH      ; Flag zum Unterdrücken des Log-Ins

```

```

; -----
;      Tabelle für das Bondwell-12-Format
; -----

```

```

BOND12  DW      36      ; SPT
        DB       4      ; BSH
        DB      15      ; BLM
        DB       1      ; EXM
        DW      84      ; DSM
        DW     127      ; DRM
        DB     192      ; AL0
        DB       0      ; AL1
        DW      32      ; CKS
        DW       2      ; OFF

BOND12A DB       0      ; FSC
        DB      17      ; NUM
        DB     002AH    ; GRW
        DB     0052H    ; GAF
        DB     00ESH    ; FIL
        DB     0001H    ; BPS
        DB     0002H    ; RPS
        DB     0000H    ; 1.Hilfsspeicher
        DB     0000H    ; 2.Hilfsspeicher
        DB     00FFH    ; Flag, das den Log-In verhindert

```

```

; -----
;      Tabelle für das Schneider-CP/M-Format
; -----

```

```

CPCTAB  DW      36      ; SPT
        DB       3      ; BSH
        DB       7      ; BLM
        DB       0      ; EXM
        DW     0AAH     ; DSM
        DW      63      ; DRM
        DB     192      ; AL0
        DB       0      ; AL1
        DW      16      ; CKS
        DW       2      ; OFF

        DB     0041H    ; FSC
        DB     0009H    ; NUM
        DB     002AH    ; GRW
        DB     0052H    ; GAF
        DB     00ESH    ; FIL
        DB     0002H    ; BPS
        DB     0004H    ; RPS
        DB     0000H    ; 1.Hilfsspeicher
        DB     0000H    ; 2.Hilfsspeicher
        DB     0000H    ; Log-In wieder zulassen

```



```
*      Titelmeldung des Programms
```

```
*      -----
```

```
MESSAG1 DB      4,1,10,10,10,32
          DB      'Fremdformatleser fuer den CPC'
          DB      13,10,10,32
          DB      '-----'
          DB      13,10,10,10,32
          DB      '1-      TA Alphatronic-PC'
          DB      13,10,10,32
          DB      '2-      Bondwell-12'
          DB      13,10,10,32
          DB      '3-      Schneider CP/M-Systemformat'
          DB      13,10,10,32
          DB      91,'ENTER',93
          DB      32,32,'Programmabbruch'
          DB      13,10,10,10,32,32,32,32,32
          DB      32,32,32,32,32,'$'
```

```
*
*      Schlußtext: MODE 2 einschalten
```

```
*      -----
```

```
MESSAG2 DB      4,2,'$'
```

```
*
*      Platz für den Programmstack
```

```
*      -----
```

```
STACK DS      20
      EQU      $
      END
```

```
;
```

- Assemblieren mit ASM SF.AAA, wenn sich die Diskette im Laufwerk A: befindet.

- Linken mit LOAD SF. Danach haben Sie ein Programm SF.COM, das durch die Eingabe von "SF" gestartet wird.

Das Turbo-Pascal-Programm wird etwas anders erzeugt:

- Laden von Turbo-Pascal, entweder über die Autostart-Option oder durch Eingabe von "TURBO". Der Speicherplatz reicht auch in der 464/664-Version von Turbo-Pascal aus, um die Fehlermeldungen zuzulassen, also "Include error messages (Y/N)?" : Y für "Yes".

- Eingabe des Programms durch "E" für "Edit" und als Dateinamen SF.PAS. Viel Spaß beim Eintippen in den Turbo-Editor!

## 21.14 Turbo-Pascal-Listing: SETFORMATPAPN

```

(*****
*)
*) Turbo-Pascal-Programm SetFormat fuer den Schneider-CPC unter CP/M 2.2 *)
*) ----- *)
*)
*)
(*****

Program SetFormat;

Var DpbAddr: Integer; (* DPB-Adresse *)
    Loop: Integer; (* Schleifenzaehler *)
    Which: Char; (* Entscheidungsvariable *)
    BondwellDpb: String[25]; (* Array fuer Bondwell-DPB *)
    AlphatronDpb: String[25]; (* Array fuer Alphatronic-DPB *)
    SchneiderDpb: String[25]; (* Array fuer Schneider-DPB *)
    NewDpb: String[25]; (* Hilfsarray fuer DPB *)

Begin
  Inline($1E/$00/ (* LD E,0 MVI E,0 *)
    $0E/$0E/ (* LD C,14 MVI C,14 *)
    $CD/$05/$00/ (* CALL 5 CALL 5 *)
    $0E/$1F/ (* LD C,31 MVI C,31 *)
    $CD/$05/$00/ (* CALL 5 CALL 5 *)
    $22/DpbAddr); (* LD (DpbAddr),HL SHLD DpbAddr *)

  DpbAddr:=DpbAddr+64; (* Der zweite DPB beginnt 64 Bytes spaeter *)

  BondwellDpb:= Chr(36)+Chr(0)+Chr(4)+Chr(15)+Chr(1)+Chr(84)+Chr(0)+
    Chr(127)+Chr(0)+Chr(192)+Chr(0)+Chr(32)+Chr(0)+Chr(2)+
    Chr(0)+Chr(0)+Chr(17)+Chr($2A)+Chr($52)+Chr($E5)+Chr(1)+
    Chr(2)+Chr(0)+Chr(0)+Chr($FF);

  AlphatronDpb:=Chr(64)+Chr(0)+Chr(4)+Chr(15)+Chr(1)+Chr(151)+Chr(0)+
    Chr(127)+Chr(0)+Chr(192)+Chr(0)+Chr(32)+Chr(0)+Chr(2)+
    Chr(0)+Chr(1)+Chr(16)+Chr($2A)+Chr($52)+Chr($E5)+Chr(1)+
    Chr(2)+Chr(0)+Chr(0)+Chr($FF);

  SchneiderDpb:=Chr(36)+Chr(0)+Chr(3)+Chr(7)+Chr(0)+Chr(170)+Chr(0)+
    Chr(63)+Chr(0)+Chr(192)+Chr(0)+Chr(16)+Chr(0)+Chr(2)+
    Chr(0)+Chr($41)+Chr(9)+Chr($2A)+Chr($52)+Chr($E5)+Chr(2)+
    Chr(4)+Chr(0)+Chr(0)+Chr(0);

  ClrScr;
  WriteLn;
  WriteLn(' Fremdformate auf dem Schneider-CPC');
  WriteLn(' -----');
  WriteLn;
  WriteLn;
  WriteLn(' 0 - Ende');
  WriteLn;
  WriteLn(' 1 - Bondwell-Format');
  WriteLn;
  WriteLn(' 2 - Alphatronic-PC');
  WriteLn;
  WriteLn(' 3 - Schneider-Systemformat');

```

```

WriteLn;
WriteLn;
Write (' Bitte waehlen Sie: ');

Repeat

    Read(Kbd,Which);

Until Which.In (.'0'..'3'.);

Case Which Of
    '0': Begin                                (* Taste 0: Programm beenden *)
        ClrScr;
        Exit
    End;
    '1': NewDpb:=BondwellDpb;    (* Taste 1: Bondwell-DPB      *)
    '2': NewDpb:=AlphatronDpb;  (* Taste 2: Alphatron-DPB *)
    '3': NewDpb:=SchneiderDpb;  (* Taste 3: Schneider-DPB *)
End; (* Case *)

Write(Which);

For Loop:=0 To 24 Do
    Mem[DpbAddr+Loop]:=Ord(NewDpb[Loop+1]);

ClrScr;

End.

```

- Wenn Sie mit der Eingabe des Listings fertig sind, können Sie Control-KD drücken und gelangen wieder in die Kommando-Ebene des Turbo-Systems.

- Dort wählen Sie "O" für "Options" und selektieren "C" zur Erzeugung von "Com-Files". Mit "Q" (Quit) landen Sie wieder in der Kommando-Ebene.

- Hier compilieren Sie das Programm mit "C" und drücken danach "Q" für Quit. Sie sind wieder im CCP (Console Command Processor) des CP/M-Betriebssystems. Auf der Diskette steht jetzt ein Programm SF.COM, das ebenfalls durch Eingabe von "SF" geladen und gestartet wird.

Falls Sie sich auch für die Funktionsweise der beiden Programme interessieren, finden Sie hier alle benötigten Informationen. Gehen wir zuerst auf das Turbo-Pascal-Programm ein, das wohl erheblich leichter zu verstehen ist.

SetFormat verwendet die folgenden Variablen: DpbAddr ist eine Integervariable, die einen Zeiger auf den Disk-Parameter-Block speichert. Loop dient als Hilfsvariable in einer Schleife. Which ist eine Zeichenvariable, die die Tastatureingabe entgegennimmt und auswertet. Die Strings BondwellDpb, AlphatronDpb und SchneiderDpb enthalten die jeweiligen DPB-Daten, die typisch für ein Diskettenformat sind. Beim Umkopieren der Blöcke werden sie

zuerst in NewDpb übertragen, eine weitere Zeichenkette.

Das Programm selbst startet mit einer Inline-Anweisung, die Maschinencode-Bytes ins Programm einbindet. Dieses kleine Maschinenprogramm holt mit Hilfe zweier BDOS-Funktionen einen Zeiger auf den DPB für das Laufwerk A: in die Variable DpbAddr. Das geht so: LD E,0 und LD C,14 sowie CALL 5 bestimmen, daß das Laufwerk A: verwendet werden soll. Würde im E-Register eine Eins stehen, dann würde das Betriebssystem das B-Laufwerk auswählen. Mit LD C,31 und CALL 5 holt der Computer die Adresse des DPB-Blocks für das Erstlaufwerk ins HL-Register. Mit LD (DpbAddr),HL sichert das Programm den Wert in der Variablen DpbAddr.

Wie wir bereits vorhin erkannt haben, liegen die beiden DPB-Blöcke genau 64 Bytes auseinander.

Die Zeile "DpbAddr:=DpbAddr+64" läßt also die Variable auf den zweiten Disk-Parameter-Block zeigen.

Mit Stringverkettungen von einzelnen Zeichen (Characters, Chr) setzt der Computer in die Datenfelder BondwellDpb, AlphasatDpb und SchneiderDpb die 25 Bytes ein, aus denen jeder DPB-Block besteht.

Durch einen ClrScr (ClearScreen, löscht den Bildschirm) und einige WriteLn-Anweisungen besorgt der Computer eine recht ansprechende Bildschirmausgabe. Read(Kbd,Which) fragt die Tastatur ab, ohne auf das Drücken der Enter-Taste zu warten. "Kbd" steht hier für "Keyboard", auf Deutsch "Tastatur". Mit der Repeat-Until-Konstruktion wird dafür gesorgt, daß der Computer in der Eingabeschleife verharret, bis ein korrektes Zeichen entdeckt wird.

In der folgenden Case-Of-Anweisung prüft der Computer den Code des Zeichens und reagiert entsprechend. War Ihre Eingabe die Taste "0", löscht das Programm den Bildschirm und kehrt nach CP/M zurück (ClrScr und Exit). Bei einem Druck auf die Tasten "1" bis "3" wird der jeweilige DPB-Block in den String NewDpb kopiert. Write(Which) zeigt auf dem Bildschirm Ihre Eingabe nochmals an. Durch die For-To-Do-Schleife POKet das Programm den Stringinhalt in den DPB des Laufwerks B:, indem es aus dem String mit der CHR-Funktion einzelne Zeichen herausnimmt und ins Integer-Format konvertiert. Über den Pseudo-Array MEMAAdresseÜ liest der Computer die Daten in den DPB. Nach dem Löschen des Bildschirm wird das Programm auch hier beendet.

Das Assembler-Programm funktioniert nicht viel anders, ist aber aufgrund der Unübersichtlichkeit von Maschinenprogrammen im allgemeinen schwerer zu verstehen. Versuchen wir es trotzdem: SF.ASM beginnt bei der Marke START. Dort wird der Prozessor-Stackpointer auf einen programmeigenen Stapelspeicher hin geladen.

Beim Label DRIVE\$A setzt der Computer durch Aufruf der BDOS-Funktion 14 einen internen Vermerk, daß alle diskettenbezogenen Funktionen auf das Laufwerk A: wirken sollen.

Das ist nötig, da ein Diskettenzugriff auf eine Fremddiskette im B-Laufwerk unter Garantie einen BDOS-Error "Bad Sector" hervor-

ruft. Ab SELECT wird die Tastatur abgefragt, das Betriebssystem liefert den ASCII-Code des eingegebenen Zeichens im Akkumulator ab. In IF\$CR, IF\$1, IF\$2 und IF\$3 kann mit CPI-Befehlen ("Compare Immediate") die Eingabe überprüft werden. Wenn Sie nur <ENTER> drücken, springt das Programm zur Adresse TEXTLST. Dort wird der Bildschirm-Modus 2 eingeschaltet, der ja unter CP/M die übliche Bildschirmdarstellung ist, und das Programm löst einen Warmstart aus. Beim Drücken der Tasten <1> bis <3> wird stattdessen die gewünschte Format-Routine aufgerufen. Wurde eine andere Taste gedrückt, gibt der Computer einen Warnton aus, bewegt den Cursor wieder nach links (aber nur dann, wenn das Zeichen kein Control-Code ist, sondern wirklich auf dem Bildschirm dargestellt wurde), und kehrt in die Tastaturabfrage zurück.

An den Adressen SET\$TA ("Setze Triumph-Adler"), SET\$BO ("Setze Bondwell") und SET\$CPC ("Setze CPC-Format") steht ein Befehl, der einen Zeiger auf den entsprechenden neuen DPB in das HL-Register lädt (LXI H,...). Beim Label CONTINUE geht es weiter: Hier meldet das BDOS des CP/M-System über die Funktion 31 (GETDPB) die Adresse des DPBs von Laufwerk A: im HL-Register. Die vier darauf folgenden Befehle addieren die Zahl 64 zu dieser Adresse. Der zweite DPB liegt bekanntlich genau 64 Bytes hinter dem ersten. Die beiden DB-Direktiven ("Define Byte") simulieren den Z80-Befehl LDIR, den der 8080-Assembler ASM.COM natürlich nicht so ohne weiteres besteht. Der Trick besteht einfach darin, die Objekt-codes als Bytes ins Programm einzusetzen. Vorher muß aber noch das BC-Doppelregister mit dem Wert 25 geladen werden, da LDIR 25 Bytes umkopieren soll.

Im Programm folgen als DB- und DW-Pseudobefehle die DPB-Blöcke der verschiedenen implementierten Formate und danach ab MESSAG1 die Textmeldungen des Programms, zuerst das Titelbild, dann der Schlußstring. Ganz am Ende darf natürlich die Definition des Programmstacks nicht fehlen.

Sie haben jetzt alle Möglichkeiten in der Hand, fremde Diskettenformate anderer CP/M-Computer zu analysieren und auf Ihrem Schneider-CPC zu lesen und zu beschreiben. Wenn Sie weitere Formate in SetFormat oder SF.ASM einbauen wollen, steht dem natürlich nichts im Wege. Beim Pascal-Programm ist das etwas leichter. Dort müssen Sie in der VAR-Definition weitere String\$250-DeklARATIONen einfügen, die Bildschirmausgabe und die Case-Of-Tastatur-decodierung erweitern. Nach den Anweisungen, die jede Menge CHR\$ enthalten, sind die neuen DPB-Daten einzusetzen.

Etwas kompliziertes ist die Abänderung des Maschinenprogramms. Unmöglich ist das aber trotzdem nicht. Hier fügen Sie nach den IF\$-Labels weitere ein:

```
IF$4  CPI  '4'
      JZ   FORM4

IF$5  CPI  '5'
      JZ   FORM5

IF$6  CPI  '6'
      JZ   FORM6 etc.
```

Vor SET\$CPC können Sie die zugehörigen Sprungziele eingeben:

```
FORM4 LXI  H, TABLE4  
      JMP  CONT
```

```
FORM5 LXI  H, TABLE5  
      JMP  CONT
```

```
FORM6 LXI  H, TABLE6  
      JMP  CONT
```

Entsprechend den vorgegebenen Muster-DPBs müssen Sie weitere erstellen und nach BOND12A bzw. nach CPCTAB eintippen. Wichtig ist, daß das letzte Byte des DPBs 00FFH ist.

MESSAG1 sollten Sie um die neuen Formatnamen erweitern. Sobald Sie das Programm mit ASM und LOAD neu assembliert haben, können Sie auf die erweiterte Formatliste zurückgreifen.

Wenn Sie sich jetzt noch fragen, woher Sie preisgünstig CP/M-Software bekommen, sollten Sie in Computerzeitschriften nach Anzeigen von Firmen suchen, die Public-Domain-Software vertreiben. Public-Domain-Programme stammen hauptsächlich aus den USA und unterliegen nicht dem Urheberrechts-Schutz.

Diese Programme können beliebig kopiert und verschenkt werden. Der Verkaufspreis darf nicht wesentlich über den eigentlichen Kopierkosten liegen. Im Endeffekt dürfte Sie 5.25-Zoll-Software etwa 20 Mark pro Diskette kosten.

Von der SIG/M User Group, der CP/M-User-Group und PicoNet gibt es insgesamt mehrere hundert Disketten mit zum Teil sehr guter Software. So gibt es verschiedene Forth-Versionen, Texteditoren, jede Menge Hilfsprogramme und sogar einen LISP-Interpreter!

## 22. Funktioniert Ihr Computer richtig?

Vielleicht kennen Sie schon das Programm MEMORY CHECK. Es dient dazu, den RAM-Speicher des Computers auf seine Funktionstüchtigkeit zu überprüfen. Hier finden Sie eine verbesserte Version des Programms, nämlich MEMORY CHECK V2.0 (im folgenden einfach "MEMORY CHECK"). Das Programm läuft auf allen Schneider-CPC-Computern.

MEMORY CHECK wurde vollständig in Maschinensprache geschrieben. So können Sie entweder den Quellcode des Programms abtippen und assemblieren oder den Basic-Loader eingeben. Der Basic-Loader ruft automatisch das Programm auf, was Sie sonst mit CALL &A200 selbst tun müssen. Wenn Sie alle Daten im Speicher weiterhin benötigen, sollten Sie MEMORY &3FFF verwenden.

### 22.1 Assemblerlisting: MEMORY CHECK

```

*****
; *
; *      MEMORY CHECK V2.0 - testet den RAM-Speicher      *
; *
; *****

                ORG      &A200          ; Startadresse des Programms
                JP       MEMCHECK      ; Einsprung ins Programm
;
; Definitionen *****
;

KM_WAIT_KEY    EQU      &BB18          ; KM WAIT KEY wartet auf Tastendruck
TXT_INITIALISE EQU      &BB4E          ; TXT INITIALISE - Text-VDU init
TXT_OUTPUT     EQU      &BB5A          ; TXT OUTPUT - Bildschirmausgabe
TXT_SET_PEN    EQU      &BB90          ; TXT SET PEN - Pen-Farbe setzen
SCR_SET_BASE   EQU      &BC08          ; SCR SET BASE - Video-RAM verlegen
SCR_SET_INK    EQU      &BC32          ; SCR SET INK - Inkfarbe setzen
SCR_SET_BORDER EQU      &BC38          ; SCR SET BORDER - Randfarbe setzen

NU             EQU      0              ; Nullbyte
MO             EQU      4              ; screenMOde
LF             EQU      10             ; LineFeed
CR             EQU      13             ; CarriageReturn
IV             EQU      24             ; InVert
SC             EQU      31             ; SetCursor
SE             EQU      32             ; SpacE

;
; Bildschirmmeldungen *****
;

TEXT1          DEFB      MO,1,LF,SE,IV,SE,SE,SE,SE,SE,SE
                DEFB      'MEMORY CHECK V2.0'
                DEFB      SE,SE,SE,SE,SE,SE,IV,CR,LF,NU
TEXT2          DEFB      SC,2,20
                DEFB      'Speicher von &0040 bis &A000 Ok'
                DEFB      CR,LF,NU

```

```

TEXT3      DEFB      SC,2,22
           DEFB      'Speicher von &C000 bis &FFFF Ok'
           DEFB      CR,LF,LF

TEXT4      DEFB      SE
           DEFB      'Bitte eine Taste druecken!'
           DEFB      NU

TEXT5      DEFB      MO,1,NU

ERRMSG     DEFB      CR,LF,LF,SE
           DEFB      'Adre',177,'fehler in'
           DEFB      SE,NU

CRLF       DEFB      CR,LF,LF,NU

CURSET     DEFB      SC,18,10,NU      ; Entspricht LOCATE 18,10

;
; Programmcode *****
;

MEMCHECK   CALL      TXT_INITIALISE ; Text-VDU initialisieren
           LD        B,10           ; Erste Farbe "Blaugrün"
           LD        C,10           ; Zweite Farbe "Blaugrün"
           CALL      SCR_SET_BORDER ; Border-Farbe setzen

INK_0      LD        A,0            ; INK 0,
           LD        B,13           ; ... 13,
           LD        C,13           ; ... 13
           CALL      SCR_SET_INK    ; Ink-Farbe setzen

INK_1      LD        A,1            ; INK 1,
           LD        B,0            ; ... 0,
           LD        C,0            ; ... 0
           CALL      SCR_SET_INK    ; Wiederum Ink-Farbe setzen

PEN_1      LD        A,1            ; PEN 1
           CALL      TXT_SET_PEN    ; Ans Betriebssystem melden

MESSAGES   LD        HL,TEXT1      ; Zeiger auf Text 1
           CALL      PRINT          ; "MEMORY CHECK V2.0" ausgeben
           DI                     ; Interrupts verbieten

CHECK_1ST  LD        HL,&0040       ; Ende des RST-Bereichs
           LD        DE,TEXT1      ; Programmanfang
           CALL      CHECK_MEM      ; Von (HL) bis (DE) prüfen
           LD        HL,TEXT2      ; Zeiger auf Text 2
           CALL      PRINT          ; "Speicher ... Ok" ausgeben

SCREEN_MOVE LD        HL,&C000      ; Originaler Bildschirmfang
           LD        DE,&4000      ; Neuer Bildschirmspeicher
           LD        BC,&4000      ; Bildschirmgröße 16 KByte
           LDIR                    ; Bildschirm umkopieren

SCREEN_SET  LD        A,&40         ; Neue Startadresse (Highbyte)

```



```

CALL      SCR_SET_BASE    ; Bildschirmbasis setzen

CHECK_2ND  LD      HL,&C000    ; Anfang des 2. Testbereichs
          LD      DE,&FFFA    ; Ende des 2. Testbereichs
          CALL    CHECK_MEM   ; Von (HL) bis (DE) pruefen
          LD      HL,TEXT3    ; Zeiger auf Text 3
          CALL    PRINT       ; "Speicher ... Ok" ausgeben
          CALL    KM_WAIT_KEY ; Auf Tastendruck warten

BAS_RETURN LD      HL,TEXT5    ; Dort stehen Steuerzeichen, die
          CALL    PRINT       ; MODE 2 einschalten
          RET              ; Rücksprung nach Basic

;
; CHECK_MEM testet von (HL) bis (DE) *****
;

CHECK_MEM  LD      C,(HL)      ; Adreßinhalt sichern
          SUB     A            ; Akku löschen
          LD      (HL),A       ; In (HL) alle Bits löschen
          LD      A,(HL)       ; Wert wieder einlesen
          CP      0            ; Auf Richtigkeit prüfen
          JR      NZ,ERROR     ; Wenn nicht Null - Fehlermeldung

          LD      A,&FF         ; Akku auf logisch Eins
          LD      (HL),A       ; In (HL) alle Bits setzen
          LD      A,(HL)       ; Wert wieder einlesen
          CP      &FF          ; Auf Richtigkeit prüfen
          JR      NZ,ERROR     ; Wenn nicht &FF - Fehlermeldung
          LD      (HL),C       ; Adreßinhalt wiederherstellen

SHOW?     LD      A,L          ; Highbyte des Adreßzeigers
          CP      0            ; Page-Grenze erreicht?
          JR      NZ,NEXT      ; Nein - weiterhin prüfen

HEXADR    PUSH    HL           ; HL-Register sichern
          LD      HL,CURSET    ; Zeiger auf LOCATE-String
          CALL    PRINT        ; Cursor positionieren
          POP     HL           ; HL-Register wiederherstellen

          PUSH    HL           ; Und wiederum sichern
          CALL    HEXOUT       ; Adresse als Hexziffer ausgeben
          POP     HL           ; HL-Register wiederherstellen

NEXT      INC      HL          ; Adreßzeiger erhöhen
          CALL    CP_HL_DE      ; CP HL,DE - Speicherende erreicht?
          JR      NZ,CHECK_MEM ; Nein, weitermachen
          RET              ; Rücksprung

; ERROR wird im Fehlerfall angesprochen *****
;

ERROR     LD      (HL),C       ; Adreßinhalt wiederherstellen
          LD      A,7          ; Fehler - BELL-Code laden
          CALL    TXT_OUTPUT    ; Und ausgeben
          DI              ; Interrupts wieder unterdrücken
          PUSH    HL           ; HL-Register sichern

```

```

LD      HL,ERRMSG      ; Zeiger auf "Adreßfehler in"
CALL    PRINT          ; Diesen String ausgeben
POP     HL             ; HL-Register wiederherstellen
PUSH    HL             ; Auf dem Stack sichern
CALL    HEXOUT         ; Adresse als Hexzahl ausgeben
LD      HL,CRLF        ; Zeiger auf CR/LF/LF-String
CALL    PRINT          ; Ausgeben
LD      HL,TEXT4       ; Zeiger auf "Bitte Taste drücken!"
CALL    PRINT          ; Ausgeben
CALL    KM_WAIT_KEY    ; Auf Tastendruck warten
POP     HL             ; HL wiederherstellen
JR      NEXT           ; Rücksprung zum Hauptprogramm

;
; CP_HL_DE simuliert CP HL,DE *****
;

CP_HL_DE    PUSH    HL      ; HL-Register sichern
            OR      A       ; Macht aus SBC ein echtes SUB
            SBC     HL,DE   ; HL=HL-DE, Flags ausrichten
            POP     HL      ; HL-Wert zurückholen
            RET           ; Rücksprung

;
; PRINT druckt von (HL) bis (HL)=0 *****
;

.PRINT      PUSH    AF      ; Akku und Flags retten

PTLOOP     LD      A,(HL)   ; Akku mit ASCII-Zeichen laden
            CP      0       ; Mit Null vergleichen
            JR      Z,PTRETN ; Bei A=0 Stringende
            CALL    TXT_OUTPUT ; Sonst Bildschirmausgabe
            DI          ; Interrupts wieder verbieten
            INC     HL      ; Adreßzeiger erhöhen
            JR      PTLOOP  ; Und Rücksprung in die Schleife

PTRETN     POP     AF      ; Akku und Flags wiederherstellen
            RET           ; Rücksprung

;
; HEXOUT gibt die Hexadezimalzahl in HL aus *****
;

HEXOUT     LD      A,91     ; ASCII-Code für eckige Klammer
            CALL    TXT_OUTPUT ; Ausgeben
            DI          ; Interrupts verbieten

HEX_HIGHBYTE LD      A,H    ; Akku mit erstem Byte laden
            CALL    SHIFT   ; Shift und als Hexziffer ausgeben
            LD      A,H    ; Akku mit erstem Byte laden
            CALL    PRT     ; Als Hexziffer ausgeben

HEX_LOWBYTE LD      A,L    ; Akku mit zweitem Byte laden
            CALL    SHIFT   ; Shift und als Hexziffer ausgeben
            LD      A,L    ; Akku mit zweitem Byte laden
            CALL    PRT     ; Als Hexziffer ausgeben

```

```

LD      A,93      ; ASCII-Code für eckige Klammer
CALL    TXT_OUTPUT ; Ausgeben
DI      ; Interrupts verbieten
RET     ; Rücksprung

SHIFT   RRA       ; Viermal je ein Bit aus dem rechten
          RRA      ; Nibble des Akkus herausshiften, so
          RRA      ; daß nur das linke Nibble übrig-
          RRA      ; bleibt

PRT     AND        &0F      ; Linkes Nibble des Akkus ausblenden
          ADD      A,48     ; ASCII-Offset addieren
          CP       58       ; Größer als "9"?
          JR       C,SHOW   ; Wenn nicht, direkt ausgeben
          ADD      A,7       ; Sonst "A"- "9" addieren

SHOW    CALL       TXT_OUTPUT ; Ausgeben
          RET       ; Rücksprung
          END
; -----

```

Hier der Basic-Loader mit dem MEMCHECK-Programm in DATA-Zeilen.

## 22.2 Listing: MEMCHECK.BAS

```

100 ' *****
110 ' *
120 ' *      MEMORY CHECK - V2.0
130 ' *
140 ' *****
150 '
160 FOR i=41472 TO 41889:READ a
170 sum=sum+a:NEXT i
180 IF sum=41256 THEN 210
190 PRINT CHR$(7); "Fehler in den DATA-Zeilen!"
200 LIST 280-:END
210 hi=HIMEM
220 MEMORY &3FFF:RESTORE
230 FOR i=41472 TO 41889
240 READ a:POKE i,a:NEXT i
250 CALL &A200 ' MEMORY CHECK anspringen
260 MEMORY hi ' Alten HIMEM wiederherstellen
270 ' Maschinencode-Daten -----
280 DATA &C3,&AE,&A2,&04,&01,&0A,&20,&18,&20,&20,&20,&20,&20,&4D,&45
290 DATA &4D,&4F,&52,&59,&20,&43,&48,&45,&43,&4B,&20,&56,&32,&2E,&30,&20
300 DATA &20,&20,&20,&20,&20,&1B,&0D,&0A,&00,&1F,&02,&14,&53,&70,&65,&69
310 DATA &63,&68,&65,&72,&20,&76,&6F,&6E,&20,&26,&30,&30,&34,&30,&20,&62
320 DATA &69,&73,&20,&26,&41,&30,&30,&30,&20,&4F,&6B,&0D,&0A,&00,&1F,&02
330 DATA &16,&53,&70,&65,&69,&63,&68,&65,&72,&20,&76,&6F,&6E,&20,&26,&43
340 DATA &30,&30,&30,&20,&62,&69,&73,&20,&26,&46,&46,&46,&46,&20,&4F,&6B
350 DATA &0D,&0A,&0A,&20,&42,&69,&74,&74,&65,&20,&65,&69,&6E,&65,&20,&54
360 DATA &61,&73,&74,&65,&20,&64,&72,&75,&65,&63,&6B,&65,&6E,&21,&00,&04
370 DATA &01,&00,&0D,&0A,&0A,&20,&41,&64,&72,&65,&B1,&66,&65,&68,&6C,&65
380 DATA &72,&20,&69,&6E,&20,&00,&0D,&0A,&0A,&00,&1F,&12,&0A,&00,&CD,&4E
390 DATA &BB,&06,&0A,&0E,&0A,&CD,&38,&BC,&3E,&00,&06,&0D,&0E,&CD,&32

```

```

400 DATA &BC,&3E,&01,&06,&00,&0E,&00,&CD,&32,&BC,&3E,&01,&CD,&90,&BB,&21
410 DATA &03,&A2,&CD,&63,&A3,&F3,&21,&40,&00,&11,&03,&A2,&CD,&0E,&A3,&21
420 DATA &29,&A2,&CD,&63,&A3,&21,&00,&C0,&11,&00,&40,&01,&00,&40,&ED,&B0
430 DATA &3E,&40,&CD,&08,&BC,&21,&00,&C0,&11,&FA,&FF,&CD,&0E,&A3,&21,&4E
440 DATA &A2,&CD,&63,&A3,&CD,&18,&BB,&21,&8F,&A2,&CD,&63,&A3,&C9,&4E,&97
450 DATA &77,&7E,&FE,&00,&20,&22,&3E,&FF,&77,&7E,&FE,&FF,&20,&1A,&71,&7D
460 DATA &FE,&00,&20,&0D,&E5,&21,&AA,&A2,&CD,&63,&A3,&E1,&E5,&CD,&72,&A3
470 DATA &E1,&23,&CD,&5D,&A3,&20,&D7,&C9,&71,&3E,&07,&CD,&5A,&BB,&F3,&E5
480 DATA &21,&92,&A2,&CD,&63,&A3,&E1,&E5,&CD,&72,&A3,&21,&A6,&A2,&CD,&63
490 DATA &A3,&21,&73,&A2,&CD,&63,&A3,&CD,&18,&BB,&E1,&18,&D4,&E5,&B7,&ED
500 DATA &52,&E1,&C9,&F5,&7E,&FE,&00,&20,&07,&CD,&5A,&BB,&F3,&23,&18,&F4
510 DATA &F1,&C9,&3E,&5B,&CD,&5A,&BB,&F3,&7C,&CD,&8F,&A3,&7C,&CD,&93,&A3
520 DATA &7D,&CD,&8F,&A3,&7D,&CD,&93,&A3,&3E,&5D,&CD,&5A,&BB,&F3,&C9,&1F
530 DATA &1F,&1F,&1F,&E6,&0F,&C6,&30,&FE,&3A,&38,&02,&C6,&07,&CD,&5A,&BB
540 DATA &C9,&00
550 END ' -----

```

Das Maschinenprogramm wählt den Bildschirm-Modus 1 und sucht geeignete Farben aus. Als Startmeldung gibt es "MEMORY CHECK V2.0" aus und beginnt sofort mit der Überprüfung des Speichers. Dieser Test ist in zwei Teile aufgeteilt. Im ersten Durchgang wird der Speicher von &0040 bis &A200 (64 bis 41472) geprüft, im zweiten Durchgang von &C000 bis &FFFF (49152 bis 65535).

MEMORY CHECK zeigt während des Tests in eckigen Klammern und hexadezimaler Schreibung die geprüften Adressen an. Da die Ausgabe aller Adressen die Arbeitsgeschwindigkeit des Programms gewaltig reduzieren würde, geschieht dies nur alle 256 Bytes und damit immer nur dann, wenn sich das Highbyte der Adresse verändert hat. Wir wünschen Ihnen natürlich, daß bei Ihnen die beiden Meldungen "Speicher von &0040 bis &A000 Ok" und "Speicher von &C000 bis &FFFF Ok" ausgegeben werden. Es kann aber auch sein, daß "Adreßfehler in ..." gemeldet wird. In diesem Fall können Sie durch Drücken einer beliebigen Taste den Speichertest fortsetzen, um so den defekten Bereich einzugrenzen. Die jetzt fällige Reparatur des Computers dürfte etwas billiger als sonst sein, da Sie dem Händler ja die genaue Fehlerquelle mit exakter Beschreibung liefern können.

Im Gegensatz zur Version 1.0 von MEMORY CHECK ist eine Rückkehr nach Basic möglich, weil die Speicherinhalte nicht mehr zerstört, sondern nur kurzfristig überschrieben und dann wiederhergestellt werden.

Und so arbeitet MEMORY CHECK: Es setzt zuerst durch den Betriebssystem-Aufruf von &BB4E die Textausgabe-Routinen zurück. An dieser Adresse steht nämlich ein Sprung nach TXT INITIALISE.

Die drei darauf folgenden Befehle bestimmen die Farbe des Bildschirmrands. "Blaugrün" sieht ganz angenehm aus. Im Programmblock INK\_0 setzt der Computer die Ink-Farbe 0 auf 13 (Weiß), in INK\_1 die Ink-Farbe 1 auf 0 (Schwarz). Mit LD A,1 und CALL TXT\_SET\_PEN wird die Schriftfarbe auf 1 festgelegt. MESSAGES gibt die Startmeldung "MEMORY CHECK V2.0" aus. Dieser Text steht unter dem Label TEXT1.

CHECK\_1ST prüft den ersten Speicherblock. Dazu wird das HL-Register mit der Startadresse &0040 geladen. Im Bereich &0000 bis &003F stehen die Restart-Vektoren, die die RAM/ROM-Auswahl des Computers besorgen. Sie dürfen keinesfalls verändert werden, weil sonst der Computer unter Garantie beim nächsten Interrupt oder Betriebssystem-Aufruf abstürzen würde. Die Endadresse im DE-Register wird durch TEXT1 repräsentiert. TEXT1 ist ja die erste Adresse des Programms. Mit CALL CHECK\_MEM wird die - später noch erklärte - Routine aufgerufen, die den Speicher prüft. Wenn der Computer aus diesem Unterprogramm zurückkehrt, ist der Speichertest erfolgreich verlaufen, und MEMORY CHECK kann eine positive Meldung ausgeben: "Speicher von &0040 bis &A200 Ok".

Da im nächsten Durchlauf der Video-RAM getestet wird, würde das ein ziemlich unschönes Durcheinander auf dem Bildschirm verursachen. Also verschiebt MEMORY CHECK den Bildschirmspeicher mit LDIR nach &4000 und teilt dem Betriebssystem über SCR SET BASE diese Änderung mit. Aus diesem Grund sollte HIMEM bei &3FFF liegen. Nach Abarbeitung des Programms kann HIMEM wieder nach oben verlegt werden.

In der Folge beginnt MEMORY CHECK mit der Prüfung des zweiten Speicherblocks. Der Aufruf von CHECK\_MEM ist identisch mit dem vorherigen, nur mit dem Unterschied, daß im HL- und im DE-Register andere Adressen stehen. Wenn alles geklappt hat, gibt der Computer "Speicher von &C000 bis &FFFF Ok" aus und wartet auf einen Tastendruck von Ihnen. Danach erfolgt der Rücksprung nach Basic.

CHECK\_MEM ist die wichtigste Routine des Programms. Hier wird der Speicher überprüft. Dazu liest die Routine den Wert der durch HL adressierten Speicherstelle ins C-Register und merkt sich diesen dort. In der Speicherstelle wird der Wert Null eingetragen, das heißt, daß alle Bits gelöscht werden. Sofort darauf wird der Wert wieder eingelesen und mit Null verglichen.

Trifft die Erwartung nicht zu, daß die Werte identisch sind, springt das Programm nach ERROR. Andernfalls setzt es alle Bits in der Speicherstelle und liest sie wieder aus. Hier sollte der Wert 255 zurückgemeldet werden. Auch hier führt ein Fehler direkt zur Marke ERROR. Wenn kein Fehler in der Speicherstelle gefunden wurde, stellt LD (HL),C deren alten Inhalt wieder her.

In SHOW? stellt der Computer fest, ob eine 256-Bytes-Grenze erreicht wurde. In diesem Fall ist ja der Inhalt des L-Registers 0.

Der Cursor wird mittels eines Steuercode-Strings positioniert, und CALL HEXOUT gibt die Zahl, die im HL-Doppelregister steht, auf dem Bildschirm aus. Schließlich wird der Adreßzeiger erhöht.

Um zu bestimmen, ob das Speicherende erreicht ist, muß der Inhalt des HL-Registers mit dem des DE-Registers verglichen werden. Dies geschieht durch die Routine CP\_HL\_DE. Solange der Vergleich negativ ausfällt, also das Zero-Flag der Z80-CPU gelöscht ist, kann der Computer die Programmschleife weiter abarbeiten. Ist das Flag gesetzt, erfolgt ein Rücksprung nach Basic.

Wurde ein Fehler entdeckt, stellt der Computer zuerst einmal den Inhalt der Speicherstelle wieder her, obwohl dies aus naheliegenden Gründen eigentlich überflüssig ist ...

Dann gibt er einen Warnton aus und meldet die fehlerhafte Adresse. Er wartet auf einen Tastendruck und kehrt in die Hauptschleife zurück.

Den Besitzern eines CPC-6128 ist vom Eintippen des Listings abzuraten. Sie sollten sich statt dessen für "MEMCHECK 6128" interessieren. Dies ist eine erweiterte Version von MEMORY CHECK V2.0, die zusätzlich die Speichererweiterung in den Test mit einbezieht. Das Programm verwendet zur Umschaltung auf die zweite 64K-Bank OUT-Befehle. Wie das genau funktioniert und was sich damit noch alles anstellen läßt, ist das Thema eines anderen Kapitels dieses Buches: Das dort vorgestellte Programm heißt "SYNOPSIS 6128".

Das Quellcode-Listing von MEMCHECK 6128 hat eine nicht abzustreitende Ähnlichkeit mit dem des normalen Speicherprüf-Programms ...

### 22.3 Assemblerlisting: MEMORY CHECK 6128

```
; *****
; *
; *      MEMORY CHECK 6128 - testet den RAM-Speicher      *
; *      ----      *
; *****

                ORG      &A200      ; Startadresse des Programms
                JP       MEMCHECK   ; Einsprung ins Programm

;
; Definitionen *****
;

KM_WAIT_KEY    EQU      &BB16      ; KM WAIT KEY wartet auf Tastendruck
TXT_INITIALISE EQU      &BB4E      ; TXT INITIALISE - Text-VDU init
TXT_OUTPUT     EQU      &BB5A      ; TXT OUTPUT - Bildschirmausgabe
TXT_SET_PEN    EQU      &BB90      ; TXT SET PEN - Pen-Farbe setzen
SCR_SET_BASE   EQU      &BC08      ; SCR SET BASE - Video-RAM verlegen
SCR_SET_INK    EQU      &BC32      ; SCR SET INK - Inkfarbe setzen
SCR_SET_BORDER EQU      &BC38      ; SCR SET BORDER - Randfarbe setzen

NU             EQU      0          ; Nullbyte
MO             EQU      4          ; screenMode
LF             EQU      10         ; LineFeed
CR             EQU      13         ; CarriageReturn
IV             EQU      24         ; InVert
SC             EQU      31         ; SetCursor
SE             EQU      32         ; SpaceE

;
; Bildschirmmeldungen *****
;
```

```

TEXT1      DEFB      MO,1,LF,SE,IV,SE,SE,SE,SE,SE,SE
           DEFB      'MEMORY CHECK V2.0'
           DEFB      SE,SE,SE,SE,SE,SE,SE,IV,CR,LF,NU

TEXT2      DEFB      SC,2,12
           DEFB      'Speicher von &0040 bis &A000 Ok'
           DEFB      CR,LF,NU

TEXT3      DEFB      SC,2,14
           DEFB      'Speicher von &C000 bis &FFFF Ok'
           DEFB      CR,LF,NU

TEXT4      DEFB      SC,2,16
           DEFB      'Erste Erweiterungsbank Ok'
           DEFB      CR,LF,NU

TEXT5      DEFB      SC,2,18
           DEFB      'Zweite Erweiterungsbank Ok'
           DEFB      CR,LF,NU

TEXT6      DEFB      SC,2,20
           DEFB      'Dritte Erweiterungsbank Ok'
           DEFB      CR,LF,NU

TEXT7      DEFB      SC,2,22
           DEFB      'Vierte Erweiterungsbank Ok'
           DEFB      CR,LF,LF,NU

TEXT8      DEFB      SE
           DEFB      'Bitte druecken Sie eine Taste!'
           DEFB      NU

TEXT9      DEFB      MO,1,NU

ERRMSG     DEFB      CR,LF,LF,SE
           DEFB      'Adre',177,'fehler in'
           DEFB      SE,NU

CRLF       DEFB      CR,LF,LF,NU

CURSET     DEFB      SC,18,10,NU      ; Entspricht LOCATE 18,10
; Programmcode *****
;

MEMCHECK   CALL      TXT_INITIALISE ; Text-VDU initialisieren
           LD        B,10           ; Erste Farbe "Blaugrün"
           LD        C,10           ; Zweite Farbe "Blaugrün"
           CALL      SCR_SET_BORDER ; Border-Farbe setzen

INK_0      LD        A,0            ; INK 0,
           LD        B,13           ; ... 13,
           LD        C,13           ; ... 13
           CALL      SCR_SET_INK    ; Ink-Farbe setzen

INK_1      LD        A,1            ; INK 1,
           LD        B,0            ; ... 0,
           LD        C,0            ; ... 0

```

```

CALL      SCR_SET_INK      ; Wiederum Ink-Farbe setzen

PEN_1     LD      A,1       ; PEN 1
CALL      TXT_SET_PEN     ; Ans Betriebssystem melden

MESSAGES  LD      HL,TEXT1  ; Zeiger auf Text 1
CALL      PRINT           ; "MEMORY CHECK V2.0" ausgeben
DI        ; Interrupts verbieten

CHECK_1ST LD      HL,&0040   ; Ende des RST-Bereichs
LD        DE,TEXT1        ; Programmstart
CALL      CHECK_MEM       ; Von (HL) bis (DE) prüfen
LD        HL,TEXT2        ; Zeiger auf Text 2
CALL      PRINT           ; "Speicher ... Ok" ausgeben

SCREEN_MOVE LD      HL,&C000  ; Originaler Bildschirmstart
LD        DE,&4000          ; Neuer Bildschirmspeicher
LD        BC,&4000          ; Bildschirmgröße 16 KByte
LDIR      ; Bildschirm umkopieren

SCREEN_SET LD      A,&40     ; Neue Startadresse (Highbyte)
CALL      SCR_SET_BASE    ; Bildschirmbasis setzen

CHECK_2ND LD      HL,&C000   ; Anfang des 2. Testbereichs
LD        DE,&FFFA          ; Ende des 2. Testbereichs
CALL      CHECK_MEM       ; Von (HL) bis (DE) prüfen
LD        HL,TEXT3        ; Zeiger auf Text 3
CALL      PRINT           ; "Speicher ... Ok" ausgeben

SCREEN_MOVEBACK LD      HL,&4000 ; Von &4000
LD        DE,&C000          ; Nach &C000
LD        BC,&4000          ; 16 Kilobytes
LDIR      ; kopieren

SCREEN_RESET LD      A,&C0   ; Bildschirm wieder nach &C000
CALL      SCR_SET_BASE    ; Mit SCR SET BASE

; Speichererweiterung des 6128 testen *****
;

BANK_1     LD      BC,&7FC4   ; 1. Speicherbank einblenden
OUT        (C),C
LD        HL,&4000           ; Startadresse
LD        DE,&7FFF           ; Endadresse
CALL      CHECK_MEM         ; Von (HL) bis (DE) prüfen
LD        HL,TEXT4          ; Zeiger auf Text 4
CALL      PRINT             ; "Erste Erweiterungsbank Ok"

BANK_2     LD      BC,&7FC5   ; 2. Speicherbank einblenden
OUT        (C),C
LD        HL,&4000           ; Startadresse
LD        DE,&7FFF           ; Endadresse
CALL      CHECK_MEM         ; Von (HL) bis (DE) prüfen
LD        HL,TEXT5          ; Zeiger auf Text 5
CALL      PRINT             ; "Zweite Erweiterungsbank Ok"

```



```

BANK_3      LD      BC,&7FC6      ; 3. Speicherbank einblenden
            OUT      (C),C
            LD      HL,&4000      ; Startadresse
            LD      DE,&7FFF      ; Endadresse
            CALL    CHECK_MEM    ; Von (HL) bis (DE) prüfen
            LD      HL,TEXT6     ; Zeiger auf Text 6
            CALL    PRINT        ; "Dritte Erweiterungsbank Ok"

BANK_4      LD      BC,&7FC7      ; 4. Speicherbank einblenden
            OUT      (C),C
            LD      HL,&4000      ; Startadresse
            LD      DE,&7FFF      ; Endadresse
            CALL    CHECK_MEM    ; Von (HL) bis (DE) prüfen
            LD      HL,TEXT7     ; Zeiger auf Text 7
            CALL    PRINT        ; "Vierte Erweiterungsbank Ok"

BANK_RESTORE LD      BC,&7FC0     ; Normale Konfiguration
            OUT      (C),C

;
; Basic-Rücksprung *****
;

BAS_RETURN  LD      HL,TEXT8     ; "Bitte druecken Sie eine Taste!"
            CALL    PRINT        ; Ausgeben
            CALL    KM_WAIT_KEY  ; Auf Tastendruck warten
            LD      HL,TEXT9     ; Dort stehen Steuerzeichen, die
            CALL    PRINT        ; MODE 2 einschalten
            RET                ; Rücksprung nach Basic

;
; CHECK_MEM testet von (HL) bis (DE) *****
;

CHECK_MEM   LD      C,(HL)       ; Adreßinhalt sichern
            SUB      A           ; Akku löschen
            LD      (HL),A       ; In (HL) alle Bits löschen
            LD      A,(HL)       ; Wert wieder einlesen
            CP      0           ; Auf Richtigkeit prüfen
            JR      NZ,ERROR     ; Wenn nicht Null - Fehlermeldung

            LD      A,&FF        ; Akku auf logisch Eins
            LD      (HL),A       ; In (HL) alle Bits setzen
            LD      A,(HL)       ; Wert wieder einlesen
            CP      &FF        ; Auf Richtigkeit prüfen
            JR      NZ,ERROR     ; Wenn nicht &FF - Fehlermeldung
            LD      (HL),C       ; Adreßinhalt wiederherstellen

SHOW?      LD      A,L          ; Highbyte des Adreßzeigers
            CP      0           ; Page-Grenze erreicht?
            JR      NZ,NEXT      ; Nein - weiterhin prüfen

HEXADR     PUSH    HL           ; HL-Register sichern
            LD      HL,CURSET    ; Zeiger auf LOCATE-String
            CALL    PRINT        ; Cursor positionieren
            POP     HL          ; HL-Register wiederherstellen
    
```

```

        PUSH    HL            ; Und wiederum sichern
        CALL    HEXOUT        ; Adresse als Hexziffer ausgeben
        POP     HL            ; HL-Register wiederherstellen

NEXT     INC     HL            ; Adreßzeiger erhöhen
        CALL    CP_HL_DE      ; CP HL,DE - Speicherende erreicht?
        JR      NZ,CHECK_MEM  ; Nein, weitermachen
        RET                     ; Rücksprung

;
; ERROR wird im Fehlerfall angesprungen *****
;

ERROR     LD      (HL),C       ; Adreßinhalt wiederherstellen
        LD      A,7           ; Fehler - BELL-Code laden
        CALL    TXT_OUTPUT    ; Und ausgeben
        DI                     ; Interrupts wieder unterdrücken
        PUSH    HL            ; HL-Register sichern
        LD      HL,ERRMSG     ; Zeiger auf "Adreßfehler in"
        CALL    PRINT         ; Diesen String ausgeben
        POP     HL            ; HL-Register wiederherstellen
        PUSH    HL            ; Auf dem Stack sichern
        CALL    HEXOUT        ; Adresse als Hexzahl ausgeben
        LD      HL,CRLF       ; Zeiger auf CR/LF/LF-String
        CALL    PRINT         ; Ausgeben
        LD      HL,TEXT8      ; Zeiger auf "Bitte Taste drücken!"
        CALL    PRINT         ; Ausgeben
        CALL    KM_WAIT_KEY   ; Auf Tastendruck warten
        POP     HL            ; HL wiederherstellen
        JR      NEXT          ; Rücksprung zum Hauptprogramm

;
; CP_HL_DE simuliert CP HL,DE *****
;
CP_HL_DE  PUSH    HL            ; HL-Register sichern
        OR      A              ; Macht aus SBC ein echtes SUB
        SBC     HL,DE          ; HL=HL-DE, Flags ausrichten
        POP     HL            ; HL-Wert zurückholen
        RET                     ; Rücksprung

;
; PRINT druckt von (HL) bis (HL)=0 *****
;

.PRINT     PUSH    AF           ; Akku und Flags retten

PTLOOP    LD      A,(HL)       ; Akku mit ASCII-Zeichen laden
        CP      0              ; Mit Null vergleichen
        JR      Z,PTRETN      ; Bei A=0 Stringende
        CALL    TXT_OUTPUT    ; Sonst Bildschirmausgabe
        DI                     ; Interrupts wieder verbieten
        INC     HL            ; Adreßzeiger erhöhen
        JR      PTLOOP        ; Und Rücksprung in die Schleife

PTRETN    POP     AF           ; Akku und Flags wiederherstellen
        RET                     ; Rücksprung

```

```

;
; HEXOUT gibt die Hexadezimalzahl in HL aus *****
;
HEXOUT      LD      A,91      ; ASCII-Code für eckige Klammer
            CALL    TXT_OUTPUT ; Ausgeben
            DI      ; Interrupts verbieten

HEX_HIGHBYTE LD      A,H      ; Akku mit erstem Byte laden
            CALL    SHIFT     ; Shift und als Hexziffer ausgeben
            LD      A,H      ; Akku mit erstem Byte laden
            CALL    PRT       ; Als Hexziffer ausgeben

HEX_LOWBYTE LD      A,L      ; Akku mit zweitem Byte laden
            CALL    SHIFT     ; Shift und als Hexziffer ausgeben
            LD      A,L      ; Akku mit zweitem Byte laden
            CALL    PRT       ; Als Hexziffer ausgeben

            LD      A,93      ; ASCII-Code für eckige Klammer
            CALL    TXT_OUTPUT ; Ausgeben
            DI      ; Interrupts verbieten
            RET      ; Rücksprung

SHIFT       RRA              ; Viermal je ein Bit aus dem rechten
            RRA              ; Nibble des Akkus herausshiften, so
            RRA              ; daß nur das linke Nibble übrig-
            RRA              ; bleibt

PRT          AND      &0F     ; Linkes Nibble des Akkus ausblenden
            ADD      A,48     ; ASCII-Offset addieren
            CP       58       ; Größer als "9"?
            JR       C,SHOW    ; Wenn nicht, direkt ausgeben
            ADD      A,7       ; Sonst "A"-"9" addieren

SHOW         CALL    TXT_OUTPUT ; Ausgeben
            RET      ; Rücksprung
            END
; -----

```

Der Basic-Loader von MEMCHECK 6128.

#### 22.4 Listing: MCHE6128.BAS

```

100 ' *****
110 ' *
120 ' * MEMORY CHECK V2.0 - CPC-6128 *
130 ' *
140 ' *****
150 '
160 FOR i=41472 TO 42128:READ a
170 sum=sum+a:NEXT i
180 IF sum=64987 THEN 210
190 PRINT CHR$(7);"FEHLER IN DEN DATA-ZEILEN!"
200 LIST 280-:END

```

```

210 hi=HIMEM
220 MEMORY &3FFF:RESTORE
230 FOR i=41472 TO 42128
240 READ a:POKE i,a:NEXT i
250 CALL &A2000 ' MEMORY CHECK aufrufen
260 MEMORY hi ' HIMEM wiederherstellen
270 ' Maschinencode-DATAs -----
280 DATA &C3,&32,&A3,&04,&01,&0A,&20,&18,&20,&20,&20,&20,&20,&20,&4D,&45
290 DATA &4D,&4F,&52,&59,&20,&43,&48,&45,&43,&4B,&20,&56,&32,&2E,&30,&20
300 DATA &20,&20,&20,&20,&20,&18,&0D,&0A,&00,&1F,&02,&0C,&53,&70,&65,&69
310 DATA &63,&68,&65,&72,&20,&76,&6F,&6E,&20,&26,&30,&30,&34,&30,&20,&62
320 DATA &69,&73,&20,&26,&41,&30,&30,&30,&20,&4F,&6B,&0D,&0A,&00,&1F,&02
330 DATA &0E,&53,&70,&65,&69,&63,&68,&65,&72,&20,&76,&6F,&6E,&20,&26,&43
340 DATA &30,&30,&30,&20,&62,&69,&73,&20,&26,&46,&46,&46,&46,&20,&4F,&6B
350 DATA &0D,&0A,&00,&1F,&02,&10,&45,&72,&73,&74,&65,&20,&45,&72,&77,&65
360 DATA &69,&74,&65,&72,&75,&6E,&67,&73,&62,&61,&6E,&6B,&20,&4F,&6B,&0D
370 DATA &0A,&00,&1F,&02,&12,&5A,&77,&65,&69,&74,&65,&20,&45,&72,&77,&65
380 DATA &69,&74,&65,&72,&75,&6E,&67,&73,&62,&61,&6E,&6B,&20,&4F,&6B,&0D
390 DATA &0A,&00,&1F,&02,&14,&44,&72,&69,&74,&74,&65,&20,&45,&72,&77,&65
400 DATA &69,&74,&65,&72,&75,&6E,&67,&73,&62,&61,&6E,&6B,&20,&4F,&6B,&0D
410 DATA &0A,&00,&1F,&02,&16,&56,&69,&65,&72,&74,&65,&20,&45,&72,&77,&65
420 DATA &69,&74,&65,&72,&75,&6E,&67,&73,&62,&61,&6E,&6B,&20,&4F,&6B,&0D
430 DATA &0A,&00,&1F,&02,&18,&69,&74,&74,&65,&20,&45,&72,&75,&65,&63,&6B
440 DATA &65,&6E,&20,&63,&69,&65,&20,&65,&69,&6E,&65,&20,&54,&61,&73,&74
450 DATA &65,&21,&00,&04,&01,&00,&0D,&0A,&0A,&20,&41,&64,&72,&65,&B1,&66
460 DATA &65,&68,&6C,&65,&72,&20,&69,&6E,&20,&00,&0D,&0A,&0A,&00,&1F,&12
470 DATA &0A,&00,&CD,&4E,&BB,&06,&0A,&0E,&0A,&CD,&3B,&BC,&3E,&00,&06,&0D
480 DATA &0E,&0D,&CD,&32,&BC,&3E,&01,&06,&00,&0E,&00,&CD,&32,&BC,&3E,&01
490 DATA &CD,&09,&BB,&21,&03,&A2,&CD,&52,&A4,&F3,&21,&03,&00,&11,&03,&A2
500 DATA &CD,&FD,&A3,&21,&29,&A2,&CD,&52,&A4,&21,&00,&C0,&11,&00,&40,&01
510 DATA &00,&40,&ED,&B0,&3E,&40,&CD,&08,&BC,&21,&00,&C0,&11,&FA,&FF,&CD
520 DATA &FD,&A3,&21,&4E,&A2,&CD,&52,&A4,&21,&00,&40,&11,&00,&C0,&01,&00
530 DATA &40,&ED,&B0,&3E,&C0,&CD,&08,&BC,&01,&C4,&7F,&ED,&49,&21,&00,&40
540 DATA &11,&FF,&7F,&CD,&FD,&A3,&21,&73,&A2,&CD,&52,&A4,&01,&C5,&7F,&ED
550 DATA &49,&21,&00,&40,&11,&FF,&7F,&CD,&FD,&A3,&21,&92,&A2,&CD,&52,&A4
560 DATA &01,&C6,&7F,&ED,&49,&21,&00,&40,&11,&FF,&7F,&CD,&FD,&A3,&21,&B2
570 DATA &A2,&CD,&52,&A4,&01,&C7,&7F,&ED,&49,&21,&00,&40,&11,&FF,&7F,&CD
580 DATA &FD,&A3,&21,&D2,&A2,&CD,&52,&A4,&01,&C0,&7F,&ED,&49,&21,&F3,&A2
590 DATA &CD,&52,&A4,&CD,&18,&BB,&21,&13,&A3,&CD,&52,&A4,&C9,&4E,&97,&77
600 DATA &7E,&FE,&00,&20,&22,&3E,&FF,&77,&7E,&FE,&FF,&20,&1A,&71,&7D,&FE
610 DATA &00,&20,&0D,&E5,&21,&2E,&A3,&CD,&52,&A4,&E1,&E5,&CD,&61,&A4,&E1
620 DATA &23,&CD,&4C,&A4,&20,&D7,&C9,&71,&3E,&07,&CD,&5A,&BB,&F3,&E5,&21
630 DATA &16,&A3,&CD,&52,&A4,&E1,&E5,&CD,&61,&A4,&21,&2A,&A3,&CD,&52,&A4
640 DATA &21,&F3,&A2,&CD,&52,&A4,&CD,&18,&BB,&E1,&18,&D4,&E5,&B7,&ED,&52
650 DATA &E1,&C9,&F5,&7E,&FE,&00,&28,&07,&CD,&5A,&BB,&F3,&23,&16,&F4,&F1
660 DATA &C9,&3E,&5B,&CD,&5A,&BB,&F3,&7C,&CD,&7E,&A4,&7C,&CD,&82,&A4,&7D
670 DATA &CD,&7E,&A4,&7D,&CD,&82,&A4,&3E,&5D,&CD,&5A,&BB,&F3,&C9,&1F,&1F
680 DATA &1F,&1F,&E6,&0F,&C6,&30,&FE,&3A,&3B,&02,&C6,&07,&CD,&5A,&BB,&C9
690 DATA &00,&00
700 END ' -----

```

## 23. Trickreiches rund um die Tastatur

Der Schneider-CPC bietet in ungewöhnlich großem Umfang Möglichkeiten, die Tastatur umzudefinieren.

Da gibt es zuerst einmal den KEY DEF-Befehl, der es gestattet, bei jeder einzelnen Taste die Wiederholungsfunktion ein- und auszuswitchen: KEY DEF 47,0 schaltet die Tastaturwiederholung der Space-Taste aus, KEY DEF 47,1 läßt sie wieder zu. Eine Tabelle der internen Tastaturcodes befindet sich im Anhang der Benutzerhandbücher und zusätzlich beim CPC-6128 auf der Oberseite des eingebauten Floppylaufwerks. Die wichtigsten Codes sind die folgenden:

```

47 - Leertaste
66 - Escape-Taste
68 - Tabulator-Taste
70 - Caps-Lock (sogar diese kann undefiniert werden!)
16 - CLR-Taste
79 - DEL-Taste
18 - Große Enter-Taste (beim CPC-6128 "Return")
0 - Cursor-Up
2 - Cursor-Down
8 - Cursor-Left
1 - Cursor-Right

```

Neben der Wiederholungsfunktion gestattet KEY DEF auch die Zuweisung eines ASCII-Codes an eine beliebige Taste. Beispielsweise zwingt KEY DEF 18,1,42 den Computer, ein Sternchen auszugeben, wenn Sie die große Enter-Taste drücken.

Eine zusätzliche Erweiterung des KEY DEF-Befehls definiert auch den Code für die Shift- und die Control-Ebene. Ein KEY DEF 66,1,48,49,50 legt auf die Escape-Taste das Zeichen "0", auf Shift-Escape "1" und auf Control-Escape "2".

Dieser KEY DEF-Befehl kann so zerlegt werden:

KEY DEF 66,	1,	48,	49,	50
-----				
Befehl	Tastencode	Wiederholung	Esc-Ebene	Shift-Esc Ctrl-Esc

Die beiden Shift-Tasten (Code 21) und die Control-Taste (Code 9) können allerdings nicht manipuliert werden. Sie erzeugen ja auch normalerweise keinen Tastencode, sondern verändern nur den ASCII-Wert einer anderen gleichzeitig gedrückten Taste.

Neben der KEY-DEF-Definition gibt es noch den KEY-Befehl, der einen String auf eine der Funktionstasten legt: Nach KEY 139,"LIST" löst der Druck auf die kleine Enter-Taste im Zehnerblock die Ausgabe des Basic-Befehls "LIST" aus. Der Zeichencode im KEY-Befehl entspricht (ausnahmsweise einmal) dem ASCII-Code. Die Funktionstasten haben Codes von 128 bis 139, Control-Enter erzeugt den Code 140. Dort liegt der vordefinierte String "RUN", der geschützte Programme vom Cassettenrecorder lädt und startet.

Laut Handbuch sind aber die Funktionstasten 128 bis 159 verfügbar. Wo liegen denn die übrigen Codes? Nun, man könnte meinen, durch Kombination der Zehnerblock-Tasten mit SHIFT oder CONTROL ließen sie sich erreichen. Doch weit gefehlt! Sie sind noch gar nicht definiert. Durch Kombination von KEY und KEY DEF werden sie auf eine der anderen Tasten gelegt.

KEY 159, "TEXT" und KEY DEF 70,1,159 weist zum Beispiel den Computer an, beim Drücken der CAPS-LOCK-Taste die Meldung "TEXT" auszugeben.

Doch wer noch mehr von den Standard-Basic-Befehlen in dieser Hinsicht erwartet, muß leider enttäuscht werden. Mehr ist ohne Hilfsprogramme oder den Aufruf von Maschinenroutinen nicht drin. Besser wird die Lage durch das Programm KeyboardKit. Dieses erweitert das Basic des Schneider-CPC um vier neue Befehle:

- !KEYDUMP gibt die Belegung der Funktionstasten aus. Damit können Sie erfahren, welche Strings nun eigentlich auf den Tasten liegen. Die Bildschirmmeldungen sehen etwa so aus:

```
KEY &80,"0"
KEY &81,"1"
KEY &82,"2"
KEY &83,"3"
.
```

Nach 25 ausgegebenen Zeilen hält das Programm an und wartet auf einen Tastendruck von Ihnen.

Das Format der Bildschirmausgabe hat den Vorteil, daß Sie mit dem Copy-Cursor Zeichendefinitionen vom Screen übernehmen und erweitern können.

Schwierigkeiten könnte es geben, wenn in einem Funktionstasten-String ein Anführungszeichen enthalten ist. Doch auch da ist bei KEYDUMP vorgesorgt. Dieses Zeichen wird mit der Basic-Funktion CHR\$ angezeigt:

```
.
.
KEY &8C,"RUN"+CHR$(34)+"PROG.BAS"
.
```

- Die beiden nächsten Befehle heißen !KEYSAVE und !KEYLOAD. Wie aus den Namen unschwer zu erkennen ist, speichern beziehungsweise laden sie eine Tastaturbelegung. So kann etwa eine deutsche Tastaturbelegung dauerhaft gesichert werden. Als Speichermedium können der Cassettenrecorder und die Diskettenstation Verwendung finden. Die Syntax sieht so aus:

```
A$="FILENAME":!KEYSAVE,@A$
A$="FILENAME":!KEYLOAD,@A$
```

Der Umweg über den Variablen-Pointer ist leider unvermeidlich und liegt in den Unzulänglichkeiten des CPC-Betriebssystems begründet.

- Der letzte Befehl, KEYBUFFER, weist den Definitionen der Funktionstasten einen neuen Speicherbereich zu. Normalerweise wird ein Puffer ab der Adresse 46150 (hexadezimal B446) reserviert. Dieser besitzt leider nur eine Länge von ca. 100 Bytes und ist damit für viele Zwecke einfach zu klein. Da das Betriebssystem aber nicht auf eine feste Adresse für den Pufferspeicher angewiesen ist, kann dieser mit KEYBUFFER verschoben werden. Sie müssen nur einen geeigneten Speicherbereich finden und diesen mit MEMORY schützen. Im KEYBUFFER-Aufruf geben Sie dann die Adresse und Länge des freien Speichers an:

:KEYBUFFER,Adresse,Länge

Zum Beispiel: MEMORY 29999:KEYBUFFER,30000,8000

Ihr Zeichenpuffer liegt dann im Speicher zwischen 30000 und 38000. Der angebotene Speicher muß mindestens 44 Bytes lang sein, damit er die Standard-Zeichen aufnehmen kann. Diese sind die Ziffern, der Dezimalpunkt, die Enter-Taste und RUN". Wenn der Speicherbereich zu klein geraten ist, bleibt der alte Speicher unverändert in Benutzung.

Nach jedem KEYBUFFER-Befehl werden alle Funktionsstrings gelöscht. Um diese über den Aufruf hinwegzureretten, bietet es sich an, auf KEYSAVE und KEYLOAD zurückzugreifen. Aufgrund des Zeitverhaltens des Cassettenrecorders ist aber die Verwendung einer Diskettenstation dringend anzuraten:

```
A$="KDAT":KEYSAVE,@A$
MEMORY 29999
:KEYBUFFER,30000,8000
:KEYLOAD,@A$
```

Das Programm ist recht gut gegen Bedienungsfehler abgesichert. Neben den normalen Basic-Fehlermeldungen gibt es folgende zusätzlichen Texte:

```
KEYSAVE: requires one argument
KEYLOAD: requires one argument
KEYBUFFER: requires two arguments
KEYDUMP: requires no argument(s)
```

Diese Meldungen erscheinen immer, wenn Sie versuchen, eine unpassende Zahl von Parametern zu übergeben. Da die Meldungen aber keine Basic-Fehlermeldungen sind und damit auch nicht ein laufendes Programm abbrechen würden, gibt KeyboardKit zusätzlich noch "Improper Argument" aus.

Bevor Sie die neuen Befehle nutzen können, bleibt Ihnen evtl. nichts anderes übrig, als das Programm abzutippen. Hier der zugehörige Quellcode.

## 23.1 Assemblerlisting: KEYBOARDKIT

```

; *****
; *
; * KeyboardKit - RSX-Erweiterung zur Verwaltung der Funktionstasten *
; *
; *****
;
; KEYDUMP      Gibt Belegung der Funktionstasten aus
; KEYSAVE      Speichert Tastaturbelegung auf Disc oder Cassette
; KEYLOAD      Lädt Tastaturbelegung von der Disc oder Cassette
; KEYBUFFER    Weist den Funktionstasten einen neuen Speicher zu
;
;
; ORG    &A000      ; Mit dem Basic-Loader verschiebbar!
; JP     RSX_INIT   ; Sprung zur RSX-Initialisierung
;
; ***** Deklarationen und Definitionen *****
;
BUFF_START EQU    &B4E1      ; Start Expbuffer, CPC-664/6128 = &B62B
BUFF_END   EQU    &B4E3      ; Ende Expbuffer, CPC-664/6128 = &B62D

KL_U_ROM_ON EQU    &B900      ; KL UROM ENABLE - Basic-ROM einblenden
KM_EXP_BUFFER EQU    &BB15      ; KM EXP BUFFER - Expbuffer einrichten
KM_WAIT_KEY EQU    &BB18      ; KM WAIT KEY - wartet auf Tastendruck
TXT_OUTPUT EQU    &BB5A      ; TXT TXT_OUTPUT - Bildschirmausgabe
TXT_WR_CHAR EQU    &BB5D      ; TXT WR CHAR - druckt auch ASCII 0-31
CAS_IN_OPEN EQU    &BC77      ; CAS IN OPEN - Datei zum Lesen öffnen
CAS_IN_CLOSE EQU    &BC7A      ; CAS IN CLOSE - Eingabedatei schließen
CAS_IN_DIRECT EQU    &BC83      ; CAS IN DIRECT - Eingabedatei lesen
CAS_OUT_OPEN EQU    &BC8C      ; CAS OUT OPEN - zum Schreiben öffnen
CAS_OUT_CLOSE EQU    &BC8F      ; CAS OUT CLOSE - Outputfile schließen
CAS_OUT_DIRECT EQU    &BC98      ; CAS OUT DIRECT - Outputfile schreiben
KL_LOG_EXT EQU    &BCD1      ; KL LOG EXT - RSX-Routinen einbinden

GET_INPBUFFER EQU    &F632      ; Inputbuffer reservieren
REL_INPBUFFER EQU    &F66D      ; Inputbuffer freigeben
GET_OUTBUFFER EQU    &F637      ; Outputbuffer reservieren
REL_OUTBUFFER EQU    &F671      ; Outputbuffer freigeben
BASIC_ERROR EQU    &CA94      ; Basic-Fehlermeldung ausgeben

;
; ***** Texte und Fehlermeldungen *****
;

ERR_KEYSAYE DEFEB 7
DEFEB "KEYSAVE:"
DEFEB 32
DEFEB "requires one argument"
DEFEB 13,10,0

ERR_KEYLOAD DEFEB 7
DEFEB "KEYLOAD:"
DEFEB 32
DEFEB "requires one argument"
DEFEB 13,10,0

```



```

ERR_KEYBUFF      DEFB      7
                  DEFM      "KEYBUFFER:"
                  DEFB      32
                  DEFM      "requires two arguments"
                  DEFB      13,10,0

ERR_KEYDUMP       DEFB      7
                  DEFM      "KEYDUMP:"
                  DEFB      32
                  DEFM      "requires no argument(s)"
                  DEFB      13,10,0

KEYDUMP_TEXT      DEFB      13
                  DEFM      "KEY"
                  DEFB      32,30
                  DEFB      0

KEYD_TEXT2        DEFB      34
                  DEFM      "+CHR$(34)+",34
                  DEFB      0

;
; ***** Daten zur Verwaltung der RSXen *****
;

RSX_TABLE          DEFW      NAME_TABLE      ; Zeiger auf Namenstabelle
                  JP        KEYDUMP         ; Sprung nach KEYDUMP
                  JP        KEYSAVE         ; Sprung nach KEYSAVE
                  JP        KEYLOAD         ; Sprung nach KEYLOAD
                  JP        KEYBUFFER        ; Sprung nach KEYBUFFER

NAME_TABLE         DEFM      "KEYDUM"       ; RSX-Name KEYDUMP
                  DEFB      "P"+&00         ; Ende des Namens

NAME_KEYSAYE       DEFM      "KEYSAV"       ; RSX-Name KEYSAYE
                  DEFB      "E"+&00         ; Ende des Namens

NAME_KEYLOAD       DEFM      "KEYLOA"       ; RSX-Name KEYLOAD
                  DEFB      "D"+&00         ; Ende des Namens

NAME_KEYBUFFER     DEFM      "KEYBUFFE"     ; RSX-Name KEYBUFFER
                  DEFB      "R"+&00         ; Ende des Namens
                  DEFB      &00            ; Ende der Tabelle

KL_SPACE          DEFB      &04             ; Hilfsspeicher für Kernel-ROM
;
; ***** Einbindung der RSXen ins Betriebssystem *****
;

RSX_INIT           LD        BC,RSX_TABLE   ; Zeiger auf RSX-Tabelle
                  LD        HL,KL_SPACE     ; Zeiger auf Hilfsspeicher
                  CALL       KL_LOG_EXT     ; RSX-Routine einbinden
                  RET                      ; Rücksprung nach Basic

;
; ***** KEYSAYE *****

```

```

;

KEYSAVE      CP      1          ; Wird genau ein Argument Übergeben?
              JR      Z,KEYSAVE_OK ; Ja - normal weiterarbeiten

KEYSAVE_ERROR LD      HL,ERR_KEYSAVE ; Zeiger auf Fehlermeldung
              CALL    STRING_PRINT  ; Meldung ausgeben
              CALL    KL_U_ROM_ON   ; Basic-ROM einschalten
              LD      E,5          ; Fehler "Improper Argument"
              LD      A,5          ; Für den 664 und 6128 im Akku!
              JP      BASIC_ERROR   ; Fehler ausgeben

KEYSAVE_OK    CALL    KL_U_ROM_ON   ; Basic-ROM einblenden
              LD      H,(IX+1)      ; MSByte des Stringdescriptors holen
              LD      L,(IX+0)      ; Und dessen Lowbyte (Adresse)
              LD      B,(HL)        ; Stringlänge lesen
              INC     HL            ; Zeiger erhöhen
              LD      E,(HL)        ; Lowbyte der Stringadresse
              INC     HL            ; Zeiger erhöhen
              LD      D,(HL)        ; Highbyte der Stringadresse
              EX      DE,HL         ; Adresse ins HL-Register
              CALL    GET_OUTBUFFER ; Ausgabepuffer reservieren
              CALL    CAS_OUT_OPEN  ; Datei zum Schreiben öffnen

KEYSAVE_ADDR  LD      DE,(BUFF_START) ; Zeiger auf Beginn der F-Tasten
              LD      HL,(BUFF_END)   ; Zeiger auf Ende der F-Tasten
              OR      A              ; SBC vorbereiten
              SBC     HL,DE          ; HL=HL-DE, HL=Länge
              EX      DE,HL         ; Länge im DE-Register
              LD      HL,(BUFF_START) ; Zeiger auf Beginn der F-Tasten

KEYSAVE_SAVE  LD      BC,0          ; Einsprungsadresse 0
              LD      A,&2C          ; Dateiart für den Kopfsatz
              CALL    CAS_OUT_DIRECT ; Daten ausgeben
              CALL    CAS_OUT_CLOSE  ; Datei schließen
              CALL    REL_OUTBUFFER  ; Ausgabepuffer freigeben
              RET                   ; Rücksprung nach Basic

;
; ***** KEYLOAD *****
;

KEYLOAD      CP      1          ; Wird genau ein Argument Übergeben?
              JR      Z,KEYLOAD_OK  ; Ja - normal weiterarbeiten

KEYLOAD_ERROR LD      HL,ERR_KEYLOAD ; Zeiger auf Fehlermeldung
              CALL    STRING_PRINT  ; Meldung ausgeben
              CALL    KL_U_ROM_ON   ; Basic-ROM einschalten
              LD      E,5          ; Fehler "Improper Argument"
              LD      A,5          ; Für den 664 und 6128 im Akku!
              JP      BASIC_ERROR   ; Fehler ausgeben

KEYLOAD_OK    CALL    KL_U_ROM_ON   ; Basic-ROM einblenden
              LD      H,(IX+1)      ; MSByte des Stringdescriptors holen
              LD      L,(IX+0)      ; Und dessen Lowbyte (Adresse)
              LD      B,(HL)        ; Stringlänge lesen
              INC     HL            ; Zeiger erhöhen
              LD      E,(HL)        ; Lowbyte der Stringadresse

```

```

        INC    HL            ; Zeiger erhöhen
        LD     D,(HL)        ; Highbyte der Stringadresse
        EX     DE,HL         ; Adresse ins HL-Register
        CALL   GET_INPBUFFER ; Eingabepuffer reservieren
        CALL   CAS_IN_OPEN   ; Datei zum Lesen öffnen

KEYLOAD_LOAD LD     HL,(BUFF_START) ; Startadresse des Expbuffers holen
        CALL   CAS_IN_DIRECT ; Datei als ganzes einlesen
        CALL   CAS_IN_CLOSE  ; Datei schließen
        CALL   REL_INPBUFFER ; Eingabepuffer freigeben
        RET                     ; Rücksprung nach Basic

;
; ***** KEYBUFFER *****
;

KEYBUFFER   CP     2          ; Genau zwei Argumente übergeben?
        JR     Z,KEYBUFFER_OK ; Ja - normal weiterarbeiten

KEYBUFFER_ERROR LD     HL,ERR_KEYBUFF ; Zeiger auf Fehlermeldung
        CALL   STRING_PRINT ; Meldung ausgeben
        CALL   KL_U_ROM_ON   ; Basic-ROM einschalten
        LD     E,5           ; Fehler "Improper Argument"
        LD     A,5           ; Für den 664 und 6128 im Akku!
        JP     BASIC_ERROR  ; Fehler ausgeben

KEYBUFFER_OK LD     H,(IX+1) ; ins HL-Register die Länge holen
        LD     L,(IX+0)
        LD     D,(IX+3)     ; ins DE-Register die Adresse
        LD     E,(IX+2)
        CALL   KM_EXP_BUFFER ; Routine aufrufen
        RET                     ; Rücksprung nach Basic

;
; ***** KEYDUMP *****
;

KEYDUMP     CP     0          ; Keine Argumente übergeben!
        JR     Z,KEYDUMP_OK  ; Wenn richtig, normal weiterarbeiten

KEYDUMP_ERROR LD     HL,ERR_KEYDUMP ; Zeiger auf Fehlermeldung
        CALL   STRING_PRINT ; Meldung ausgeben
        CALL   KL_U_ROM_ON   ; Basic-ROM einschalten
        LD     E,5           ; Fehler "Improper Argument"
        LD     A,5           ; Für den 664 und 6128 im Akku!
        JP     BASIC_ERROR  ; Fehler ausgeben

KEYDUMP_OK  LD     HL,(BUFF_START) ; Zeiger auf Speicher der F-Tasten
        LD     C,128         ; Code der ersten Funktionstaste

KEYDUMP_CLS LD     A,12      ; Steuercode für CLEAR SCREEN
        CALL   TXT_OUTPUT    ; Und damit den Bildschirm löschen

SHOW_TEXT  PUSH    HL        ; HL sichern
        LD     HL,KEYDUMP_TEXT ; Zeiger auf "KEY &"
        CALL   STRING_PRINT  ; String ausgeben
        POP    HL           ; HL wiederherstellen

```

```

SHOW_HEX      CALL  HEX           ; ASCII-Code der F-Taste ausgeben
               LD    A,44         ; ASCII-Code des Kommas
               CALL  TXT_OUTPUT    ; Ebenfalls ausgeben
               LD    A,34         ; ASCII-Code des Anführungszeichens
               CALL  TXT_OUTPUT    ; Wiederum ausgeben

LOOP1         LD    A,(HL)        ; Längenbyte der Zeichenkette lesen
               LD    B,A          ; Als Schleifenzähler ins B-Register
               INC   HL           ; Zeiger auf Zeichenadresse erhöhen
               CP    0            ; War die F-Taste ein Leerstring?
               JR    Z,CLOSE      ; Ja - Schlußzeichen, nächste Taste

LOOP2         LD    A,(HL)        ; Zeichen des Funktionsstrings lesen
               CP    34           ; Ist es ein Anführungszeichen?
               JR    NZ,NORM_CHAR ; Nein - ein normales Zeichen

QUOTATION_MARK PUSH  HL          ; HL-Register sichern
               LD    HL,KEYD_TEXT2 ; Zeiger auf "+CHR$(34)+"
               CALL  STRING_PRINT  ; Ausgeben
               POP   HL          ; HL zurückholen
               JR    LOOP_CONTIN  ; Weiter in der Schleife

NORM_CHAR     PUSH  AF           ; Akku & Flags sichern
               PUSH  BC           ; BC-Register sichern
               PUSH  DE           ; DE-Register sichern
               PUSH  HL          ; HL-Register sichern
               CALL  TXT_WR_CHAR  ; Zeichen ausgeben
               POP   HL          ; HL-Register wiederherstellen
               POP   DE           ; DE-Register wiederherstellen
               POP   BC           ; BC-Register wiederherstellen
               POP   AF          ; Akku und Flags wiederherstellen

LOOP_CONTIN   INC    HL          ; Zeiger aufs nächste Zeichen
               DJNZ  LOOP2        ; Weiter, bis der String gelesen ist

CLOSE         LD    A,34         ; Schlußzeichen laden
               CALL  TXT_OUTPUT    ; Und ausgeben
               LD    A,10        ; Line-Feed laden
               CALL  TXT_OUTPUT    ; Und ausgeben

NEXT          INC    C           ; Zähler der Funktionstasten erhöhen
               LD    A,C          ; Vorbereitung des CP-Befehls
               CP    153         ; Nach 25 Tasten ist der Screen voll
               CALL  Z,KM_WAIT_KEY ; Wenn ja - auf Tastendruck warten
               CP    160         ; Höchster F-Tastencode ist 159
               JR    NZ,SHOW_TEXT ; Weitermachen, bis erreicht
               RET               ; Sonst Rücksprung nach Basic

;
; ***** STRING_PRINT druckt Text von (HL) bis (HL)=0 *****
;

STRING_PRINT  PUSH  AF           ; Akku und F-Register retten

PRINT_LOOP   LD    A,(HL)        ; Zeichen lesen
               CP    0            ; Nullbyte?
               JR    Z,PRINT_RETURN ; Ja, Ende

```

```

SHOW_CHARACTER  CALL  TXT_OUTPUT      ; Zeichen ausgeben
                  INC    HL            ; Zeiger erhöhen
                  JR     PRINT_LOOP    ; Weiter in der Schleife

PRINT_RETURN    POP     AF            ; Akku und Flags wiederherstellen
                  RET                  ; Rücksprung

;
; ***** Hexadezimale Zeichenausgabe *****
;

HEX              LD     A,C            ; Hexziffer in den Akku laden
                  CALL  SHIFT          ; Shiften und linkes Nibble ausgeben
                  LD     A,C            ; Akku wiederherstellen
                  CALL  HXPRNT         ; Rechtes Nibble ausgeben
                  RET                  ; Rücksprung zum aufrufenden Programm

SHIFT            RRA                  ; Mit diesen vier RRA-Befehlen wird
                  RRA                  ; das rechte Nibble aus dem Akku
                  RRA                  ; herausgeschifft und das linke Nibble
                  RRA                  ; zum rechten Nibble gemacht

HXPRNT           AND     &0F           ; Löscht das linke Nibble
                  ADD     A,48          ; Wandlung von Binär nach ASCII
                  CP      58            ; Größer als 9?
                  JR     C,HEXOUT       ; Nein - Zeichen ausdrucken
                  ADD     A,7           ; Ja - ASCII-Zahlen-Buchstaben-Offset!

HEXOUT           CALL  TXT_OUTPUT      ; Hexziffer ausgeben
                  RET                  ; Und Rücksprung
                  END

; -----

```

Da die Eingabe eines doch recht langen Programms nicht jedermanns Sache ist, finden Sie hier gleich noch einen Basic-Loader, der ein ganzes Stück kürzer ist, aber die gleiche Aufgabe erfüllt.

### 23.2 Listing: KBKIT.BAS

```

100 ' *****
110 ' *
120 ' *      KeyboardKit      *
130 ' *
140 ' *****
150 '
160 DEF FNlsb(a)=255 AND UNT(a)
170 DEF FNmsb(a)=255 AND INT(a/256)
180 DIM checksum(33)
190 SYMBOL AFTER 256:MEMORY HIMEM-537
200 start=HIMEM+1:SYMBOL AFTER 240
210 ' Initialisierung des Basic-Loaders *****
220 SYMBOL 253,&66,&0,&78,&C,&7C,&CC,&76,&0
230 SYMBOL 254,&66,&0,&3C,&66,&66,&66,&3C,&0

```

```

240 SYMBOL 255,&66,&0,&66,&66,&66,&3E,&0
250 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
260 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
270 ' Titelbild und Benutzerinformationen *****
280 LOCATE 1,1:PRINT STRING$(40,210);
290 LOCATE 1,2
300 PRINT CHR$(24)+SPACE$(14)+"KeyboardKit"+SPACE$(15)+CHR$(24)
310 PRINT:PRINT STRING$(40,210):PRINT
320 PEN 1:PRINT " KeyboardKit ist eine RSX-Erweiterung,"
330 PRINT:PRINT " die die Programmierung der Funktions-"
340 PRINT:PRINT " tasten erheblich vereinfacht:"
350 PEN 3:PRINT:PRINT STRING$(40,210):PEN 2
360 PRINT:PRINT TAB(5);"!KEYSAVE,@FILE$"
370 PRINT:PRINT TAB(5);"!KEYLOAD,@FILE$"
380 PRINT:PRINT TAB(5);"!KEYBUFFER,Start,L"+CHR$(253)+"nge"
390 PRINT:PRINT TAB(5);"!KEYDUMP"
400 PEN 1:RESTORE 820
410 FOR i=&A000 TO &A215:READ a:sum=sum+a:NEXT i
420 IF sum=59213 THEN 460
430 PRINT CHR$(7);"* Fehler in den DATA-Zeilen!":PRINT
440 LIST 810:END
450 ' Zeile für Zeile auf DATA-Fehler prüfen *****
460 RESTORE
470 FOR i=1 TO 33:READ checksum(i):NEXT i
480 RESTORE 820
490 FOR i=1 TO 33
500   sum=0
510   FOR j=i TO 16
520     READ a
530     sum=sum+a
540   NEXT j
550   IF sum<>checksum(i) THEN PRINT "FEHLER!":END
560 NEXT i
570 RESTORE
580 ' Das Maschinenprogramm wird eingePOKEt und verschoben *****
590 RESTORE 820:FOR i=start TO start+&215:READ a:POKE i,a:NEXT i
600 FOR i=1 TO 23:READ a:a=a-40960+start
610   wert=PEEK(a)+PEEK(a+1)*256-40960+start
620   POKE a,FNlsb(wert):POKE a+1,FNmsb(wert)
630 NEXT i
640 CALL start ' RSXen initialisieren
650 POKE &160,&CD:POKE &161,&0:POKE &162,&B9:POKE &163,&3A:POKE &164,&2
660 POKE &165,&C0:POKE &166,&32:POKE &167,&60:POKE &168,&1:POKE &169,&C9
670 CALL &160:cpcversion=PEEK(&160)
680 IF cpcversion=0 THEN RESTORE 1240
690 IF cpcversion=1 THEN RESTORE 1290
700 IF cpcversion=2 THEN RESTORE 1340
710 FOR i=1 TO 13:READ address,routine.address
720 address=address-40960+start
730 POKE address,FNlsb(routine.address)
740 POKE address+1,FNmsb(routine.address)
750 NEXT i
760 MODE 1:END
770 ' Prüfsummen für alle Zeilen *****
780 DATA 1529,1579,981,1583,1059,1482,1245,1422,1229,793,2105,1587,1452
790 DATA 1784,1347,1824,2725,2001,2301,1767,1945,2325,2091,1929,1451
800 DATA 2096,2492,1550,2767,2053,2405,2091,1553

```

```

810 ' Daten des Maschinenprogramms *****
820 DATA &C3,&D5,&A0,&07,&4B,&45,&59,&53,&41,&56,&45,&3A,&20,&72,&65,&71
830 DATA &75,&69,&72,&65,&73,&20,&6F,&6E,&65,&20,&61,&72,&67,&75,&6D,&65
840 DATA &6E,&74,&0D,&0A,&00,&07,&4B,&45,&59,&4C,&4F,&41,&44,&3A,&20,&72
850 DATA &65,&71,&75,&69,&72,&65,&73,&20,&6F,&6E,&65,&20,&61,&72,&67,&75
860 DATA &6D,&65,&6E,&74,&0D,&0A,&00,&07,&4B,&45,&59,&42,&55,&46,&46,&45
870 DATA &52,&3A,&20,&72,&65,&71,&75,&69,&72,&65,&73,&20,&74,&77,&6F,&20
880 DATA &61,&72,&67,&75,&6D,&65,&6E,&74,&73,&0D,&0A,&00,&07,&4B,&45,&59
890 DATA &44,&55,&4D,&50,&3A,&20,&72,&65,&71,&75,&69,&72,&65,&73,&20,&6E
900 DATA &6F,&20,&61,&72,&67,&75,&6D,&65,&6E,&74,&2B,&73,&29,&0D,&0A,&00
910 DATA &0D,&4B,&45,&59,&20,&2B,&00,&22,&2B,&43,&48,&52,&24,&28,&33,&34
920 DATA &29,&2B,&22,&00,&B2,&A0,&C3,&7F,&A1,&C3,&DF,&A0,&C3,&25,&A1,&C3
930 DATA &5B,&A1,&4B,&45,&59,&44,&55,&4D,&D0,&4B,&45,&59,&53,&41,&56,&C5
940 DATA &4B,&45,&59,&4C,&4F,&41,&C4,&4B,&45,&59,&42,&55,&46,&46,&45,&D2
950 DATA &00,&00,&00,&00,&00,&01,&A4,&A0,&21,&D1,&A0,&CD,&D1,&BC,&C9,&FE
960 DATA &01,&28,&10,&21,&03,&A0,&CD,&EC,&A1,&CD,&00,&B9,&1E,&05,&3E,&05
970 DATA &C3,&94,&CA,&CD,&00,&B9,&DD,&66,&01,&DD,&6E,&00,&46,&23,&5E,&23
980 DATA &56,&EB,&CD,&37,&F6,&CD,&8C,&BC,&ED,&5B,&E1,&B4,&2A,&E3,&B4,&B7
990 DATA &ED,&52,&EB,&2A,&E1,&B4,&01,&00,&00,&3E,&2C,&CD,&9B,&BC,&CD,&8F
1000 DATA &BC,&CD,&71,&F6,&C9,&FE,&01,&28,&10,&21,&25,&A0,&CD,&EC,&A1,&CD
1010 DATA &00,&B9,&1E,&05,&3E,&05,&C3,&94,&CA,&CD,&00,&B9,&DD,&66,&01,&DD
1020 DATA &6E,&00,&46,&23,&5E,&23,&56,&EB,&CD,&32,&F6,&CD,&77,&BC,&2A,&E1
1030 DATA &B4,&CD,&83,&BC,&CD,&7A,&BC,&CD,&6D,&F6,&C9,&FE,&02,&28,&10,&21
1040 DATA &47,&A0,&CD,&EC,&A1,&CD,&00,&B9,&1E,&05,&3E,&05,&C3,&94,&CA,&DD
1050 DATA &66,&01,&DD,&6E,&00,&DD,&56,&03,&DD,&5E,&02,&CD,&15,&BB,&C9,&FE
1060 DATA &00,&28,&10,&21,&6C,&A0,&CD,&EC,&A1,&CD,&00,&B9,&1E,&05,&3E,&05
1070 DATA &C3,&94,&CA,&2A,&E1,&B4,&0E,&00,&3E,&0C,&CD,&5A,&BB,&E5,&21,&90
1080 DATA &A0,&CD,&EC,&A1,&E1,&CD,&FA,&A1,&3E,&2C,&CD,&5A,&BB,&3E,&22,&CD
1090 DATA &5A,&BB,&7E,&47,&23,&FE,&00,&28,&1D,&7E,&FE,&22,&20,&0A,&E5,&21
1100 DATA &97,&A0,&CD,&EC,&A1,&E1,&18,&0B,&F5,&C5,&D5,&E5,&CD,&BB,&E1
1110 DATA &D1,&C1,&F1,&23,&10,&E3,&3E,&22,&CD,&5A,&BB,&3E,&0A,&CD,&5A,&BB
1120 DATA &0C,&79,&FE,&99,&CC,&18,&BB,&FE,&A0,&20,&B2,&C9,&F5,&7E,&FE,&00
1130 DATA &28,&06,&CD,&5A,&BB,&23,&18,&F5,&F1,&C9,&79,&CD,&03,&A2,&79,&CD
1140 DATA &07,&A2,&C9,&1F,&1F,&1F,&1F,&E6,&0F,&C6,&30,&FE,&3A,&38,&02,&C6
1150 DATA &07,&CD,&5A,&BB,&C9,&00
1160 ' DATAs zum Relokalisieren des Maschinencodes *****
1170 DATA &A001,&A0A4,&A0A7,&A0AA,&A0AD,&A0B0,&A0D6,&A0D9,&A0E4,&A0E7,&A12A
1180 DATA &A12D,&A160,&A163,&A184,&A1B7,&A19F,&A1A2,&A1A6,&A1C0,&A1C3,&A1FC
1190 DATA &A200
1200 '
1210 ' DATAs zum Anpassen an den CPC-464, CPC-664 und CPC-6128 *****
1220 '
1230 ' - CPC-464-Tabelle:
1240 DATA &A0F1,&CA94,&A137,&CA94,&A16D,&CA94,&A191,&CA94
1250 DATA &A103,&F637,&A122,&F671,&A149,&F632,&A158,&F66D
1260 DATA &A10A,&B4E1,&A114,&B4E1,&A14F,&B4E1,&A194,&B4E1
1270 DATA &A10D,&B4E3
1280 ' - CPC-664-Tabelle:
1290 DATA &A0F1,&CB55,&A137,&CB55,&A16D,&CB55,&A191,&CB55
1300 DATA &A103,&F725,&A122,&F75D,&A149,&F720,&A158,&F759
1310 DATA &A10A,&B62B,&A114,&B62B,&A14F,&B62B,&A194,&B62B
1320 DATA &A10D,&B62D
1330 ' - CPC-6128-Tabelle:
1340 DATA &A0F1,&CB4D,&A137,&CB4D,&A16D,&CB4D,&A191,&CB4D
1350 DATA &A103,&F725,&A122,&F75D,&A149,&F720,&A158,&F759
1360 DATA &A10A,&B62B,&A114,&B62B,&A14F,&B62B,&A194,&B62B
1370 DATA &A10D,&B62D

```

1380 END ' -----

Das Programm erkennt fast jeden Eingabefehler in den DATA-Zeilen, denn es besitzt zwei Prüfsummen-Tests.

Wenn der Computer "FEHLER!" meldet, können Sie durch PRINT 1 feststellen, welcher DATA-Wert falsch ist. Der Fehler liegt dann irgendwo in derjenigen Zeile, die Sie durch PRINT 1 gefunden haben.

Der Basic-Loader übernimmt auch gleich noch die Arbeit, das Programm im Speicher so zu verschieben, daß es unmittelbar unter HIMEM zu liegen kommt. Außerdem paßt er das Maschinenprogramm an den verwendeten Computer (CPC-464, CPC-664 und CPC-6128) an.

Zum Aufbau des Maschinenprogramms: Um die RSX-Befehle zu initialisieren, muß die Routine RSX\_INIT aufgerufen werden. Dies macht der Basic-Loader automatisch.

Die einzelnen Programmteile sind recht deutlich voneinander abgetrennt. Beginnen wir mit KEYSAVE. Dieses prüft, ob wirklich ein Parameter übergeben wird (CP 1). Ist alles glatt gegangen, springt das Programm zur Marke KEYSAVE\_OK. Andernfalls geht es bei KEYSAVE\_ERROR weiter: Dieser Programmteil gibt eine Fehlermeldung auf dem Bildschirm aus ("KEYSAVE: requires one argument") und veranlaßt den Basic-Interpreter, die Meldung "Improper Argument" von sich zu geben. Danach landen Sie wieder im Basic und können Ihren Fehler analysieren.

Gesetzt den Fall, daß der Aufruf korrekt erfolgte, holt sich das Programm einen Zeiger auf den String, der den Dateinamen enthält. Mit CALL GET\_OUTBUFFER wird ein 2 Kilobytes großer Ausgabepuffer angelegt. Im DE-Register steht nach Aufruf der Routine ein Zeiger auf den Speicherbereich. CALL CAS\_OUT\_OPEN öffnet die Ausgabedatei auf dem Cassettenrecorder oder der Diskette.

KEYSAVE\_ADDRS holt sich mittels einiger Berechnungen einen Zeiger auf den Beginn des Funktionstasten-Puffers ins HL-Register und die Länge des Puffers ins DE-Doppelregister. KEYSAVE\_SAVE nutzt CAS\_OUT\_DIRECT, um den Speicherauszug auf dem Datenträger zu sichern. CALL CAS\_OUT\_CLOSE schließt die Datei wieder und CALL REL\_OUTBUFFER ("Release Output Buffer") gibt den belegten Speicherplatz wieder frei.

Die Fehlerüberprüfung bei KEYLOAD und bei den anderen Routinen ist ganz ähnlich, so daß wir darauf nicht mehr eingehen werden.

KEYLOAD\_OK selbst holt sich einen Zeiger auf den Dateinamen, reserviert Speicherplatz und öffnet die einzulesende Datei. Über CAS\_IN\_DIRECT wird die Datei in den Speicher übertragen, und CAS\_OUT\_CLOSE schließt sie wieder. CALL REL\_INPBUFFER ("Release Input Buffer") gibt den als Eingabepuffer benötigten Speicherplatz wieder frei.

KEYBUFFER ist ganz einfach zu programmieren, denn die Routine besteht im wesentlichen aus der Datenübernahme (im HL-Register



steht die Länge des zugewiesenen Speicherbereichs, im DE-Register dessen Startadresse) und dem Aufruf der Systemroutine KM EXP BUFFER.

Umfangreicher ist da der Programmcode von KEYDUMP. In KEYDUMP\_OK holt sich das Programm einen Zeiger auf die Startadresse des Puffers und lädt das C-Register des Prozessors mit dem ASCII-Code der ersten Funktionstaste.

KEYDUMP\_CLS löscht durch Ausgabe des Controlcodes 12 den Bildschirm. Der Computer läuft dann in eine Programmschleife.

SHOW\_TEXT gibt auf dem Bildschirm den Text "KEY &" aus, der ja am Anfang jeder KEY-Definition steht. Im Programmteil SHOW\_HEX gibt KeyboardKit den ASCII-Wert der bearbeiteten Taste hexadezimal aus. Dazu dient der Unterprogramm-Aufruf CALL HEX. Diese Routine erwartet im C-Register das zu druckende Zeichen. In LOOP1 liest der Computer das Längenbyte des Tastenstrings. Dieses wird ins B-Register übernommen, um eine einfache Schleife mit DJNZ konstruieren zu können.

Um das zu verstehen, müssen Sie sich vergegenwärtigen, wie die Funktionszeichen-Tabelle im Speicher aufgebaut ist. Sie beginnt immer mit einem Längenbyte, das bei nicht belegten Tasten eben den Wert Null hat. Darauf folgen im ASCII-Code die einzelnen Zeichen:

Start+0:	Längenbyte der Taste 128	-	1
Start+1:	ASCII-Zeichen	-	"0"
Start+2:	Längenbyte der Taste 129	-	1
Start+3:	ASCII-Zeichen	-	"1"
Start+4:	Längenbyte der Taste 130	-	1
Start+5:	ASCII-Zeichen	-	"2"
.			
.			
Start+20:	Längenbyte der Taste 138	-	1
Start+21:	ASCII-Zeichen	-	","
Start+22:	Längenbyte der Taste 139	-	1
Start+23:	ASCII-Zeichen	-	13
Start+24:	Längenbyte der Taste 140	-	5
Start+25:	ASCII-Zeichen 1	-	R
Start+26:	ASCII-Zeichen 2	-	U
Start+27:	ASCII-Zeichen 3	-	N
Start+28:	ASCII-Zeichen 4	-	"
Start+29:	ASCII-Zeichen 5	-	13 (Wagenrücklauf)
.			
.			
.			

Wenn im B-Register eine Null steht und damit die Funktionstaste nicht belegt ist, springt der Computer aus der Schleife heraus

und beginnt mit der Abarbeitung der nächsten Funktionstaste (CP Ø und JR Z,CLOSE).

Sind allerdings Zeichen auszugeben, liest er diese mit LD A,(HL) und prüft, ob er ein Anführungszeichen findet. Dieses muß gesondert behandelt werden. Es wird als "+CHR\$(34)+" gedruckt. Die übrigen Zeichen hingegen können normal mit TXT WR CHAR ausgegeben werden. Die PUSH- und POP-Befehle sind notwendig, weil TXT WR CHAR alle Registerinhalte von AF, BC, DE und HL zerstören darf.

In LOOP\_CONTIN wird der Zeiger auf das nächste Codebyte gerichtet, und DJNZ bewirkt einen Rücksprung in die Ausgabeschleife.

Wenn alle Zeichen eines Funktionsstrings ausgegeben sind, führt der Computer das Programm ab CLOSE aus. Dort gibt er ein Schlußzeichen (Anführungszeichen) aus und erhöht den intern verwendeten ASCII-Wert der Taste. Wenn 25 Tastenbelegungen auf dem Bildschirm stehen, wartet der Computer, bis Sie eine Taste drücken (CP 153 und CALL Z,KM\_WAIT\_KEY). Sobald der ASCII-Code 160 erreicht ist, hat der Computer die Ausgabe zu beenden und kehrt nach Basic zurück.

Ein weiteres Programm, das auch in den Bereich der Tastaturprogrammierung gehört, ist KEYPROG.BAS. Es erzeugt von der gerade verwendeten Tastaturbelegung ein Basic-Programm, das aus lauter KEY-Befehlen besteht. Bei der Standard-Tastaturbelegung sieht das erzeugte Programm so aus:

```
100 KEY &80,"Ø"
110 KEY &81,"1"
120 KEY &82,"2"
130 KEY &83,"3"
140 KEY &84,"4"
150 KEY &85,"5"
160 KEY &86,"6"
170 KEY &87,"7"
180 KEY &88,"8"
190 KEY &89,"9"
200 KEY &8A,"."
210 KEY &8B,""+CHR$(13)+" "
220 KEY &8C,"RUN"+CHR$(34)+" "+CHR$(13)+" "
230 KEY &8D,""
240 KEY &8E,""
.
.
.
```

Das Programm "umschreibt" alle Anführungszeichen, Control-Codes zwischen Ø und 31 und Grafikzeichen mit einem ASCII-Code über 126 durch CHR\$. Damit kann das erzeugte Programm auch auf dem Drucker gelistet werden.

Mit nachfolgendem Programm wird die Datei erstellt.

## 23.3 Listing: KEYPROG.BAS

```

100 ' *****
110 ' *
120 ' *          KEYPROG.BAS          *
130 ' *
140 ' *****
150 '
160 ' Titelbild und Informationen *****
170 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
180 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
190 LOCATE 1,1:PRINT STRING$(40,210);
200 LOCATE 1,2
210 PRINT CHR$(24)+SPACE$(14)+"KEYPROG.BAS"+SPACE$(15)+CHR$(24)
220 PRINT:PRINT STRING$(40,210):PRINT
230 PEN 1:PRINT " Dieses Programm erzeugt auf der"
240 PRINT:PRINT " Diskette oder Cassette ein Basic~"
250 PRINT:PRINT " Programm, das aus KEY-Befehlen be~"
260 PRINT:PRINT " steht, die die derzeitigen Tasten~"
270 PRINT:PRINT " definitionen enthalten."
280 PEN 3:PRINT:PRINT STRING$(40,210):PRINT
290 PEN 2:PRINT:PRINT " Name der Ausgabedatei: ";
300 PEN 1:LINE INPUT a$
310 OPENOUT a$
320 ' Computertyp feststellen *****
330 POKE &160,&CD:POKE &161,&0:POKE &162,&B9 ' CALL &B900
340 POKE &163,&3A:POKE &164,&2:POKE &165,&C0 ' LD A,(&C002)
350 POKE &166,&32:POKE &167,&60:POKE &168,&1 ' LD (&160),A
360 POKE &169,&C9 ' RET
370 CALL &160
380 cpcversion=PEEK(&160)
390 IF cpcversion=0 THEN start.expbuffers=&B4E1
400 IF cpcversion=1 OR cpcversion=2 THEN start.expbuffers=&B62B
410 IF start.expbuffers=0 THEN PRINT " Unbekannter Computer!":END
420 start=PEEK(start.expbuffers)+PEEK(start.expbuffers+1)*256
430 ' Hauptprogramm *****
440 taste=128
450 zeile=100
460 laenge=PEEK(start)
470 PRINT #9,MID$(STR$(zeile),2);"KEY ";HEX$(taste);";";CHR$(34);
480 zeile=zeile+10
490 FOR j=1 TO laenge
500 asciizeichen=PEEK(j+start)
510 ' Anfuhrungszeichen? -----
520 IF asciizeichen<>34 THEN 560
530 PRINT #9,CHR$(34)+"+CHR$(34)+";CHR$(34);
540 GOTO 650
550 ' Controlcode? -----
560 IF asciizeichen>31 THEN 600
570 PRINT #9,CHR$(34);"+CHR$(";MID$(STR$(asciizeichen),2);")+"+CHR$(34);
580 GOTO 650
590 ' Grafikzeichen? -----
600 IF asciizeichen<127 THEN 640
610 PRINT #9,CHR$(34);"+CHR$(";MID$(STR$(asciizeichen),2);")+"+CHR$(34);
620 GOTO 650
630 ' Normales Zeichen! -----
640 PRINT #9,CHR$(PEEK(j+start));

```

```

650 NEXT j
660 PRINT #9,CHR$(34)
670 taste=taste+1
680 start=start+laenge+1
690 IF taste<160 THEN 460
700 CLOSEOUT
710 END ' *****

```

KEYPROG.BAS ist - mehr oder weniger - eine Umsetzung von KEYDUMP in ein Basic-Programm. Der Unterschied liegt darin, daß alle Ausgaben nach #9 gehen, also auf den Cassettenrecorder oder die Diskettenlaufwerke.

Vom Verständnis her sollte das Programm keine übergroßen Schwierigkeiten bereiten: In den Zeilen 320 bis 420 stellt es fest, auf welchem Schneider-Computer es läuft. Über die Adresse &B4E1 beziehungsweise &B62B holt sich das Programm einen Zeiger auf den Speicherbereich, in dem die Zeichenketten abgelegt sind.

Diese Adresse wird in der Variablen START vermerkt. Ab da läuft eigentlich alles wie beim KEYDUMP-Befehl. KEYPROG liest hier das Längenbyte einer Tastendefinition und gibt auf dem Datenträger die einzelnen Zeichen aus. Diese müssen je nach ASCII-Wert in eine CHR\$-Angabe konvertiert werden. Letztere Aufgabe übernehmen die Zeilen 510 bis 620.

Sobald alle Zeichen ausgegeben sind, geht es in Zeile 700 mit einem CLOSEOUT- und einem END-Befehl weiter. Da stoppt das Programm, und der Computer reagiert wieder auf Ihre Eingaben. Das erzeugte Programm läßt sich ohne weiteres mit LOAD "Dateiname" in den Speicher laden und mit RUN "Dateiname" auch gleich starten.

## 24. Dateien kopiert und verbunden

Wenn Sie mit dem Schneider-CPC Dateien kopieren wollen, müssen Sie auf CP/M umschalten und diese Arbeit mit einem der mitgelieferten Hilfsprogramme durchführen.

Unter CP/M 2.2 sind dies PIP.COM ("Peripheral Interchange Program") und FILECOPY.COM. Bei CP/M 3.0 findet sich nur PIP.COM. Doch die Anwendung dieser Programme ist nicht gerade als komfortabel zu bezeichnen, denn Sie verlieren beim Laden von CP/M alle Daten und Programme im Speicher. Diese müßten Sie nach der Rückkehr aus CP/M wieder neu laden. Außerdem ist wahrscheinlich gerade in diesem Augenblick die CP/M-Systemdiskette unauffindbar, oder das Kopierprogramm steht auf der falschen Diskette. Und mit dem Auswechseln von Disketten unter CP/M 2.2 hat es so einige Schwierigkeiten auf sich: Disketten dürfen nur gewechselt werden, wenn Sie sie sofort mit Control-C "anmelden". Doch Control-C bricht auch jedes laufende Programm ab. Wenn Sie dann also erst noch das Kopierprogramm auf die Diskette mit der zu kopierenden Datei kopieren müssen, wird es doch etwas unangenehm.

Erheblich einfacher läßt sich diese Aufgabe mit der RSX-Erweiterung COPYFILE erledigen. Diese legt von ASCII-Textdateien Kopien auf derselben Diskette an. Wenn Sie ein zweites Laufwerk besitzen, können Sie die Datei auch dorthin übertragen.

Zu diesem Zweck wird das Schneider-Basic um den RSX-Befehl COPYFILE erweitert. Dieser besitzt die folgende Syntax:

```
a$="QUELLE"
b$="ZIEL"
!COPYFILE,@a$,@b$
```

Die beiden Stringvariablen enthalten also die beiden Dateinamen. Im Gegensatz zu PIP.COM steht der Name der Zieldatei rechts neben dem Namen der Quelldatei, was eigentlich auch logischer ist.

Das Maschinenprogramm COPYFILE.ASM ist von der Quellcode-Länge her recht umfangreich. Betrachtet man allerdings die Länge des assemblierten Programms, kann man sagen, daß es doch recht kurz geraten ist. Hier zuerst der Quellcode. Diesen können Sie entweder abtippen und assemblieren oder auch einfach nur studieren, wenn Sie neugierig sind, wie COPYFILE funktioniert:

### 24.1 Assemblerlisting: COPYFILE

```
; *****
; *
; *      RSX-Erweiterung COPYFILE zum Kopieren von ASCII-Dateien      *
; *                                                                 *
; *****

ORG      &A000      ; Mit dem Basic-Loader relocatibel!

;
```

380

```

; ***** VERWENDETE UNTERPROGRAMME *****
;

CAS_OPENIN      EQU      &BC77      ; CAS IN OPEN
CAS_INPUT       EQU      &BC80      ; CAS IN CHAR
CAS_CLOSEIN     EQU      &BC7A      ; CAS IN CLOSE
CAS_IN_ABANDON  EQU      &BC7D      ; CAS IN ABANDON
CAS_OPENOUT     EQU      &BC8C      ; CAS OUT OPEN
CAS_OUTPUT      EQU      &BC95      ; CAS OUT CHAR
CAS_CLOSEOUT    EQU      &BC8F      ; CAS OUT CLOSE
CAS_OUT_ABANDON EQU      &BC92      ; CAS OUT ABANDON
KL_U_ROM_ON     EQU      &B900      ; KL U ROM ENABLE
KL_LOG_EXT      EQU      &BCD1      ; KL LOG EXTERNAL

GET_INPBUFFER   EQU      &F632      ; CPC-664+CPC-6128 = &F720
REL_INPBUFFER   EQU      &F66D      ; CPC-664+CPC-6128 = &F759
GET_OUTBUFFER   EQU      &F637      ; CPC-664+CPC-6128 = &F725
REL_OUTBUFFER   EQU      &F671      ; CPC-664+CPC-6128 = &F75D
BASIC_ERROR     EQU      &CA94      ; CPC-664 = &CB50, CPC-6128 = &CB4D

;
; ***** INITIALISIERUNG DES RSX-KOMMANDOS *****
;

INIT_RSX_COPYF LD      BC,JUMPTABLE ; Zeiger auf RSX-Sprungtabelle
                LD      HL,KERNEL_RAM ; Zeiger auf Hilfsspeicher
                CALL    KL_LOG_EXT    ; RSX ins Betriebssystem einbinden
                RET                     ; Rücksprung nach Basic

JUMPTABLE       DEFW    NAMETABLE    ; Vektor auf Namenstabelle
                JP      COPYFILE     ; Sprung nach COPYFILE

NAMETABLE        DEFW    "COPYFIL"    ; RSX-Name "COPYFILE"
                DEFB    "E"&80        ; Wortende mit gesetztem 7.Bit
                DEFB    &00           ; Tabellenende durch Nullbyte

KERNEL_RAM       DEFS    &04          ; Vier Bytes reserviert für Kernel

;
; ***** PROGRAMMCODE DES COPYFILE-BEFEHLS *****
;

COPYFILE         PUSH    AF           ; Akku vor ROM-Routine sichern
                CALL    KL_U_ROM_ON   ; Basic-ROM einblenden
                POP      AF           ; Akku wiederherstellen
                CP       2            ; Werden 2 Parameter übergeben?
                JR       Z,PARNUMBER_OK ; Ja - alles OK!

PARAM_NUM_WRONG LD      E,22          ; Fehler OPERAND MISSING
                JP      BASIC_ERROR   ; Fehlermeldung ausgeben

PARNUMBER_OK     PUSH    IX           ; IX wird von CAS-Routinen
                POP      IY           ; zerstört, IY hingegen nicht!

OPEN_INPUTFILE   CALL    GET_INPBUFFER ; Eingabepuffer bereitstellen
                PUSH    DE           ; DE-Register sichern
                LD      H,(IY+3)      ; Highbyte der Adresse 1.Dateiname

```

```

        LD      L,(1Y+2)      ; Und dazu das Lowbyte
        CALL    STRING_PARAMS ; Stringparameter dazu holen
        POP     DE            ; Zeiger auf 2K-Puffer zurUckholen
        CALL    CAS_OPENIN    ; OPENIN "File"
        JR      NC,FILE_ERR   ; Bei Fehler dorthin springen

OPEN_OUTPUTFILE CALL    GET_OUTBUFFER ; Ausgabepuffer bereitstellen
        PUSH    DE            ; DE-Register sichern
        LD      H,(1Y+1)      ; Highbyte der Adresse 3.Dateiname
        LD      L,(1Y+0)      ; Und dazu das Lowbyte
        CALL    STRING_PARAMS ; Stringparameter dazu holen
        POP     DE            ; Ausgabepuffer-Adresse zurUckholen
        CALL    CAS_OPENOUT   ; OPENOUT "File"
        JR      NC,FILE_ERR   ; Bei Fehler dorthin springen

READ_LOOP. CALL    CAS_INPUT    ; Ein Zeichen aus der 1.Datei lesen
        JR      NC,EOF_FILE    ; Ende der ersten Datei erreicht?
        CALL    CAS_OUTPUT    ; Akku ins Ausgabeffile Uebertragen
        JR      READ_LOOP     ; Solange, bis Dateiende erreicht

EOF_FILE CALL    CAS_CLOSEIN    ; Eingabedatei 2 schlieBen
        CALL    REL_INPBUFFER   ; Eingabepuffer wieder freigeben
        CALL    CAS_CLOSEOUT   ; Ausgabedatei schlieBen
        CALL    REL_OUTBUFFER   ; Ausgabepuffer wieder freigeben
        RET                    ; Rucksprung nach Basic

;
; ***** UNTERPROGRAMM - ZEIGER AUF DATEINAMEN HOLEN *****
;

STRING_PARAMS LD      B,(HL)    ; Stringlnge ins B-Register
        INC     HL             ; Zeiger erhhen
        LD      E,(HL)        ; Lowbyte der Stringadresse
        INC     HL             ; Zeiger auf das Highbyte
        LD      D,(HL)        ; Und das Highbyte lesen
        EX      DE,HL          ; Adresse des Dateinamens nach HL
        RET                    ; Rucksprung zum Hauptprogramm

;
; ***** BEI FEHLER IN OPENIN/OPENOUT WIRD HIERHER GESPRUNGEN *****
;

FILE_ERR POP     DE            ; Stack fr Rucksprung korrigieren

FILE_ERR2 CALL    CAS_IN_ABANDON ; Dateibehandlung abbrechen
        CALL    CAS_OUT_ABANDON
        RET                    ; Und Rucksprung nach Basic

        END

```

## 24.2 Listing: COPYFILE.BAS

```

100 ' *****
110 ' *
120 ' *          COPYFILE          *
130 ' *

```

```

140 ' *****
150 '
160 ' Definitionen und Titelbild *****
170 DEF FNisb(a)=255 AND UNT(a)
180 DEF FNmsb(a)=255 AND INT(a/256)
190 SYMBOL AFTER 256:MEMORY HIMEM-126
200 start=HIMEM+1:SYMBOL AFTER 240
210 ' Initialisierung des Basic-Loaders *****
220 SYMBOL 253,&66,&0,&78,&C,&7C,&CC,&76,&0
230 SYMBOL 254,&66,&0,&3C,&66,&66,&66,&3C,&0
240 SYMBOL 255,&66,&0,&66,&66,&66,&66,&3E,&0
250 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
260 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
270 ' Titelbild und Benutzerinformationen *****
280 LOCATE 1,1:PRINT STRING$(40,210);
290 LOCATE 1,2
300 PRINT CHR$(24)+SPACE$(16)+"COPYFILE"+SPACE$(16)+CHR$(24)
310 PRINT:PRINT STRING$(40,210):PRINT
320 PEN 1:PRINT " COPYFILE erweitert das Basic um einen"
330 PRINT:PRINT " Befehl, mit dem Sie auch ohne PIP.COM"
340 PRINT:PRINT " Dateien kopieren k"+CHR$(254)+"nnen:"
350 PEN 3:PRINT:PRINT STRING$(40,210):PRINT
360 PEN 2:PRINT TAB(5);"A*="+CHR$(34)+"FILE1"+CHR$(34)
370 PRINT:PRINT TAB(5);"B*="+CHR$(34)+"FILE2"+CHR$(34)
380 PRINT:PRINT TAB(5);"!COPYFILE,0A*,0B*"
390 PRINT:PRINT:PEN 1
400 FOR i=&A000 TO &A07A:READ a:sum=sum+a:NEXT i
410 IF sum=16824 THEN 450
420 PRINT CHR$(7);"* Fehler in den DATA-Zeilen!":PRINT
430 LIST 650-:END
440 ' Das Maschinenprogramm wird eingePOKEt und verschoben *****
450 RESTORE:FOR i=start TO start+&7A:READ a:POKE i,a:NEXT i
460 FOR i=1 TO 6:READ a:a=a-40960+start
470 wert=PEEK(a)+PEEK(a+1)*256-40960+start
480 POKE a,FNisb(wert):POKE a+1,FNmsb(wert)
490 NEXT i
500 CALL start ' RSX COPYFILE ins System einbinden
510 ' Computertyp feststellen *****
520 POKE &160,&CD:POKE &161,&0:POKE &162,&B9 ' CALL &B900
530 POKE &163,&3A:POKE &164,&2:POKE &165,&C0 ' LD A,(&C002)
540 POKE &166,&32:POKE &167,&60:POKE &168,&1 ' LD (&0160),A
550 POKE &169,&C9 ' RET
560 CALL &160:cpcversion=PEEK(&160)
570 ' Programm an den verwendeten Computer anpassen *****
580 IF cpcversion=0 THEN END ' CPC-464 benoetigt keine Aenderung!
590 POKE start+&29,&CB
600 IF cpcversion=1 THEN POKE start+&28,&50 ELSE POKE start+&28,&4D
610 POKE start+&2F,&20:POKE start+&30,&F7
620 POKE start+&42,&25:POKE start+&43,&F7
630 POKE start+&62,&59:POKE start+&63,&F7
640 POKE start+&88,&5D:POKE start+&69,&F7
650 ' Maschinencode-DATAs *****
660 DATA &01,&0A,&A0,&21,&18,&A0,&CD,&D1,&BC,&C9,&0F,&A0,&C3,&1C,&A0,&43
670 DATA &4F,&50,&59,&46,&49,&4C,&C5,&00,&00,&00,&00,&F5,&CD,&00,&B9
680 DATA &F1,&FE,&02,&28,&05,&1E,&16,&C3,&94,&CA,&DD,&E5,&FD,&E1,&CD,&32
690 DATA &7A,&F6,&D5,&FD,&66,&03,&FD,&6E,&02,&CD,&6B,&A0,&D1,&CD,&77,&BC,&30
700 DATA &31,&CD,&37,&F6,&D5,&FD,&66,&01,&FD,&6E,&00,&CD,&6B,&A0,&D1,&CD

```



```

710 DATA &BC,&BC,&30,&1E,&CD,&80,&BC,&30,&05,&CD,&95,&BC,&18,&F6,&CD,&7A
720 DATA &BC,&CD,&6D,&F6,&CD,&8F,&BC,&CD,&71,&F6,&C9,&46,&23,&5E,&23,&56
730 DATA &EB,&C9,&D1,&CD,&7D,&BC,&CD,&92,&BC,&C9,&00
740 ' DATAs zum Relokalisieren des Maschinencodes *****
750 DATA &A001,&A004,&A00A,&A00D,&A039,&A04C
760 END ' -----

```

COPYFILE selbst gibt keinerlei Meldungen auf dem Bildschirm aus, denn die Datei wird sozusagen "in einem Rutsch" kopiert. Fehlermeldungen können allerdings durchaus auftauchen, zum Beispiel, wenn Sie nicht genau zwei Parameter beim Aufruf angeben oder die übergebenen Dateinamen nicht den Vorschriften entsprechen.

Das Maschinenprogramm ist nicht frei verschiebbar, wird aber vom Basic-Loader so verändert, daß es direkt unter HIMEM zu liegen kommt. Der Loader nimmt zusätzlich die Anpassung an den CPC-664 und CPC-6128 vor. Wenn Sie das Programm nur auf einem CPC-464 betreiben wollen und sicher sind, daß sich das auch in Zukunft nicht ändert, können Sie die Zeilen 510 bis 640 weglassen. Diese sind nämlich für die Anpassung zuständig.

Die Funktionsweise von COPYFILE ist recht einfach, sie wird nur durch die zwangsläufige "Ausführlichkeit" von Maschinenprogrammen verdeckt. In COPYFILE schaltet das Programm das Basic-ROM ein (wichtig zur Benutzung der Routinen im Basic-ROM!) und stellt fest, ob Sie wirklich zwei Parameter übergeben haben. Gegebenenfalls erzeugt PARAM\_NUM\_WRONG eine Basic-Fehlermeldung.

In PARNUMBER\_OK wird der Zeiger auf den Parameterblock, der sich bei Maschinenprogrammen im Indexregister IX befindet, ins IY-Indexregister übertragen.

Dies muß geschehen, weil diverse CAS-Dateiroutinen des Betriebssystems das IX-Register zerstören dürfen, während das IY-Register unangetastet bleibt. Da die beiden Parameter nicht sofort ins Programm übernommen werden, muß der Zeiger auf sie erhalten bleiben. Und im IY-Register ist er nun einmal sicher.

OPEN\_INPUTFILE reserviert einen Eingabepuffer von 2 Kilobytes Größe und holt sich den Dateinamen der einzulesenden Datei. Mit CALL CAS\_OPENIN wird die Datei geöffnet. Sollte dabei ein Fehler gemeldet werden, geht es für den Computer bei FILE\_ERR weiter.

OPEN\_OUTPUTFILE arbeitet recht ähnlich. Hier wird ein Ausgabepuffer bereitgestellt, und anhand des Dateinamens wird eine Ausgabedatei auf der Diskette erzeugt. Auch hier verzweigt das Programm bei einem Fehler nach FILE\_ERR.

READ\_LOOP liest mit CAS IN CHAR jeweils ein Zeichen aus der ersten Datei und überträgt es mit CAS OUT CHAR in die andere. Sobald bei CAS IN CHAR das Carry-Flag nicht mehr gesetzt ist - so heißen die Betriebssystem-Konventionen -, ist die Leseroutine am Ende der Datei angelangt. Dann geht es für COPYFILE bei EOF\_FILE weiter. Dort werden die beiden Dateien geschlossen, und die Pufferspeicher werden freigegeben.

Ähnlich wie COPYFILE ist CONCAT aufgebaut. Dies ist wieder eine RSX-Erweiterung. Sie fügt zwei Eingabedateien zu einer dritten zusammen. So können Sie zum Beispiel einzelne Dateien eines Textverarbeitungsprogramms zu einem großen File zusammenstellen. CONCAT ist etwas unübersichtlicher bei seinem Aufruf, da Sie ja drei Dateien angeben müssen:

```
Eingabe1$="TEXT.A"
Eingabe2$="TEXT.B"
Ausgabe$="TEXT.A+B"
!CONCAT,@Eingabe1$,@Eingabe2$,@Ausgabe$
```

#### 24.3 Assemblerlisting: CONCAT

```
; *****
; *
; *      RSX-Erweiterung CONCAT zur Verknüpfung zweier Dateien      *
; *
; *****

                ORG      &A000          ; Mit dem Basic-Loader relocierbar!

;
; ***** VERWENDETE UNTERPROGRAMME *****
;

CAS_OPENIN      EQU      &BC77          ; CAS IN OPEN
CAS_INPUT       EQU      &BC80          ; CAS IN CHAR
CAS_CLOSEIN     EQU      &BC7A          ; CAS IN CLOSE
CAS_IN_ABANDON  EQU      &BC7D          ; CAS IN ABANDON
CAS_OPENOUT     EQU      &BC8C          ; CAS OUT OPEN
CAS_OUTPUT      EQU      &BC95          ; CAS OUT CHAR
CAS_CLOSEOUT    EQU      &BC8F          ; CAS OUT CLOSE
CAS_OUT_ABANDON EQU      &BC92          ; CAS OUT ABANDON
KL_U_ROM_ON     EQU      &B900          ; KL U ROM ENABLE
KL_LOG_EXT      EQU      &BCD1          ; KL LOG EXTERNAL

GET_INPBUFFER   EQU      &F632          ; CPC-664+CPC-6128 = &F720
REL_INPBUFFER   EQU      &F66D          ; CPC-664+CPC-6128 = &F759
GET_OUTBUFFER   EQU      &F637          ; CPC-664+CPC-6128 = &F725
REL_OUTBUFFER   EQU      &F671          ; CPC-664+CPC-6128 = &F75D
BASIC_ERROR     EQU      &CA94          ; CPC-664 = &CB50, CPC-6128 = &CB4D

;
; ***** INITIALISIERUNG DES RSX-KOMMANDOS *****
;

INIT_RSX_CONCAT LD      BC,JUMPTABLE    ; Zeiger auf RSX-Sprungtabelle
                LD      HL,KERNEL_RAM  ; Zeiger auf Hilfsppeicher
                CALL    KL_LOG_EXT      ; RSX ins Betriebssystem einbinden
                RET                     ; Rücksprung nach Basic

JUMPTABLE       DEFW     NAMETABLE      ; Vektor auf Namenstabelle
                JP      CONCAT          ; Sprung nach CONCAT

NAMETABLE       DEFM     "CONCA"        ; RSX-Name "CONCAT"
```

```

DEFB      "T"&80      ; Wortende mit gesetztem 7.Bit
DEFB      &00         ; Tabellenende durch Nullbyte

KERNEL_RAM  DEFS      &04      ; Vier Bytes reserviert für Kernel

;
; ***** PROGRAMMCODE DES CONCAT-BEFEHLS *****
;

CONCAT      PUSH      AF        ; Akku vor ROM-Routine sichern
            CALL      KL_U_ROM_ON ; Basic-ROM einblenden
            POP       AF        ; Akku wiederherstellen
            CP        3         ; Werden 3 Parameter übergeben?
            JR        Z,PARNUMBER_OK ; Ja - alles OK!

PARAM_NUM_WRONG LD      E,22      ; Fehler OPERAND MISSING
                JP      BASIC_ERROR ; Fehlermeldung ausgeben

PARNUMBER_OK  PUSH      IX        ; IX wird von CAS-Routinen
                POP      IY        ; zerstört, IY hingegen nicht!

OPEN_INPUTFILE1 CALL     GET_INPBUFFER ; Eingabepuffer bereitstellen
                PUSH     DE        ; DE-Register sichern
                PUSH     DE        ; Für später nochmals sichern
                LD       H,(IY+5)   ; Highbyte der Adresse 1.Dateiname
                LD       L,(IY+4)   ; Und dazu das Lowbyte
                CALL     STRING_PARAMS ; Stringparameter dazu holen
                POP      DE        ; Zeiger auf 2K-Puffer zurückholen
                CALL     CAS_OPENIN  ; OPENIN "File"
                JR        NC,FILE_ERR ; Bei Fehler dorthin springen

OPEN_OUTPUTFILE CALL     GET_OUTBUFFER ; Ausgabepuffer bereitstellen
                PUSH     DE        ; DE-Register sichern
                LD       H,(IY+1)   ; Highbyte der Adresse 3.Dateiname
                LD       L,(IY+0)   ; Und dazu das Lowbyte
                CALL     STRING_PARAMS ; Stringparameter dazu holen
                POP      DE        ; Ausgabepuffer-Adresse zurückholen
                CALL     CAS_OPENOUT ; OPENOUT "File"
                JR        NC,FILE_ERR ; Bei Fehler dorthin springen

READ_LOOP_1   CALL      CAS_INPUT   ; Ein Zeichen aus der 1.Datei lesen
                JR        NC,EOF_FILE_1 ; Ende der ersten Datei erreicht?
                CALL      CAS_OUTPUT ; Akku ins Ausgabefile übertragen
                JR        READ_LOOP_1 ; Solange, bis Dateiende erreicht

EOF_FILE_1    CALL      CAS_CLOSEIN ; Eingabedatei 1 schließen

OPEN_INPUTFILE2 LD       H,(IY+3)   ; Highbyte der Adresse 2.Dateiname
                LD       L,(IY+2)   ; Und dazu das Lowbyte
                CALL     STRING_PARAMS ; Stringparameter dazu holen
                POP      DE        ; Eingabepuffer-Adresse zurückholen
                CALL     CAS_OPENIN  ; Die zweite Eingabedatei öffnen
                JR        NC,FILE_ERR2 ; Bei Fehler dorthin springen

READ_LOOP_2   CALL      CAS_INPUT   ; Ein Zeichen der Datei lesen
                JR        NC,EOF_FILE_2 ; Ende der zweiten Datei?
                CALL      CAS_OUTPUT ; Akku in die Ausgabedatei schicken

```

```

JR      READ_LOOP_2      ; Weitermachen, bis EOF erreicht

EOF_FILE_2  CALL      CAS_CLOSEIN      ; Eingabedatei 2 schließen
            CALL      REL_INPBUFFER    ; Eingabepuffer wieder freigeben
            CALL      CAS_CLOSEOUT     ; Ausgabedatei schließen
            CALL      REL_OUTBUFFER    ; Ausgabepuffer wieder freigeben
            RET              ; Rücksprung nach Basic

;
; ***** UNTERPROGRAMM - ZEIGER AUF DATEINAMEN HOLEN *****
;

STRING_PARAMS  LD      B,(HL)          ; Stringlänge ins B-Register
              INC      HL              ; Zeiger erhöhen
              LD      E,(HL)          ; Lowbyte der Stringadresse
              INC      HL              ; Zeiger auf das Highbyte
              LD      D,(HL)          ; Und das Highbyte lesen
              EX      DE,HL           ; Adresse des Dateinamens nach HL
              RET              ; Rücksprung zum Hauptprogramm

;
; ***** BEI FEHLER IN OPENIN/OPENOUT WIRD HIERHER GESPRUNGEN *****
;

FILE_ERR      POP      DE              ; Stack für Rücksprung korrigieren

FILE_ERR2     CALL      CAS_IN_ABANDON ; Dateibehandlung abbrechen
              CALL      CAS_OUT_ABANDON
              RET              ; Und Rücksprung nach Basic

END
; -----

```

#### 24.4 Listing: CONCAT.BAS

```

100 ' *****
110 ' *
120 ' *   CONCAT: Verbindet zwei Dateien zu einer dritten
130 ' *
140 ' *****
150 '
160 ' Initialisierung und Benutzerinformationen *****
170 DEF FNlsb(a)=255 AND INT(a)
180 DEF FNmsb(a)=255 AND INT(a/256)
190 SYMBOL AFTER 256:MEMORY HIMEM-160:start=HIMEM+1:SYMBOL AFTER 240
200 SYMBOL 255,&66,&0,&3C,&66,&66,&66,&3C,&0
210 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
220 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
230 LOCATE 1,1:PRINT STRING$(40,210);
240 LOCATE 1,2
250 PRINT CHR$(24)+SPACE$(17)+"CONCAT"+SPACE$(17)+CHR$(24)
260 LOCATE 10,5:PRINT CHR$(164);" Martin Kotulla 1986"
270 PRINT:PRINT STRING$(40,210):PRINT
280 PEN 1:PRINT " Mit einem neuen RSX-Befehl k"+CHR$(255)+"nnen Sie"
290 PRINT:PRINT " zwei Diskettendateien zu einer neuen"
300 PRINT:PRINT " verbinden. Die Syntax des CONCAT-"
310 PRINT:PRINT " Befehls ist folgende:"

```

```

320 PEN 3:PRINT:PRINT STRING$(40,210)
330 PEN 2:PRINT:PRINT " 1:CONCAT,@infile1$,@infile2$,@outfile$"
340 PRINT:PEN 1
350 ' Pruefsumme feststellen und entsprechend reagieren *****
360 FOR i=&A000 TO &A095:READ a:sum=sum+a:NEXT i
370 IF sum=20708 THEN 410
380 PRINT CHR$(7);"* Fehler in den DATAs!":PRINT
390 LIST 600-:END
400 ' Maschinenprogramm lesen und einPOKEN *****
410 RESTORE:FOR i=start TO start+&95:READ a:POKE i,a:NEXT i
420 ' Jetzt muss es noch verschoben werden *****
430 FOR i=1 TO 7:READ a:a=a-40960+start
440 wert=PEEK(a)+PEEK(a+1)*256-40960+start
450 POKE a,FN:SB(wert):POKE a+1,FN:MSB(wert)
460 NEXT i
470 CALL start ' RSX "CONCAT" initialisieren
480 ' Die Anpassung an den 664/6126 wird nicht vergessen! *****
490 POKE &160,&CD:POKE &161,&0:POKE &162,&B9:POKE &163,&3A:POKE &164,&2
500 POKE &165,&C0:POKE &166,&32:POKE &167,&60:POKE &168,&1:POKE &169,&C9
510 CALL &160:cpcvsn=PEEK(&160)
520 IF cpcvsn=0 THEN END ' weil es ein CPC-464 ist!
530 POKE start+&27,&CB
540 IF cpcvsn=1 THEN POKE start+&26,&50 ELSE POKE start+&26,&4D
550 POKE start+&2D,&20:POKE start+&2E,&F7
560 POKE start+&41,&25:POKE start+&42,&F7
570 POKE start+&7D,&59:POKE start+&7E,&F7
580 POKE start+&83,&5D:POKE start+&84,&F7
590 ' Maschinencode-Zeilen *****
600 DATA &01,&0A,&A0,&21,&16,&A0,&CD,&D1,&BC,&C9,&0F,&A0,&C3,&1A,&A0,&43
610 DATA &4F,&4E,&43,&41,&D4,&00,&00,&00,&00,&00,&F5,&CD,&00,&B9,&F1,&FE
620 DATA &03,&28,&05,&1E,&16,&C3,&94,&CA,&DD,&E5,&FD,&E1,&CD,&32,&F6,&D5
630 DATA &D5,&FD,&66,&05,&FD,&6E,&04,&CD,&86,&A0,&D1,&CD,&77,&BC,&30,&4D
640 DATA &CD,&37,&F6,&D5,&FD,&66,&01,&FD,&6E,&00,&CD,&86,&A0,&D1,&CD,&8C
650 DATA &BC,&30,&3A,&CD,&80,&BC,&30,&05,&CD,&95,&BC,&18,&F6,&CD,&7A,&BC
660 DATA &FD,&66,&03,&FD,&6E,&02,&CD,&86,&A0,&D1,&CD,&77,&BC,&30,&1F,&CD
670 DATA &80,&BC,&30,&05,&CD,&95,&BC,&18,&F6,&CD,&7A,&BC,&CD,&6D,&F6,&CD
680 DATA &8F,&BC,&CD,&71,&F6,&C9,&46,&23,&5E,&23,&56,&EB,&C9,&D1,&CD,&7D
690 DATA &BC,&CD,&92,&BC,&C9,&00
700 ' DATAs zum Relokalisieren des Maschinenprogramms *****
710 DATA &A001,&A004,&A00A,&A00D,&A038,&A04B,&A067
720 END '

```

Wie COPYFILE prüft auch CONCAT die Zahl der übergebenen Parameter nach und gibt im Bedarfsfall eine Fehlermeldung aus. Die erste Eingabedatei und die Ausgabedatei werden wie gehabt geöffnet. In READ LOOP 1 kopiert Concat die erste Datei in die Ausgabedatei.

Sobald das End-Of-File, das Dateiende also, erreicht ist, schließt Concat die Datei und öffnet die zweite Eingabedatei. Nach demselben Schema kopiert Concat auch die zweite Datei in die Ausgabedatei und schließt beide Dateien. Dann werden die Ein- und die Ausgabepuffer freigegeben, und es erfolgt ein Rücksprung nach Basic.

## 25. Listschutz auf CPC-664 und 6128

Inzwischen gibt es haufenweise Routinen, mit denen sich der Listschutz von Programmen, die versehentlich mit der P-Option abgespeichert wurden, wieder aufheben läßt. Leider ist aber keine von ihnen auf dem CPC-664 und CPC-6128 lauffähig.

Die wahrscheinlich kürzeste und dennoch einwandfrei funktionierende Lösung zum Aufheben des Listschutzes auf dem CPC-464 besteht aus genau sechs POKE-Befehlen:

```
POKE &30,&AF:POKE &31,&32:POKE &32,&45
POKE &33,&AE:POKE &34,&C9:POKE &AC01,&F7

LOAD "PROGRAMM"
```

Im Quellformat des Maschinenprogramms sind das die folgenden Z80-Befehle:

```
&0030 XOR A ; Akku löschen
&0031 LD (&AE45),A ; Nach &AE45 Übertragen
&0034 RET ; Rücksprung

&AC01 RST 6 ; RST 6 geht nach &0030
```

In der Speicherstelle &AE45 steht nach dem Laden eines geschützten Programms ein Wert ungleich Null. Wenn der Computer in den Ready-Modus zurückkehrt, prüft er dieses Flag und löscht gegebenenfalls den Speicher. Im Basic-ROM schaut das disassembliert so aus:

```
&C064 CALL &AC01
&C067 LD SP,&C000
&C06A CALL &C162
&C06D CALL &DD06
&C070 CALL C,SOUND_HOLD
&C073 CALL KM_DISARM_BREAK
&C076 CALL &C386
&C079 LD A,(&AE45)
&C07C OR A
&C07D CALL NZ,&C13E
```

Der Einsprung in den Ready-Modus geschieht an der Adresse &C064. In &C079 lädt der Computer den Akkumulator des Prozessors mit dem Wert des Protected-Flags. Wenn sich ein Wert ungleich Null ergibt (OR A), wird das Programm durch einen Sprung zur Unteroutine &C13E gelöscht.

Ein einfacher POKE-Befehl wie POKE &AE45,0 ist nutzlos, da der Computer ja die Speicherstelle sofort beim Rücksprung in den Editor-Modus abfragt. Glücklicherweise haben die Amstrad-Programmierer sogenannte "Dummy>Returns" vorgesehen. An bestimmten Stellen im Basic-ROM steht ein Sprungbefehl mitten in das normale RAM hinein. An den Zieladressen befinden sich normalerweise nur RETs

(Z80-Opcode &C9), so daß die Sprünge ohne Wirkung bleiben. Jeder Dummy-Return besteht aus drei RETs. Somit ist genügend Platz für einen absoluten Sprung (JP) vorhanden.

Setzt der Programmierer einen Sprungbefehl auf eine spezielle Routine ein, wird diese immer wieder aufgerufen. Meist dienen die Dummy>Returns dazu, neue Basic-Befehle oder Funktionen zu implementieren. Es gibt folgende Aufrufe:

```
&AC01 Sprung in den Ready-Modus
&AC04 Sprung zur Fehlerausgabe
&AC07 Sprung zur Routine, die Befehle interpretiert
&AC0A Sprung zur Routine, die Funktionen berechnet
&AC0D Sprung zur Routine, die Konstanten hoit
&AC10 Sprung zur Routine, die Eingabezeilen tokenisiert
&AC13 Sprung zur LIST-Routine (Decodierung der Tokens)
&AC16 Sprung zur Routine, die Zahlen umwandelt
&AC19 Sprung zur Routine, die Rechenoperatoren hoit
```

Ohne weitere Informationen helfen die Adressen aber nicht viel. Hier deshalb noch die Adressen, an denen die Sprünge in die Dummy>Returns stehen. Sie befinden sich alle im Basic-ROM zwischen &C000 (49152) und &FFFF (65535):

```
&AC01 - &C064 = &CD &01 &AC CALL &AC01
&AC04 - &CA94 = &CD &04 &AC CALL &AC04
&AC07 - &DDC3 = &CD &07 &AC CALL &AC07
&AC0A - &D0A9 = &CD &0A &AC CALL &AC0A
&AC0D - &D078 = &CD &0D &AC CALL &AC0D
&AC10 - &DEE1 = &CD &10 &AC CALL &AC10
&AC13 - &E196 = &CD &13 &AC CALL &AC13
&AC16 - &DF51 = &CD &16 &AC CALL &AC16
&AC19 - &E30B = &CD &19 &AC CALL &AC19
```

Durch den Eintrag eines Sprungbefehls in die Adresse &AC01 kann ein Unterprogramm eingebunden werden, das aktiv wird, bevor der Computer das Protected-Flag prüfen kann. Als Speicherplatz für das Maschinenprogramm bietet sich der Restart RST6 an der Adresse &0030 an. Dieser Restart wurde von Amstrad für den persönlichen Gebrauch des Computer-Besitzers freigegeben. Durch den Eintrag &F7 in &AC01 wird der Restart angesprungen. Da hinterher in den Adressen &AC02 und &AC03 noch RETs stehen, muß nicht einmal ein Rücksprungbefehl in den Dummy-Return gePOKEt werden.

In den Adressen &0030 bis &0034 steht der bekannte Maschinencode: XOR A löscht den Akku und kostet im Gegensatz zu LD A,0 nur ein Byte Speicherplatz. Mit LD (&AE45),A wird der Wert Null in das Flag übertragen, und RET bewirkt den Rücksprung in den Dummy-Return. Von dortaus erfolgt der Rücksprung ins Basic-ROM.

Diese elegante Programmierung ist auf dem CPC-664 und CPC-6128 nicht mehr möglich. Denn Amstrad hat aus unerklärlichen Gründen die Dummy>Returns bei diesen Computern wieder weggelassen. Doch mit einigen Tricks ist das "Entschützen" trotzdem durchführbar.

Schauen wir uns doch einmal das ROM-Listing an, wie es aus dem Disassembler kommt:

Adresse	CPC-664	CPC-6128
&C058	LD SP,&C000	LD SP,&C000
&C05B	CALL &C166	CALL &C166
&C05E	GALL &DEBA	CALL &DEB5
&C061	CALL C,&BCB6	CALL C,&BCB6
&C064	CALL &C4D3	CALL &C4D0
&C067	CALL &C3D3	CALL &C3D0
&C06A	LD A,(&AE2C)	LD A,(&AE2C)
&C06D	OR A	OR A
&C06E	CALL NZ,&C145	CALL NZ,&C145

Entscheidend ist hier die Adresse &C06A: Dort wird der Akku mit dem Inhalt des Protected-Flags geladen. Dessen Speicheradresse wurde beim CPC-664 und CPC-6128 von &AE45 nach &AE2C verlegt. Ist der Flag-Wert ungleich Null, wird das Basic-Programm gelöscht (Verzweigung nach &C145). Die einzige Lösung für unser Problem findet sich in der Adresse &C061: Dort verzweigt der Computer über die RAM-Sprüngeleiste ins Unterprogramm SOUND HOLD. Da die Verzweigung in diese Routine des Betriebssystems über einen Vektor im RAM erfolgt, läßt sich hier ein Programm integrieren.

Der Aufruf von SOUND HOLD erfolgt nur, wenn das Carry-Flag des Z80-Prozessors gesetzt ist. Dies wird von der vorhergehenden Routine (&C05E) gesetzt, wenn der LOAD-Befehl aus einem Basic-Programm heraus erfolgt, nicht jedoch bei Eingabe der Befehle im Direkt-Modus. Tippen Sie also das folgende Mini-Programm ab:

```
10 POKE &30,&97:POKE &31,&32:POKE &32,&2C
20 POKE &33,&AE:POKE &34,&C9
30 POKE &BCB6,&F7 ' RST6 = User-Restart
40 POKE &BCB7,&C9
50 LOAD "Programm" ' hier Namen einsetzen
```

```
RUN
CALL &BD37
!DISC ' nur bei der Floppy verwenden!
```

Auch hier dient der User-Restart RST 6 dazu, den Maschinencode aufzunehmen. Nach dem RUN-Befehl müssen Sie warten, bis die Ready-Meldung erscheint. Jetzt können Sie das Programm listen. Sie sollten aber noch ein CALL &BD37 nachschicken. Dieser Aufruf springt JUMP RESTORE an. Die Systemroutine stellt alle Betriebssystem-Vektoren wieder auf ihre Ursprungswerte. Davon sind auch die Disketten-Vektoren betroffen, so daß alle Dateibefehle wieder auf den Cassettenrecorder wirken. Der RSX-Befehl !DISC macht dies wieder rückgängig.

Nach diesen Maßnahmen können Sie das "entschützte" Programm listen, starten und abspeichern. Übrigens eignet sich das Hilfsprogramm nur dazu, Ihre eigene versehentlich mit SAVE "xx",P gespeicherte Software wieder listbar zu machen. Raubkopierer haben keine Chance: Wohl keine Softwarefirma wird sich mehr auf diesen unsicheren Programmschutz einlassen, zumal derartige Unprotect-Programme schon seit längerer Zeit weitverbreitet sind - bisher eben nur für den CPC-464.



## 26. Der Screen-Editor

Immer wenn in Maschinenprogrammen eine Simulation des INPUT-Befehls benötigt wird, versuchen die meisten Programmierer dies ganz einfach aus Bequemlichkeit zu umgehen: Denn alle Editor-Funktionen nachzubilden, erfordert einen ungeheuren Programmieraufwand.

Dabei können zumindest Schneider-Programmierer auf den eingebauten Bildschirm-Editor zurückgreifen! Der folgende Quellcode einer Maschinenroutine zeigt, wie einfach das geht: Das HL-Register wird mit der Adresse des Pufferspeichers geladen, in den die gelesenen Zeichen eingetragen werden - und ab geht's.

```
;
; DEMO - Screen-Editor aufrufen
; -----

      ORG      &A000

EDITOR EQU     &BD3A           ; 664=&BD5B, 6128=&BD5E

      LD       HL,BUFFER
      CALL     EDITOR
      RET

BUFFER DEFS    256
      END
```

Je nach Computertyp muß allerdings eine andere Adresse für den Sprungvektor des Editors eingesetzt werden:

```
CPC-464: CALL &BD3A
CPC-664: CALL &BD5B
CPC-6128: CALL &BD5E
```

Das Betriebssystem zeigt übrigens vorher den gesamten Pufferinhalt von bis zu 255 Zeichen auf dem Bildschirm an. Die Ausgabe wird gestoppt, sobald ein Null-Byte entdeckt wird.

Durch LD IX,BUFFER und LD (IX+30),0 können Sie zum Beispiel die ersten 30 Zeichen des Puffers übernehmen. Der Benutzer kann in diesen Zeichen beliebig mit dem Cursor herumfahren und sie übernehmen. Eventuell braucht er nur noch einzelne Zeichen auszubessern - ein Komfort, den INPUT in Basic bekanntlich nicht bietet.

Das hat aber den Nachteil, daß das Programm selbst herausfinden muß, wo die Eingabezeile endet. Dies geschieht durch Abfrage des Pufferspeichers auf das Null-Byte hin.

Das kurze Basic-Programm demonstriert den Aufruf des Screen-Editors. Die Zeile 130 lädt dabei den Eingabepuffer mit dem gesamten Zeichensatz des Computers. In Zeile 140 erscheint die Aufforde-

rung "ihre Eingabe:", und der Computer ruft die Maschinenroutine ab Adresse &A000 auf.

#### 26.1 Listing: SCREENED

```
100 MEMORY &9FFF:CLS
110 DATA &21,&07,&A0,&CD,&3A,&BD,&C9
120 FOR i=&A000 TO &A006:READ a:POKE i,a:NEXT i
130 FOR i=&A007 TO &A107:POKE i,i AND 255:NEXT i
140 PRINT "ihre Eingabe: ";;CALL &A000
150 FOR i=&A007 TO &A107:PRINT CHR$(1)CHR$(PEEK(i));:NEXT
```

## 27. SYNOPSIS – Der Durchblick beim CPC

"Synopsis", auf Englisch "Synopsis", nennt man eine theologische Wissenschaft, die es sich zur Aufgabe gemacht hat, die drei ersten Evangelien des Neuen Testaments zu vergleichen und Gemeinsamkeiten aufzuzeigen. So etwas Ähnliches leistet das Programm SYNOPSIS 6128 beim Schneider CPC-6128. Doch lesen Sie selbst ...

Wenn Sie in Maschinensprache auf Ihrem Schneider-CPC programmieren oder Basic-Programme erstellen, die sich durch besonders trickreiche Anwendung von PEEKs und POKEs auszeichnen, werden Sie innerhalb kurzer Zeit feststellen, daß es enorme Schwierigkeiten macht, diese Programme auf andere Schneider-Computer zu übertragen. Wenn Sie etwa auf Ihrem CPC-464 die interne Speicherstelle auslesen wollen, die die Aufzeichnungsgeschwindigkeit des Cassettenrecorders enthält (SPEED WRITE), benutzen Sie dazu die Adressen &B8D1 und &B8D2: PRINT PEEK(&B8D1);PEEK(&B8D2).

Doch derselbe Leseversuch beim CPC-664 und CPC-6128 geht garantiert schief, und Sie landen irgendwo in den Tiefen des Computers. Bei diesen Modellen liegt die Adresse nämlich bei &B1E9 und &B1EA.

Um die verschiedenen Adressen von Systemvariablen und ROM-Routinen zu erfahren, benötigt man manchmal wirklich soviel Knebeleien und Rätselraten wie die Wissenschaftler, die sich mit der Synopse der Evangelien beschäftigen. Wohl kaum jemand wird alle drei Schneider-Computer zu Hause stehen haben, um alle Vergleiche direkt durchführen zu können. Und nach einigen Besuchen beim Freund mit dem CPC-664 oder CPC-6128 wird dieser ganz bestimmt nicht mehr erfreut sein, wenn Sie sich immer mit dem Computer in eine Ecke zurückziehen und nach ein paar Stunden wieder verschwinden.

Doch Rettung ist in Sicht, zumindest für alle Besitzer eines CPC-6128: "SYNOPSIS 6128" heißt das Wunderprogramm und stellt einen vergleichenden Disassembler dar.

Die zugrundeliegende Idee in Kürze: Sie kopieren sich die Inhalte der Betriebssystem- und Basic-ROMs des CPC-464 und CPC-664 auf Diskette (beachten Sie dabei bitte die Urheberrechtsbestimmungen!). Diese vier mal 16 Kilobyte ergeben genau 64K und passen damit exakt in die zweite Speicherbank des CPC-6128.

Das Programm SYNOPSIS 6128 kann dann sowohl das 6128-ROM als auch die von der Diskette geladenen Dateien disassemblieren. Es zeigt in drei Bildschirmfenstern die von Ihnen gewählten Sektionen der ROMs auf dem Bildschirm an. Diese können Sie entweder durchlaufen lassen oder im Einzelschritt-Modus in Ruhe anschauen. Wenn Sie die gesuchte Routine oder Adresse gefunden haben, brechen Sie die Disassemblierung ab und schauen sich in den anderen Bildschirmfenstern die entsprechenden Routinen der anderen Computer an.

Auch diese können Sie stoppen und wieder beim ersten Bildschirmfenster weitermachen. Damit das Ganze für Sie komfortabel wird, arbeitet SYNOPSIS mit einem Übersichtlich aufgebauten Schirmbild:

CPC-464	CPC-664	CPC-6128	SYSTEM	BASIC	DISC
-----					
----	CPC-464	----	CPC-664	----	CPC-6128
-----					
&0000 LD	BC,&7F89	&C006 JP	&C1BC	&CDE4 CALL	&CD73
&0003 OUT	(C),C	&C009 JP	&C1B2	&CDE7 CALL	&CDC2
&0005 JP	&0580	&C00C JP	&CCD1	&CDEA CALL	&CDC7
&0008 JP	&B982	&C00F JP	&CCD5	&CDED DEC	B
&000B JP	&B97C			&CDEE JP	NZ,&CDAF
&000E PUSH	BC			&CDF1 LD	A,(HL)
				&CDF2 CALL	&CAA6
-----					
Start: &0000		Start: &C006		Start: &CDE4	
-----					

Natürlich ist es schwierig, mit der "Matrixdrucker-Pseudografik" das Aussehen des Bildschirms wiederzugeben, aber zumindest die Grundzüge dürften erkennbar sein.

Bevor Sie jedoch in den Genuß von SYNOPSIS 6128 kommen, steht vor Ihnen die Mühe des Programmeintippens. Und, zugegeben, es ist nicht gerade wenig. Beginnen wir mit dem Programm, das zwei Code-dateien erzeugt. Suchen Sie sich am besten eine möglichst leere Diskette heraus und tippen Sie das folgende Listing ab. Die REM-Zeilen können Sie sich schenken, das Programm springt keine von ihnen an. Genauer gesagt: Im ganzen Programm ist ein einziger GOTO-Befehl, und der springt in seine eigene Zeile. Also können Sie auch Zeilenblöcke zu einer Zeile zusammenfassen, wenn Sie sich Tipparbeit sparen wollen. Achten Sie aber zumindest auf die Numerierung der DATA-Zeilen, weil einige RESTORE-Befehle darauf Bezug nehmen:

#### 27.1 Listing: CODEGEN.BAS

```

1000 ' *****
1010 ' *
1020 ' * CODEGEN.BAS erzeugt Codedateien fuer SYNOPSIS 6128 *
1030 ' *
1040 ' *****
1050 '
1060 MODE 1:INK 3,24,1
1070 PEN 3:PRINT:PRINT " BITTE WARTEN SIE EINEN AUGENBLICK!":PRINT:PEN 1
1080 MEMORY &7FFF
1090 addr=&8000
1100 FOR i=0 TO 255
1110 READ a$
1120 FOR j=1 TO 13
1130 code=ASC(MID$(a$,j,1))
1140 IF code=ASC(".") THEN POKE addr,32 ELSE POKE addr,code
1150 addr=addr+1
1160 NEXT j

```

```

1170 NEXT i
1180 '-----
1190 DATA "1NOP.....", "2LD...BC, ^...", "1LD...(BC), A.", "1INC..BC....."
1200 DATA "1INC..B.....", "1DEC..B.....", "3LD...B, ^...", "1RLCA....."
1210 DATA "1EX...AF, AF'", "1ADD..HL, BC..", "1LD...A, (BC)..", "1DEC..BC....."
1220 DATA "1INC..C.....", "1DEC..C.....", "3LD...C, ^...", "1RRCA....."
1230 DATA "4DJNZ.^.....", "2LD...DE, ^...", "1LD...(DE), A.", "1INC..DE....."
1240 DATA "1INC..D.....", "1DEC..D.....", "3LD...D, ^...", "1RLA....."
1250 DATA "4JR...^.....", "1ADD..HL, DE..", "1LD...A, (DE)..", "1DEC..DE....."
1260 DATA "1INC..E.....", "1DEC..E.....", "3LD...E, ^...", "1RRA....."
1270 DATA "4JR...NZ, ^...", "2LD...HL, ^...", "2LD...( ^ ), HL.", "1INC..HL....."
1280 DATA "1INC..H.....", "1DEC..H.....", "3LD...H, ^...", "1DAA....."
1290 '-----
1300 DATA "4JR...Z, ^...", "1ADD..HL, HL..", "2LD...HL, ( ^ )..", "1DEC..HL....."
1310 DATA "1INC..L.....", "1DEC..L.....", "3LD...L, ^...", "1CPL....."
1320 DATA "4JR...NC, ^...", "2LD...SP, ^...", "2LD...( ^ ), A..", "1INC..SP....."
1330 DATA "1INC..(HL)....", "1DEC..(HL)....", "3LD...(HL), ^...", "1SCF....."
1340 DATA "4JR...C, ^...", "1ADD..HL, SP..", "2LD...A, ( ^ )..", "1DEC..SP....."
1350 DATA "1INC..A.....", "1DEC..A.....", "3LD...A, ^...", "1CCF....."
1360 DATA "1LD...B, B....", "1LD...B, C....", "1LD...B, D....", "1LD...B, E...."
1370 DATA "1LD...B, H....", "1LD...B, L....", "1LD...B, (HL)..", "1LD...B, A...."
1380 DATA "1LD...C, B....", "1LD...C, C....", "1LD...C, D....", "1LD...C, E...."
1390 DATA "1LD...C, H....", "1LD...C, L....", "1LD...C, (HL)..", "1LD...C, A...."
1400 '-----
1410 DATA "1LD...D, B....", "1LD...D, C....", "1LD...D, D....", "1LD...D, E...."
1420 DATA "1LD...D, H....", "1LD...D, L....", "1LD...D, (HL)..", "1LD...D, A...."
1430 DATA "1LD...E, B....", "1LD...E, C....", "1LD...E, D....", "1LD...E, E...."
1440 DATA "1LD...E, H....", "1LD...E, L....", "1LD...E, (HL)..", "1LD...E, A...."
1450 DATA "1LD...H, B....", "1LD...H, C....", "1LD...H, D....", "1LD...H, E...."
1460 DATA "1LD...H, H....", "1LD...H, L....", "1LD...H, (HL)..", "1LD...H, A...."
1470 DATA "1LD...L, B....", "1LD...L, C....", "1LD...L, D....", "1LD...L, E...."
1480 DATA "1LD...L, H....", "1LD...L, L....", "1LD...L, (HL)..", "1LD...L, A...."
1490 DATA "1LD...(HL), B..", "1LD...(HL), C..", "1LD...(HL), D..", "1LD...(HL), E.."
1500 DATA "1LD...(HL), H..", "1LD...(HL), L..", "1HALT.....", "1LD...(HL), A.."
1510 '-----
1520 DATA "1LD...A, B....", "1LD...A, C....", "1LD...A, D....", "1LD...A, E...."
1530 DATA "1LD...A, H....", "1LD...A, L....", "1LD...A, (HL)..", "1LD...A, A...."
1540 DATA "1ADD..A, B....", "1ADD..A, C....", "1ADD..A, D....", "1ADD..A, E...."
1550 DATA "1ADD..A, H....", "1ADD..A, L....", "1ADD..A, (HL)..", "1ADD..A, A...."
1560 DATA "1ADC..A, B....", "1ADC..A, C....", "1ADC..A, D....", "1ADC..A, E...."
1570 DATA "1ADC..A, H....", "1ADC..A, L....", "1ADC..A, (HL)..", "1ADC..A, A...."
1580 DATA "1SUB..B.....", "1SUB..C.....", "1SUB..D.....", "1SUB..E....."
1590 DATA "1SUB..H.....", "1SUB..L.....", "1SUB...(HL)....", "1SUB..A....."
1600 DATA "1SBC..A, B....", "1SBC..A, C....", "1SBC..A, D....", "1SBC..A, E...."
1610 DATA "1SBC..A, H....", "1SBC..A, L....", "1SBC..A, (HL)..", "1SBC..A, A...."
1620 '-----
1630 DATA "1AND..B.....", "1AND..C.....", "1AND..D.....", "1AND..E....."
1640 DATA "1AND..H.....", "1AND..L.....", "1AND...(HL)....", "1AND..A....."
1650 DATA "1XOR..B.....", "1XOR..C.....", "1XOR..D.....", "1XOR..E....."
1660 DATA "1XOR..H.....", "1XOR..L.....", "1XOR...(HL)....", "1XOR..A....."
1670 DATA "1OR...B.....", "1OR...C.....", "1OR...D.....", "1OR...E....."
1680 DATA "1OR...H.....", "1OR...L.....", "1OR...(HL)....", "1OR...A....."
1690 DATA "1CP...B.....", "1CP...C.....", "1CP...D.....", "1CP...E....."
1700 DATA "1CP...H.....", "1CP...L.....", "1CP...(HL)....", "1CP...A....."
1710 DATA "1RET..NZ.....", "1POP..BC.....", "2JP...NZ, ^...", "2JP...^....."
1720 DATA "2CALL.NZ, ^...", "1PUSH.BC.....", "3ADD..A, ^...", "1RST.&00...."
1730 '-----

```

```

1740 DATA "1RET..Z.....", "1RET.....", "2JP...Z,^....", "5....."
1750 DATA "2CALL.Z,^....", "2CALL.^.....", "3ADC..A,^....", "1RST..&08...."
1760 DATA "1RET..NC.....", "1POP..DE.....", "2JP...NC,^....", "3OUT..(^),A.."
1770 DATA "2CALL.NC,^....", "1PUSH.DE.....", "3SUB..^.....", "1RST..&10...."
1780 DATA "1RET..C.....", "1EXX.....", "2JP...C,^....", "3IN...A,(^).."
1790 DATA "2CALL.C,^....", "6.....", "3SBC..A,^....", "1RST..&18...."
1800 DATA "1RET..PO.....", "1POP..HL.....", "2JP...PO,^....", "1EX... (SP),HL"
1810 DATA "2CALL.PO,^....", "1PUSH.HL.....", "3AND..^.....", "1RST..&20...."
1820 DATA "1RET..PE.....", "1JP...(HL)....", "2JP...PE,^....", "1EX...DE,HL.."
1830 DATA "2CALL.PE,^....", "7.....", "3XOR..^.....", "1RST..&28...."
1840 ' -----
1850 DATA "1RET..P.....", "1POP..AF.....", "2JP...P,^....", "1DI....."
1860 DATA "2CALL.P,^....", "1PUSH.AF.....", "3OR...^.....", "1RST..&30...."
1870 DATA "1RET..M.....", "1LD...SP,HL...", "2JP...M,^....", "1EI....."
1880 DATA "2CALL.M,^....", "8.....", "3CP...^.....", "1RST..&38...."
1890 ' Befehle mit Vorsatzbyte &CB -----
1900 addr=&8D00
1910 RESTORE 1930
1920 FOR i=1 TO 8:READ reg$(i):NEXT i
1930 DATA B,C,D,E,H,L,M,A
1940 FOR i=0 TO 255 STEP 8
1950   READ a$
1960   FOR j=1 TO 8
1970     code$=a$+reg$(j)
1980     IF LEN(code$)<8 THEN code$=code$+" ":GOTO 1980
1990     FOR k=1 TO LEN(code$)
2000       code=ASC(MID$(code$,k,1))
2010       IF code=ASC(" ") THEN POKE addr,32 ELSE POKE addr,code
2020       addr=addr+1
2030     NEXT k
2040   NEXT j
2050 NEXT i
2060 ' -----
2070 DATA "RLC..", "RRC..", "RL...", "RR..."
2080 DATA "SLA...", "SRA...", "***...", "SRL..."
2090 DATA "BIT..0,", "BIT..1,", "BIT..2,", "BIT..3,", "BIT..4,"
2100 DATA "BIT..5,", "BIT..6,", "BIT..7,", "RES..0,", "RES..1,"
2110 DATA "RES..2,", "RES..3,", "RES..4,", "RES..5,", "RES..6,"
2120 DATA "RES..7,", "SET..0,", "SET..1,", "SET..2,", "SET..3,"
2130 DATA "SET..4,", "SET..5,", "SET..6,", "SET..7,"
2140 ' Befehle mit Vorsatzbyte &ED -----
2150 addr=&9500
2160 DIM codebyte(56)
2170 FOR i=1 TO 56
2180   READ codebyte(i)
2190 NEXT i
2200 FOR i=1 TO 56
2210   READ a$
2220   POKE addr,codebyte(i)
2230   addr=addr+1
2240   FOR j=1 TO LEN(a$)
2250     code=ASC(MID$(a$,j,1))
2260     IF code=ASC(" ") THEN POKE addr,32 ELSE POKE addr,code
2270     addr=addr+1
2280   NEXT j
2290 NEXT i
2300 ' -----

```

```

2310 DATA &40,&41,&42,&43,&44,&45,&46,&47,&48,&49,&4A,&4B,&4D,&4F,&50,&51
2320 DATA &52,&53,&56,&57,&58,&59,&5A,&5B,&5E,&5F,&60,&61,&62,&67,&68,&69
2330 DATA &6A,&6F,&72,&73,&78,&79,&7A,&7B,&A0,&A1,&A2,&A3,&A8,&A9,&AA,&AB
2340 DATA &B0,&B1,&B2,&B3,&B8,&B9,&BA,&BB
2350 ' -----
2360 DATA "1IN...B,(C)...", "1OUT...(C),B...", "1SBC...HL,BC...", "2LD...(^),BC."
2370 DATA "1NEG.....", "1RETN.....", "1IM...0.....", "1LD...I,A..."
2380 DATA "1IN...C,(C)...", "1OUT...(C),C...", "1ADC...HL,BC...", "2LD...BC, (^)."
2390 DATA "1RETI.....", "1LD...R,A...", "1IN...D,(C)...", "1OUT...(C),D..."
2400 DATA "1SBC...HL,DE...", "2LD...(^),DE...", "1IM...1.....", "1LD...A,I..."
2410 DATA "1IN...E,(C)...", "1OUT...(C),E...", "1ADC...HL,DE...", "2LD...DE, (^)."
2420 DATA "1IM...2.....", "1LD...A,R...", "1IN...H,(C)...", "1OUT...(C),H..."
2430 DATA "1SBC...HL,HL...", "1RRD.....", "1IN...L,(C)...", "1OUT...(C),L..."
2440 DATA "1ADC...HL,HL...", "1RLD.....", "1SBC...HL,SP...", "2LD...(^),SP..."
2450 DATA "1IN...A,(C)...", "1OUT...(C),A...", "1ADC...HL,SP...", "2LD...SP, (^)."
2460 DATA "1LDI.....", "1CPI.....", "1INI.....", "1OUTI....."
2470 DATA "1LDD.....", "1CPD.....", "1IND.....", "1OUTD....."
2480 DATA "1LDIR.....", "1CPIR.....", "1INIR.....", "1OTIR....."
2490 DATA "1LDDR.....", "1CPDR.....", "1INDR.....", "1OTDR....."
2500 ' Befehle mit Vorsatzbyte &DD und &FD -----
2510 addr=&9820
2520 FOR i=1 TO 38
2530   READ a,a$
2540   a$=CHR$(a)+a$
2550   FOR j=1 TO LEN(a$)
2560     code=ASC(MID$(a$,j,1))
2570     IF code=ASC(".") THEN POKE addr,32 ELSE POKE addr,code
2580     addr=addr+1
2590   NEXT j
2600 NEXT i
2610 ' -----
2620 DATA &09,"1ADD...IX,BC...", &19,"1ADD...IX,DE...", &23,"1INC...IX,...."
2630 DATA &29,"1ADD...IX,IX...", &2B,"1DEC...IX,....", &39,"1ADD...IX,SP..."
2640 DATA &E1,"1POP...IX,....", &E3,"1EX...(SP),IX.", &E5,"1PUSH...IX,...."
2650 DATA &E9,"1JP...(IX),....", &F9,"1LD...SP,IX..."
2660 ' -----
2670 DATA &21,"2LD...IX, ^...", &22,"2LD...(^),IX...", &2A,"2LD...IX, (^)..."
2680 DATA &34,"3INC...(IX+^)...", &35,"3DEC...(IX+^)...", &46,"3LD...B, (IX+^)"
2690 DATA &4E,"3LD...C, (IX+^)" , &56,"3LD...D, (IX+^)" , &5E,"3LD...E, (IX+^)"
2700 DATA &66,"3LD...H, (IX+^)" , &6E,"3LD...L, (IX+^)" , &70,"3LD...(IX+^),B"
2710 DATA &71,"3LD...(IX+^),C" , &72,"3LD...(IX+^),D" , &73,"3LD...(IX+^),E"
2720 DATA &74,"3LD...(IX+^),H" , &75,"3LD...(IX+^),L" , &77,"3LD...(IX+^),A"
2730 DATA &7E,"3LD...A, (IX+^)" , &86,"3ADD...A, (IX+^)" , &8E,"3ADC...A, (IX+^)"
2740 DATA &96,"3SUB...(IX+^)...", &9E,"3SBC...A, (IX+^)" , &A6,"3AND...(IX+^)..."
2750 DATA &AE,"3XOR...(IX+^)...", &B6,"3OR... (IX+^)...", &BE,"3CP...(IX+^)..."
2760 ' -----
2770 addr=&9A60:RESTORE 2800
2780 FOR i=1 TO 7
2790   READ a,a$
2800   a$=CHR$(a)+MID$(a$,2)
2810   FOR j=1 TO LEN(a$)
2820     code=ASC(MID$(a$,j,1))
2830     IF code=ASC(".") THEN POKE addr,32 ELSE POKE addr,code
2840     addr=addr+1
2850   NEXT j
2860 NEXT i
2870 ' -----

```

```

2880 DATA &06,"3RLC..(IX+^)..", &0E,"3RRC..(IX+^)..", &16,"3RL..(IX+^)..",
2890 DATA &1E,"3RR..(IX+^)..", &26,"3SLA..(IX+^)..", &2E,"3SRA..(IX+^)..",
2900 DATA &3E,"3SRL..(IX+^)..",
2910 ' -----
2920 FOR i=1 TO 24
2930   READ a,a#
2940   a#=CHR$(a)+MID$(a$,2)
2950   FOR j=1 TO LEN(a#)
2960     code=ASC(MID$(a#,j))
2970     IF code=ASC(".") THEN POKE addr,32 ELSE POKE addr,code
2980     addr=addr+1
2990   NEXT j
3000 NEXT i
3010 ' -----
3020 DATA &46,"3BIT..0,(IX+^)", &4E,"3BIT..1,(IX+^)", &56,"3BIT..2,(IX+^)",
3030 DATA &5E,"3BIT..3,(IX+^)", &66,"3BIT..4,(IX+^)", &6E,"3BIT..5,(IX+^)",
3040 DATA &76,"3BIT..6,(IX+^)", &7E,"3BIT..7,(IX+^)", &86,"3RES..0,(IX+^)",
3050 DATA &8E,"3RES..1,(IX+^)", &96,"3RES..2,(IX+^)", &9E,"3RES..3,(IX+^)",
3060 DATA &AE,"3RES..4,(IX+^)", &AE,"3RES..5,(IX+^)", &BE,"3RES..6,(IX+^)",
3070 DATA &BE,"3RES..7,(IX+^)", &CE,"3SET..0,(IX+^)", &CE,"3SET..1,(IX+^)",
3080 DATA &DE,"3SET..2,(IX+^)", &DE,"3SET..3,(IX+^)", &EE,"3SET..4,(IX+^)",
3090 DATA &EE,"3SET..5,(IX+^)", &FE,"3SET..6,(IX+^)", &FE,"3SET..7,(IX+^)",
3100 ' Binaerdatei abspeichern -----
3110 SAVE "DISCODE.BIN",b,&8000,&1D00
3120 ' Binaerdatei mit Hilfsroutinen erzeugen -----
3130 FOR i=40960 TO 41104
3140   READ a
3150   POKE i,a
3160 NEXT i
3170 SAVE "SYNAID.BIN",b,&A000,&90
3180 ' -----
3190 DATA &C3,&3F,&A0,&C3,&57,&A0,&C3,&73,&A0,&0B,&44,&49,&53,&43,&4F,&44
3200 DATA &45,&2E,&42,&49,&4E,&09,&42,&41,&53,&49,&43,&2E,&34,&36,&34,&0A
3210 DATA &53,&59,&53,&54,&45,&4D,&2E,&34,&36,&34,&09,&42,&41,&53,&49,&43
3220 DATA &2E,&36,&36,&34,&0A,&53,&59,&53,&54,&45,&4D,&2E,&36,&36,&34,&CD
3230 DATA &00,&B9,&CD,&06,&B9,&DD,&66,&03,&DD,&6E,&02,&7E,&DD,&66,&01,&DD
3240 DATA &6E,&00,&77,&23,&36,&00,&C9,&0E,&07,&CD,&0F,&B9,&C5,&DD,&66,&03
3250 DATA &DD,&6E,&02,&7E,&DD,&66,&01,&DD,&6E,&00,&77,&23,&36,&00,&C1,&CD
3260 DATA &18,&B9,&C9,&3A,&62,&01,&47,&2A,&64,&01,&11,&F0,&37,&CD,&77,&BC
3270 DATA &2A,&66,&01,&CD,&83,&BC,&CD,&7A,&BC,&C9,&00,&00,&00,&00,&00,&00
3280 DATA &00,&00
3290 ' -----

```

Geschafft? (im doppelten Sinn). Nun zumindest sollten Ihnen die Punkte in den DATA-Zeilen beim Abzählen der Positionen geholfen haben. Sie haben Sie doch hoffentlich wirklich mit abgetippt? Wenn Sie stattdessen Leerzeichen eingegeben haben, ist das allerdings auch nicht so schlimm.

Starten Sie das Programm mit RUN. Der Computer arbeitet jetzt für einige Minuten.

Dann läuft das Diskettenlaufwerk an, und der CATalog-Befehl beweist, daß auf der Diskette zwei neue Dateien stehen: DISCODE.BIN und SYNAID.BIN.



DISCODE.BIN enthält alle Z80-Maschinencodes in disassemblierter Form. Durch diese Art der Speicherung kann der Disassembler selbst relativ kurz gehalten werden und erreicht eine für ein Basic-Programm sehr hohe Geschwindigkeit.

SYNAID.BIN ist eine Datei, die dem SYNOPSIS-Programm Hilfsroutinen in Maschinensprache zur Verfügung stellt. Es handelt sich um CALL-Aufrufe zum Lesen der ROMs und zum Laden der Binärdateien. Falls Sie neugierig sind, können Sie ja den folgenden Quellcode analysieren.

## 27.2 Assemblerlisting: Hilfsroutinen für SYNOPSIS 6128

```
; *****
; *
; *           Hilfsroutinen für SYNOPSIS 6128           *
; *
; *****
;
;           ORG           &A000
;
;           JP           ROM_PEEK
;           JP           FLOPPY_PEEK
;           JP           BIN_LOAD
;
; ***** DATEINAMEN *****
;
LEN_1      DEFB          11
FILE_1     DEFM          "DISCODE.BIN"

LEN_2      DEFB          9
FILE_2     DEFM          "BASIC.464"

LEN_3      DEFB          10
FILE_3     DEFM          "SYSTEM.464"

LEN_4      DEFB          9
FILE_4     DEFM          "BASIC.664"

LEN_5      DEFB          10
FILE_5     DEFM          "SYSTEM.664"
; ***** DEFINITIONEN *****
;
KL_UROM_ENABLE EQU      &B900      ; KL U ROM ENABLE
KL_LROM_ENABLE EQU      &B906      ; KL L ROM ENABLE
KL_ROM_SELECT EQU      &B90F      ; KL ROM SELECT
KL_ROM_DESELECT EQU     &B918      ; KL ROM DESELECT
CAS_IN_OPEN EQU      &BC77      ; CAS IN OPEN
CAS_IN_DIRECT EQU     &BC83      ; CAS IN DIRECT
CAS_IN_CLOSE EQU     &BC7A      ; CAS IN CLOSE
NAME_LENGTH EQU      &0162      ; Speicher für File-Namenslänge
NAME_ADDRESS EQU     &0164      ; Adresse des Filenamens
LOAD_ADDRESS EQU     &0166      ; Ladeadresse der Binärdatei
```

```

;
; ***** BASIC- UND SYSTEM-PEEK *****
;

ROM_PEEK      CALL    KL_UROM_ENABLE    ; Basic-ROM einschalten
              CALL    KL_LROM_ENABLE    ; System-ROM einblenden
              LD      H,(IX+3)          ; Adresse - Highbyte lesen
              LD      L,(IX+2)          ; Adresse - Lowbyte lesen
              LD      A,(HL)            ; Byte an der Adresse lesen
              LD      H,(IX+1)          ; Variablenadresse - Highbyte
              LD      L,(IX+0)          ; Variablenadresse - Lowbyte
              LD      (HL),A            ; Byte in die Variable schreiben
              INC      HL                ; Zeiger auf Variablen-Highbyte
              LD      (HL),0            ; Highbyte auf 0 setzen
              RET                       ; Rücksprung nach Basic

;
; ***** FLOPPY-PEEK *****
;

FLOPPY_PEEK   LD      C,7                ; Select-Nummer des Floppy-ROMs
              CALL    KL_ROM_SELECT      ; ROM auswählen
              PUSH    BC                 ; ROM-Status sichern
              LD      H,(IX+3)          ; Adresse - Highbyte lesen
              LD      L,(IX+2)          ; Adresse - Lowbyte lesen
              LD      A,(HL)            ; Byte an der Adresse lesen
              LD      H,(IX+1)          ; Variablenadresse - Highbyte
              LD      L,(IX+0)          ; Variablenadresse - Lowbyte
              LD      (HL),A            ; Byte in die Variable schreiben
              INC      HL                ; Zeiger auf Variablen-Highbyte
              LD      (HL),0            ; Highbyte auf 0 setzen
              POP     BC                 ; ROM-Status zurückholen
              CALL    KL_ROM_DESELECT    ; ROM ausblenden
              RET                       ; Rücksprung nach Basic

;
; ***** LADEN VON BINKRDATFIEN *****
;

BIN_LOAD      LD      A,(NAME_LENGTH)    ; Länge des Dateinamens lesen
              LD      B,A                ; ins B-Register
              LD      HL,(NAME_ADDRESS)  ; Adresse des Dateinamens holen
              LD      DE,14320           ; 2K-Dateipuffer
              CALL    CAS_IN_OPEN        ; Datei zum Lesen öffnen
              LD      HL,(LOAD_ADDRESS)  ; Ladeadresse holen
              CALL    CAS_IN_DIRECT      ; Datei einlesen
              CALL    CAS_IN_CLOSE       ; Datei schließen
              RET                       ; Rücksprung nach Basic
              END

; -----

```

Die Programmroutine ROM\_PEEK liest jeweils ein einzelnes Byte aus dem Basic- oder Betriebssystem-ROM. Der Aufruf erfolgt als a%=0:

```
CALL &A000,adresse,0a%.
```

Die Funktionsweise von ROM\_PEEK: Zuerst schaltet die Routine mit KL U ROM ENABLE (Adresse &B900) und KL L ROM ENABLE (Adresse &B906) die beiden ROMs der Hauptplatine ein und holt sich aus dem CALL-Aufruf aus Basic die Adresse, die ausgelesen werden soll: LD H, (IX+3) und LD L, (IX+2). Mit LD A, (HL) überträgt der Computer den Inhalt der Speicherstelle in den Akku.

Die beiden folgenden Ladebefehle setzen die Adresse der Integer-Variablen aus dem Aufruf ins HL-Register ein. LD (HL), A setzt das Lowbyte der Variablen auf den gelesenen Wert. INC HL und LD (HL), 0 löschen das Highbyte in der Variablen. Und RET besorgt selbstverständlich den Rücksprung ins Basic.

FLOPPY\_PEEK ist identisch aufgebaut, nur mit dem Unterschied, daß ein anderes ROM eingeblendet werden muß. Das AMSDOS-Floppy-ROM besitzt die Nummer 7, also schalten LD A, 7 und CALL KL ROM SELECT (Adresse &B90F) das entsprechende ROM in den Adreßbereich zwischen &C000 und &FFFF. Diese Systemroutine übergibt im BC-Register den vorherigen ROM-Status, der dann auch mit PUSH BC auf dem Stack gesichert wird. Nach Abarbeitung des Programms kann er mit POP BC zurückgeholt und mit KL ROM DESELECT (&B918) wieder hergestellt werden.

Das BIN\_LOAD-Programm lädt Binärdateien von der Diskette in den RAM-Speicher. Diese Routine ist notwendig geworden, weil die vorhandenen Unterprogramme im Basic-ROM bei sämtlichen Dateioperationen sowohl einen Eingabepuffer als auch einen Ausgabepuffer belegen. Das kostet 4096 Bytes, auch wenn nur Dateien eingelesen werden sollen. Die 2048 Bytes des Ausgabepuffers wären verschwendet. Im Gegensatz dazu erfordern die Routinen des System-ROMs wirklich nur die 2048 Bytes, die man braucht. Der Dateipuffer wird zwei Kilobytes unterhalb des neuen HIMEMs bei 16383 (&3FFF) positioniert.

Die Startadresse 14320 enthält noch eine Sicherheitsmarge von 15 Bytes für alle Fälle.

Der Aufruf von BIN\_LOAD ist nicht ganz so einfach wie der der bisherigen Routinen. Denn es müssen vorher noch im RAM einige Zeiger gesetzt werden:

- Die Adresse mit dem Label NAME\_LENGTH (im Speicher an der Adresse &0162) wird mit der Länge des Dateinamens geladen, zum Beispiel POKE &0162, 10.
- Die symbolische Adresse NAME\_ADDRESS (Adresse &0164) zeigt auf den Dateinamen im Speicher, zum Beispiel POKE &0164, &00:POKE &0165, &80. Diese POKEs weisen auf die Adresse &8000. Dort muß dann der Dateiname im Klartext stehen, etwa "BASIC.464".
- Das Label LOAD\_ADDRESS (Adresse &0166) enthält die Startadresse, an der die Binärdatei im Speicher abgelegt werden soll. POKE &0166, &00:POKE &0167, &40 legt zum Beispiel die Datei bei &4000 (16384) ab.

Sie brauchen nicht alles, was hier dargestellt ist, notwendigerweise verstehen oder sich gar merken, da Sie diese Informationen

keinesfalls zur Benutzung von SYNOPLIS 6128 benötigen. Sie sind nur für die "Programmier-Freaks" gedacht, die unbedingt wissen wollen, wie jedes Detail des Programms funktioniert.

Als nächstes müssen Sie die Betriebssystem- und Basic-ROMs der anderen Schneider-Computer (CPC-464 und CPC-664) auf die Diskette bekommen. Dazu dient das Maschinenprogramm COPYROM.ASM. Hier ist es erst einmal in Quellcode-Form. Diese Fassung tippen Sie aber bitte NICHT ab, sie dient nur zu Ihrer Information!

### 27.3 Assemblerlisting: COPYROM.ASM

```
; *****
; *
; *   COPYROM.ASM kopiert Betriebssystem und Basic-ROM auf die Diskette   *
; *                                                                 *
; *****

                ORG        &A000
                JP         SYSTEM_ROM

;
; ***** Equates *****
;

TXT_OUTPUT      EQU        &BB5A
KL_URDM_ENABLE  EQU        &B900
KL_LROM_ENABLE  EQU        &B906

CAS_OUT_OPEN    EQU        &BC8C
CAS_OUT_DIRECT  EQU        &BC98
CAS_OUT_CLOSE   EQU        &BC8F

;
; ***** Dateinamen *****
;

FILE_1          DEFM       "SYSTEM.ROM"
FILE_2          DEFM       "BASIC.ROM"

;
; ***** Bildschirmmeldungen *****
;

TEXT1           DEFM       "SAVE "
                DEFB       34
                DEFM       "SYSTEM.ROM"
                DEFB       34
                DEFM       ",B,16384,16384"
                DEFB       13,10,10

TEXT2           DEFM       "SAVE "
                DEFB       34
                DEFM       "BASIC.ROM"
                DEFB       34
                DEFM       ",B,16384,16384"
```

```

DEFB      13,10,10

;
; ***** Abspeichern des Betriebssystem-ROMs *****
;

SYSTEM_ROM    CALL    KL_LROM_ENABLE
               LD      HL,&0000
               LD      DE,&4000
               LD      BC,&4000
               LDIR

MSG1           LD      B,34
               LD      HL,TEXT1
               CALL    MESSAGE

OPENOUT_1ST   LD      B,10
               LD      HL,FILE_1
               LD      DE,&8000
               CALL    CAS_OUT_OPEN

SAVE_1ST      LD      A,2
               LD      HL,&4000
               LD      DE,&4000
               LD      BC,&0000
               CALL    CAS_OUT_DIRECT

CLOSEOUT_1ST  CALL    CAS_OUT_CLOSE

;
; ***** Abspeichern des Basic-ROMs *****
;

BASIC         CALL    KL_UROM_ENABLE
               LD      HL,&C000
               LD      DE,&4000
               LD      BC,&4000
               LDIR

MSG2           LD      B,33
               LD      HL,TEXT2
               CALL    MESSAGE

OPENOUT_2ND   LD      B,9
               LD      HL,FILE_2
               LD      DE,&8000
               CALL    CAS_OUT_OPEN

SAVE_2ND      LD      A,2
               LD      HL,&4000
               LD      DE,&4000
               LD      BC,&0000
               CALL    CAS_OUT_DIRECT

CLOSEOUT_2ND  CALL    CAS_OUT_CLOSE
               RET

```

```

;
; ***** Unterprogramm MESSAGE druckt den String (HL) aus *****
;

```

```

MESSAGE      LD      A,(HL)
              CALL    TXT_OUTPUT
              INC      HL
              DJNZ     MESSAGE
              RET

              END

```

```

; -----

```

Das Programm COPYROM.ASM startet bei der Marke SYSTEM.ROM. Dort kopiert es mit LDIR das Betriebssystem aus dem Bereich von &0000 bis &3FFF in den RAM-Speicher zwischen &4000 und &7FFF. Vorher muß mit KL L ROM ENABLE (Adresse &B906) der ROM-Baustein in den Speicher eingeblendet werden. Das Umkopieren in den frei lesbaren Speicher ist notwendig, weil sämtliche Disketten-Routinen nicht aus den ROMs lesen können, sondern immer auf die darunterliegenden RAM-Bereiche zugreifen.

MSG1 gibt die Meldung SAVE "SYSTEM.ROM",B,16384,16384 aus, damit Sie sehen können, wie der Computer bei seiner Arbeit voranschreitet.

OPENOUT\_1ST öffnet eine Diskettendatei zur Ausgabe von Daten. SAVE\_1ST überträgt die 16 Kilobytes aus dem Speicher in die Datei. Dazu wird CAS OUT DIRECT verwendet. CLOSE\_1ST schließt die Datei wieder.

Ganz ähnlich arbeitet das Programm beim Speichern des Basic-ROMs. Hier wird zuerst mit KL U ROM ENABLE (Adresse &B900) das Basic-ROM eingeschaltet. LDIR überträgt den ROM-Inhalt von &C000 bis &FFFF in den ungebankten Speicher von &4000 bis &7FFF. MSG2 sorgt für die Bildschirmmeldung SAVE "BASIC.ROM",B,16384,16384. OPENOUT\_2ND öffnet die Datei auf der Diskette, in die die Daten geschrieben werden. SAVE\_2ND führt die Übertragung der Daten mit CALL CAS OUT DIRECT durch, und CLOSEOUT\_2ND schließt die Datei wieder.

Der im folgenden abgedruckte Basic-Loader enthält das Maschinenprogramm in DATA-Zeilen, verfügt aber zusätzlich auch über eine gewisse "Eigenintelligenz". So erkennt er selbsttätig, ob er auf einem CPC-464 oder einem CPC-664 gestartet wird und legt automatisch die richtigen zwei Dateien an.

Auf dem CPC-464 speichert er die beiden Dateien "SYSTEM.464" und "BASIC.464", auf dem CPC-664 erzeugt er die Files "SYSTEM.664" und "BASIC.664".

Ein Programmstart auf dem CPC-6128 hat eine Fehlermeldung zur Folge: "Nicht auf diesem Computer starten!"

Ein Start auf dieser Maschine wäre auch völlig sinnlos, da beim CPC-6128 ja schon die entsprechenden ROMs eingebaut sind. Eine

zusätzliche Speicherung der ROM-Inhalte auf der Diskette ergäbe wenig Sinn und würde nur den wertvollen Disketten-Speicherplatz vermindern. SYNOPSIS 6128 wird nämlich so arbeiten, daß es die System- und Basic-ROMs des CPC-464 und CPC-664 in die Speichererweiterung lädt. Über ein recht trickreiches Unterprogramm PEEKt es immer aus dem passenden Speicher das gewünschte Byte.

Ein paar Beispiele dafür:

- Adresse &C03D beim CPC-464 im Basic-ROM lesen:  
Speichererweiterung.
- Adresse &0003 beim CPC-664 im System-ROM lesen:  
Speichererweiterung.
- Adresse &1FFF beim CPC-6128 im System-ROM lesen:  
System-ROM des CPC-6128 PEEKen.

Eine Sonderstellung besitzt das Disc-AMSDOS-ROM, das von SYNOPSIS 6128 ebenfalls analysiert werden kann.

Da bei den bisherigen Modellen von Schneider alle AMSDOS-ROMs absolut identisch sind, ist eine Speicherung des Disc-ROMs auf Diskette unnötig.

Das Programm gestattet dem Benutzer den Zugriff auf alle drei Disc-ROMs, intern wird aber immer aus dem eingebauten gelesen.

Nachfolgenden Basic-Loader sollten Sie abtippen!

#### 27.4 Listing: COPYROM.BAS

```

100 ' *****
110 ' *
120 ' *          COPYROM.BAS          *
130 ' *
140 ' *****
150 '
160 MEMORY &3FFF
170 FOR i=40960 TO 41159
180   READ a
190   sum=sum+a
200 NEXT i
210 IF sum=15136 THEN 230
220 PRINT "DATA-Fehler!":LIST 420-
230 RESTORE
240 FOR i=40960 TO 41159
250   READ a
260   POKE i,a
270 NEXT i
280 POKE &160,&CD:POKE &161,&0:POKE &162,&B9:POKE &163,&3A:POKE &164,&2
290 POKE &165,&C0:POKE &166,&32:POKE &167,&60:POKE &168,&1:POKE &169,&C9
300 CALL &160:cpcversion=PEEK(&160)
310 IF cpcversion=2 THEN PRINT "Nicht auf diesem Computer starten!":END
320 IF cpcversion=0 THEN a$="464" ELSE a$="664"
330 adr(0)=&A00A:adr(1)=&A013:adr(2)=&A023:adr(3)=&A044

```

```

340 FOR i=0 TO 3
350   x=adr(i)
360   FOR j=1 TO 3
370     POKE x+j-1,ASC(MID$(a$,j))
380   NEXT j
390 NEXT i
400 CALL &A000
410 ' Maschinencode in DATA-Zeilen *****
420 DATA &C3,&59,&A0,&53,&59,&53,&54,&45,&4D,&2E,&52,&4F,&4D,&42,&41,&53
430 DATA &49,&43,&2E,&52,&4F,&4D,&53,&41,&56,&45,&20,&22,&53,&59,&53,&54
440 DATA &45,&4D,&2E,&52,&4F,&4D,&22,&2C,&42,&2C,&31,&36,&33,&38,&34,&2C
450 DATA &31,&36,&33,&38,&34,&0D,&0A,&0A,&53,&41,&56,&45,&20,&22,&42,&41
460 DATA &53,&49,&43,&2E,&52,&4F,&4D,&22,&2C,&42,&2C,&31,&36,&33,&38,&34
470 DATA &2C,&31,&36,&33,&38,&34,&0D,&0A,&0A,&CD,&06,&B9,&21,&00,&00,&11
480 DATA &00,&40,&01,&00,&40,&ED,&B0,&06,&22,&21,&16,&A0,&CD,&BE,&A0,&06
490 DATA &0A,&21,&03,&A0,&11,&00,&80,&CD,&8C,&BC,&3E,&02,&21,&00,&40,&11
500 DATA &00,&40,&01,&00,&00,&CD,&98,&BC,&CD,&8F,&BC,&CD,&00,&B9,&21,&00
510 DATA &C0,&11,&00,&40,&01,&00,&40,&ED,&B0,&06,&21,&21,&3B,&A0,&CD,&BE
520 DATA &A0,&06,&09,&21,&0D,&A0,&11,&00,&80,&CD,&8C,&BC,&3E,&02,&21,&00
530 DATA &40,&11,&00,&40,&01,&00,&00,&CD,&98,&BC,&CD,&8F,&BC,&C9,&7E,&CD
540 DATA &5A,&BB,&23,&10,&F9,&C9,&00,&00
550 END ' *****

```

Starten Sie das Programm einmal auf dem CPC-464 und dann auf einem CPC-664. Es werden vier Dateien mit je 17 Kilobyte Länge angelegt. Geben Sie den Directory-Befehl ein, mußte er folgende Dateien zeigen:

- CODEGEN.BAS
- DISCODE.BIN
- SYNAID.BIN
- SYSTEM.464
- BASIC.464
- SYSTEM.664
- BASIC.664

Und nun der letzte große "Brocken", das Hauptprogramm SYNOPSIS. Wenn Sie auch diese abgetippt haben, ist SYNOPSIS 6128 voll einsatzfähig!

## 27.5 Listing: SYNOPSIS.BAS

```

1000 ' *****
1010 ' * *
1020 ' * SYNOPSIS 6128 - Vergleichender Disassembler *
1030 ' * *
1040 ' *****
1050 '
1060 ' Pruefung, ob SYNOPSIS 6128 auf einem CPC-6128 gestartet wird ----
1070 OUT &7F00,&C0:POKE &4000,&E9
1080 OUT &7F00,&C4:POKE &4000,&E5
1090 OUT &7F00,&C0:rdvalue=PEEK(&4000)
1100 IF rdvalue=&E9 THEN 1140
1110 PRINT "SYNOPSIS 6128 arbeitet mit diesem Computer nicht!"

```



```

1120 PRINT:PRINT:END
1130 ' Titelbild und Abfrage auf Farbmonitor -----
1140 CALL &BB4E ' TXT INITIALISE = Text-VDU initialisieren
1150 CALL &BBBA ' GRA INITIALISE = Grafik-VDU initialisieren
1160 CALL &BBFF ' SCR INITIALISE = Screen-Paket initialisieren
1170 IF PEEK(&160)=&E5 THEN 1550
1180 POKE &160,&E5
1190 MODE 1
1200 INK 1,24:INK 0,1:BORDER 1:PEN 1
1210 PRINT STRING$(40,210);
1220 PRINT CHR$(24);SPACE$(13);"SYNOPSIS 6128";SPACE$(14);CHR$(24)
1230 PRINT:PRINT STRING$(40,210):PRINT
1240 PRINT:PRINT " Dieses Programm analysiert die ROMs"
1250 PRINT:PRINT " des CPC-464, CPC-664 und CPC-6128."
1260 PRINT:PRINT
1270 PRINT STRING$(40,210)
1280 PRINT:PRINT " Besitzen Sie einen Farbmonitor? ";CHR$(143);
1290 a$=UPPER$(INKEY$):IF a$="" THEN 1290
1300 IF a$="N" THEN 1330
1310 IF a$<>"J" THEN PRINT CHR$(11);CHR$(7);:GOTO 1280
1320 INK 1,0:INK 0,13:BORDER 10
1330 MEMORY &9FFF
1340 LOAD "SYNAID.BIN",&A000
1350 MEMORY 14320
1360 '
1370 POKE &162,11:POKE &164,&A:POKE &165,&A0:POKE &166,0:POKE &167,&0
1380 CALL &A006 ' DISCODE.BIN
1390 '
1400 POKE &162,9:POKE &164,&16:POKE &167,&40
1410 OUT &7F00,&C4:CALL &A006 ' BASIC.464
1420 '
1430 POKE &162,10:POKE &164,&20
1440 OUT &7F00,&C5:CALL &A006 ' SYSTEM.464
1450 '
1460 POKE &162,9:POKE &164,&2B
1470 OUT &7F00,&C6:CALL &A006 ' BASIC.664
1480 '
1490 POKE &162,10:POKE &164,&35
1500 OUT &7F00,&C7:CALL &A006 ' SYSTEM.664
1510 '
1520 CLEAR:MEMORY &3FFF
1530 OUT &7F00,&C0
1540 ' Bildschirm des Hauptprogramms aufbauen -----
1550 MODE 2
1560 segment$="SYSTEM"
1570 computer$="464"
1580 rom$=computer$+"/"+segment$
1590 row=396:column=10:breite=2:breit2=2
1600 text$=" CPC-464 ":t$(0)=text$:menu(0)=column
1610 length(0)=LEN(text$):GOSUB 4460
1620 text$=" CPC-664 ":t$(1)=text$:menu(1)=column
1630 length(1)=LEN(text$):GOSUB 4460
1640 text$=" CPC-6128 ":t$(2)=text$:menu(2)=column
1650 length(2)=LEN(text$):GOSUB 4460
1660 text$=" SYSTEM ":t$(3)=text$:menu(3)=column
1670 length(3)=LEN(text$):GOSUB 4460
1680 text$=" BASIC ":t$(4)=text$:menu(4)=column

```

```

1690 length(4)=LEN(text$):GOSUB 4460
1700 text$=" DISC ":t$(5)=text$:menu(5)=column
1710 length(5)=LEN(text$):GOSUB 4460
1720 MOVE 0,0:DRAW 639,0:DRAW 639,360:DRAW 0,360:DRAW 0,0
1730 MOVE 1,0:DRAW 1,360:MOVE 638,0:DRAW 638,360
1740 MOVE 212,0:DRAW 212,360:MOVE 213,0:DRAW 213,360
1750 MOVE 424,0:DRAW 424,360:MOVE 425,0:DRAW 425,360
1760 MOVE 0,44:DRAW 640,44
1770 FOR i=332 TO 356 STEP 4
1780     MOVE 0,i:DRAW 640,i
1790 NEXT i
1800 MOVE 70,353:TAG:PRINT " CPC-464 ";:TAGOFF
1810 MOVE 282,353:TAG:PRINT " CPC-664 ";:TAGOFF
1820 MOVE 490,353:TAG:PRINT " CPC-6128 ";:TAGOFF
1830 WINDOW #1,2,25,6,21
1840 WINDOW #2,29,52,6,21
1850 WINDOW #3,55,79,6,21
1860 sel=1
1870 row=396:column=10:breite=6:breit2=5:text$=" CPC-464 ":GOSUB 4460
1880 row=396:column=306:breite=6:breit2=5:text$=" SYSTEM ":GOSUB 4460
1890 LOCATE 3,24:PRINT "Startadresse:";
1900 LOCATE 30,24:PRINT "Startadresse:";
1910 LOCATE 56,24:PRINT "Startadresse:";
1920 DEFINT x
1930 rom$="464/SYSTEM"
1940 addr1=&8000
1950 addr2=&8D00
1960 addr3=&9500
1970 addr4=&9820
1980 addr5=&9A60
1990 a$=INKEY$:IF a$="" THEN 1990
2000 IF a$=CHR$(&F2) THEN 2130
2010 IF a$=CHR$(&F3) THEN 2210
2020 IF a$=CHR$(&E0) THEN 2290
2030 IF NOT (a$="1" OR a$="2" OR a$="3") THEN 1990
2040 which.one=VAL(a$)
2050 start=addr(which.one)
2060 rom$=romarr$(which.one)
2070 computer$=comparr$(which.one)
2080 segment$=segarr$(which.one)
2090 GOSUB 4010
2100 LOCATE curpos1(which.one)+1,curpos2(which.one)+1
2110 IF COPYCHR$(#0)=" " THEN 2380
2120 LOCATE 1,20:PRINT:GOTO 2100
2130 sel=choice1:text$=t$(sel):GOSUB 4600
2140 choice1=choice1+1:IF choice1=3 THEN choice1=0
2150 sel=choice1:column=menu(sel):text$=t$(sel):breite=6:breit2=5:GOSUB 4460
2160 IF choice1=0 THEN computer$="464"
2170 IF choice1=1 THEN computer$="664"
2180 IF choice1=2 THEN computer$="6128"
2190 rom$=computer$+"/"+segment$
2200 GOTO 1990
2210 sel=choice2+3:text$=t$(sel):GOSUB 4600
2220 choice2=choice2+1:IF choice2=3 THEN choice2=0
2230 sel=choice2+3:column=menu(sel):text$=t$(sel):breite=6:breit2=5:GOSUB 4460
2240 IF choice2=0 THEN segment$="SYSTEM"

```

```

2250 IF choice2=1 THEN segment$="BASIC"
2260 IF choice2=2 THEN segment$="DISC"
2270 rom$=computer$+"/"+segment$
2280 GOTO 1990
2290 GOSUB 4810 ' Windows setzen
2300 CLS:CLS #1
2310 LINE INPUT #1,"",a$:start=VAL(a$)
2320 CLS #1:PRINT #1,"&";HEX$(start,4);
2330 IF start<0 THEN start=start+65536
2340 IF segment$="SYSTEM" AND (start<0 OR start>&3FFF) THEN CLS #1:GOTO 2310
2350 IF segment$="DISC" AND start<49152 THEN CLS #1:GOTO 2310
2360 IF segment$="BASIC" AND start<49152 THEN CLS #1:GOTO 2310
2370 IF start<&4000 THEN ende=16383 ELSE ende=65535
2380 WHILE start<=ende
2390   taste$=UPPER$(INKEY$)
2400   IF taste$=CHR$(3) THEN GOSUB 4710:start=65536:GOTO 4310 ' Ctrl-C
2410   IF taste$=" " THEN CALL &BB18
2420   IF taste$=CHR$(19) THEN singlestep=singlestep XOR 1
2430   IF singlestep=0 THEN 2480
2440   taste$=UPPER$(INKEY$)
2450   IF taste$=" " THEN 2440
2460   IF taste$=CHR$(19) THEN singlestep=singlestep XOR 1
2470   IF taste$=CHR$(3) THEN GOSUB 4710:start=65536:GOTO 4310 ' Ctrl-C
2480   PRINT "&";HEX$(start,4);" ";
2490   GOSUB 4340
2500   adrs=x*13+adrl
2510   adrend=adrs+11
2520   kenner=PEEK(adrs)-48
2530   adrs=adrs+1
2540 ' Kenner 1: Keine Parameter im Befehl -----
2550   IF kenner<>1 THEN 2670
2560     FOR i=adrs TO adrs+11
2570       PRINT CHR$(1)CHR$(PEEK(i));
2580     NEXT i
2590     IF x<>&CF AND x<>&D7 AND x<>&DF AND x<>&EF THEN 2650
2600     start=start+1:GOSUB 4340:lo=x
2610     start=start+1:GOSUB 4340:hi=x
2620     start=start+1:bit16=lo+hi*256
2630     PRINT STRING$(4,8);",&";HEX$(bit16,4)
2640     GOTO 4310
2650     PRINT:start=start+1:GOTO 4310
2660 ' Kenner 2: 16-Bit-Wert -----
2670   IF kenner<>2 THEN 2830
2680     start=start+1:GOSUB 4340:lo=x
2690     start=start+1:GOSUB 4340:hi=x
2700     bit16=lo+hi*256:start=start-2
2710     WHILE CHR$(PEEK(adrs))<>"^"
2720       PRINT CHR$(PEEK(adrs));
2730       adrs=adrs+1
2740     WEND
2750     PRINT "&";HEX$(bit16,4);
2760     WHILE adrs<adrend
2770       adrs=adrs+1
2780       PRINT CHR$(PEEK(adrs));
2790     WEND
2800     start=start+3
2810     PRINT:GOTO 4310
410

```

```

2820 ' Kenner 3: 8-Bit-Wert -----
2830 IF kenner<>3 THEN 2970
2840 start=start+1:GOSUB 4340:start=start-1:bit8=x
2850 WHILE CHR$(PEEK(adrs))<>"^"
2860 PRINT CHR$(PEEK(adrs));
2870 adrs=adrs+1
2880 WEND
2890 PRINT "&";HEX$(bit8,2);
2900 WHILE adrs<adrend
2910 adrs=adrs+1
2920 PRINT CHR$(PEEK(adrs));
2930 WEND
2940 start=start+2
2950 PRINT:GOTO 4310
2960 ' Kenner 4: 8-Bit-Displacement -----
2970 IF kenner<>4 THEN 3070
2980 WHILE CHR$(PEEK(adrs))<>"^"
2990 PRINT CHR$(PEEK(adrs));
3000 adrs=adrs+1
3010 WEND
3020 start=start+1:GOSUB 4340:dis=x
3030 IF dis>127 THEN dis=dis-256
3040 PRINT "&";HEX$(start+dis+1,4)
3050 start=start+1:GOTO 4310
3060 ' Kenner 5: Vorsatzbyte &CB -----
3070 IF kenner<>5 THEN 3220
3080 start=start+1:GOSUB 4340:start=start-1
3090 IF x<&40 THEN pnt=5:laenge=4 ELSE pnt=7:laenge=6
3100 adrs=adrs+2+8*x
3110 IF CHR$(PEEK(adrs))<>"*" THEN 3140
3120 PRINT "DB &CB,&";HEX$(x,2)
3130 start=start+2:GOTO 4310
3140 FOR i=0 TO laenge
3150 PRINT CHR$(PEEK(i+adrs));
3160 NEXT i
3170 a$=CHR$(PEEK(adrs+pnt))
3180 IF a$="M" THEN PRINT "(HL)" ELSE PRINT a$
3190 start=start+2
3200 GOTO 4310
3210 ' Kenner 7: Vorsatzbyte &ED -----
3220 IF kenner<>7 THEN 3500
3230 start=start+1:GOSUB 4340:codebyte=x
3240 FOR i=adrs3 TO adrs3+800 STEP 14
3250 IF PEEK(i)=codebyte THEN 3280
3260 NEXT i
3270 PRINT "DB &ED,&";HEX$(codebyte,2):start=start+1:GOTO 4310
3280 adrs=i+1:adrend=adrs+12:kenner=PEEK(adrs)-48
3290 IF kenner<>1 THEN 3340
3300 FOR i=adrs+1 TO adrs+12
3310 PRINT CHR$(PEEK(i));
3320 NEXT i
3330 start=start+1:PRINT:GOTO 4310
3340 start=start+1:GOSUB 4340:lo=x
3350 start=start+1:GOSUB 4340:hi=x
3360 start=start+1
3370 bit16=lo+hi*256
3380 adrs=adrs+1

```

```

3390      WHILE CHR$(PEEK(adrs))<>"^"
3400          PRINT CHR$(PEEK(adrs));
3410          adrs=adrs+1
3420      WEND
3430      PRINT "&";HEX$(bit16,4);
3440      WHILE adrs<adrend-1
3450          adrs=adrs+1
3460          PRINT CHR$(PEEK(adrs));
3470      WEND
3480      PRINT:GOTO 4310
3490 ' Kenner 6 und 8: Vorsatzbytes &DD und &FD -----
3500      IF kenner=6 THEN reg$="IX":vorbyt=&DD
3510      IF kenner=8 THEN reg$="Y":vorbyt=&FD
3520      start=start+1:GOSUB 4340:prebyte=x
3530      IF prebyte<>&36 THEN 3600
3540          IF reg$="IX" THEN PRINT "LD    (IX"; ELSE PRINT "LD    (Y";
3550          PRINT "+&";
3560          start=start+1:GOSUB 4340:PRINT HEX$(x,2);
3570          PRINT "),"&";
3580          start=start+1:GOSUB 4340:PRINT HEX$(x,2)
3590          start=start+1:GOTO 4310
3600      IF prebyte<>&CB THEN 3790
3610          start=start+2:GOSUB 4340:start=start-1:codebyte=x
3620          FOR i=adrs5 TO adrs5+434 STEP 14
3630              IF PEEK(i)=codebyte THEN 3670
3640          NEXT i
3650          PRINT "DB    &ED,&CB"
3660          GOTO 4310
3670          adrs=i+1
3680          WHILE CHR$(PEEK(adrs))<>"^"
3690              PRINT CHR$(PEEK(adrs));
3700              adrs=adrs+1
3710          WEND
3720          GOSUB 4340:PRINT "&";HEX$(x,2);
3730          start=start+1
3740          WHILE adrs<i+13
3750              adrs=adrs+1
3760              PRINT CHR$(PEEK(adrs));
3770          WEND
3780          PRINT:start=start+1:GOTO 4310
3790      FOR i=adrs4 TO adrs4+38*15 STEP 15
3800          IF PEEK(i)=prebyte THEN 3850
3810      NEXT i
3820      PRINT "DB    &";HEX$(vorbyt,2);","&";HEX$(prebyte,2)
3830      start=start+1
3840      GOTO 4310
3850      adrs=i+1:kenner=PEEK(adrs)-48
3860      adrend=adrs+13
3870      IF kenner<>1 THEN 3950
3880          FOR i=adrs+1 TO adrs+13
3890              a$=CHR$(PEEK(i))
3900              IF a$<>"X" THEN PRINT a$;:GOTO 3930
3910              IF PEEK(i-1)<>ASC("I") THEN PRINT "X";:GOTO 3930
3920              IF reg$="IY" THEN PRINT "Y"; ELSE PRINT "X";
3930          NEXT i
3940      PRINT:start=start+1:GOTO 4310
3950      IF kenner<>2 THEN 4110

```

```

3960     adrs=adrs+1
3970     WHILE CHR$(PEEK(adrs))<>"^"
3980         PRINT CHR$(PEEK(adrs));
3990         adrs=adrs+1
4000     WEND
4010     start=start+1:GOSUB 4340:lo=x
4020     start=start+1:GOSUB 4340:hi=x
4030     bit16=lo+hi*256
4040     PRINT "&";HEX$(bit16,4);
4050     WHILE adrs<adrend
4060         adrs=adrs+1
4070         PRINT CHR$(PEEK(adrs));
4080     WEND
4090     start=start+1:PRINT
4100     GOTO 4310
4110 IF kenner<>3 THEN start=start+1:GOTO 4310
4120     adrs=adrs+1
4130     start=start+1:GOSUB 4340:bit8=x
4140     WHILE CHR$(PEEK(adrs))<>"^"
4150         a$=CHR$(PEEK(adrs))
4160         IF a$<>"X" THEN PRINT a$;:GOTO 4190
4170         IF PEEK(adrs-1)<>ASC("I") THEN PRINT "X";:GOTO 4190
4180         IF reg$="IY" THEN PRINT "Y"; ELSE PRINT "X";
4190         adrs=adrs+1
4200     WEND
4210     PRINT "&";HEX$(bit8,2);
4220     WHILE adrs<adrend
4230         adrs=adrs+1
4240         a$=CHR$(PEEK(adrs))
4250         IF a$<>"X" THEN PRINT a$;:GOTO 4280
4260         IF PEEK(adrs-1)<>ASC("I") THEN PRINT "X";:GOTO 4280
4270         IF reg$="IY" THEN PRINT "Y"; ELSE PRINT "X";
4280     WEND
4290     start=start+1
4300     PRINT:GOTO 4310
4310 WEND
4320 GOTO 1990
4330 ' Unterprogramme -----
4340 x=3 ' Dummy-Wertzuweisung
4350 IF rom$="6128/BASIC" THEN CALL &A000,start,0x:RETURN
4360 IF rom$="6128/SYSTEM" THEN CALL &A000,start,0x:RETURN
4370 IF RIGHT$(rom$,4)="DISC" THEN CALL &A003,start,0x:RETURN
4380 IF rom$="464/BASIC" THEN OUT &7F00,&C4
4390 IF rom$="464/SYSTEM" THEN OUT &7F00,&C5
4400 IF rom$="664/BASIC" THEN OUT &7F00,&C6
4410 IF rom$="664/SYSTEM" THEN OUT &7F00,&C7
4420 IF RIGHT$(rom$,5)="BASIC" THEN x=PEEK(start-32768):OUT &7F00,&C0:RETURN
4430 x=PEEK(start+16384):OUT &7F00,&C0
4440 RETURN
4450 ' Ein Element der Menueleiste zeichnen -----
4460 PLOT 800,800,1:TAG:MOVE column+5,row-2
4470 PRINT text$;:TAGOFF
4480 MOVE column,row+2:DRAW 8*LEN(text$)+9,0,1
4490 MOVE column,row+4:DRAW 8*LEN(text$)+9,0
4500 FOR i=1 TO breite
4510 DRAW 0,-22:MOVER i,22:NEXT i
4520 MOVER -1,0

```

```

4530 MOVER 0,-22:xp=XPOS:yp=YPOS:row2=row
4540 FOR i=1 TO breit2:yp=yp-1:row2=row2-1
4550 MOVE xp,yp:DRAW column,row2-18:NEXT i
4560 DRAW 0,23:MOVER -1,-23:DRAW 0,23
4570 column=column+8*LEN(text$)+24
4580 RETURN
4590 ' Verstaerkung der Menueleiste entfernen -----
4600 column=menu(sel)
4610 ln=length(sel)
4620 MOVE column,row+2
4630 MOVER 8*LEN(t$(sel))+9,0
4640 FOR i=1 TO 6:DRAW 0,-22,0:MOVER 1,22:NEXT i
4650 MOVER 0,-20:xp=XPOS:yp=YPOS:row2=row
4660 FOR i=3 TO 7:yp=yp-1:row2=row2-1
4670 MOVE xp,yp:DRAW column-2,row2-18,0:NEXT i
4680 breite=2:breit2=2:text$=t$(sel):GOSUB 4460
4690 RETURN
4700 ' Aktuellen Arbeitsstand fuer F1, F2 und F3-Fortsetzung speichern --
4710 IF computer$="484" THEN index=1
4720 IF computer$="664" THEN index=2
4730 IF computer$="6128" THEN index=3
4740 addr(index)=start
4750 romarr$(index)=rom$
4760 comparr$(index)=computer$
4770 segarr$(index)=segment$
4780 curpos1(index)=POS(0)-1:curpos2(index)=VPOS(0)-1
4790 RETURN
4800 ' Windows entsprechend computer$ setzen -----
4810 IF computer$="484" THEN WINDOW 2,25,6,21:WINDOW #1,17,25,24,24
4820 IF computer$="664" THEN WINDOW 29,52,6,21:WINDOW #1,44,52,24,24
4830 IF computer$="6128" THEN WINDOW 55,79,6,21:WINDOW #1,70,78,24,24
4840 RETURN

```

Bei der Eingabe des Programms können Sie natürlich alle REM-Zeilen weglassen und auch auf die Einrückungen des Programmtextes verzichten.

Besonders die Einrückungen aber können bei der Fehlersuche in Ihrer eingetippten Version recht hilfreich sein. Verzichteten Sie aber bitte auf Programmerweiterungen oder zusätzliche REMs und Einrückungen, denn zeitweise sind während des Programmaufs nur noch etwa 100 Bytes im Speicher frei!

Nachdem Sie das Listing abgetippt und auf der Diskette als

"SYNOPSIS.BAS"

gespeichert haben, sollten Sie durch gleichzeitiges Drücken der Control-, Shift- und Escape-Taste einen Reset auslösen. Danach können Sie es mit RUN "SYNOPSIS" starten.

Falls Sie versuchen, SYNOPSIS.BAS auf einem anderen Computer als dem CPC-6128 zu starten, gibt das Programm eine Fehlermeldung aus und stoppt die Programmausführung.

Der Computer gibt ein Titelbild aus und fragt Sie, ob Sie einen

Farbmonitor besitzen. Für monochrome Monitore wird helle Schrift auf dunklem Grund gewählt, für den Farbmonitor dunkle Schrift auf hellem Grund. Nun müssen Sie etwas warten, denn der Computer lädt immerhin fast 80 Kilobyte an Daten in den Speicher.

Danach erscheint der Arbeitsbildschirm des Programms.

Alle Aktionen werden auf diesem durchgeführt. In der obersten Zeile finden Sie sechs Kästchen. In den linken drei Kästchen stehen die Texte "CPC-464", "CPC-664" und "CPC-6128", in den rechten "SYSTEM", "BASIC" und "DISC".

Verstärkt dargestellt sind die Kästchen "CPC-464" und "SYSTEM".

Wenn Sie jetzt die Cursor-Links-Taste drücken, wandert die stärkere Umrandung vom 464-Kästchen zum 664-Kästchen, beim nächsten Drücken der Taste zum 6128-Kästchen. Weiter geht es dann wieder mit dem CPC-464-Kästchen.

Drücken Sie die Cursor-Rechts-Taste, wandert die Verstärkung vom SYSTEM-Kästchen zum BASIC-Kästchen, weiter zum DISC-Kästchen und wieder zum SYSTEM-Kästchen.

Damit ist klar: Wollen Sie ein bestimmtes ROM disassemblieren, dann stellen Sie mit der Cursor-Links-Taste den Computertyp ein, mit der Cursor-Rechts-Taste das gewünschte ROM.

Haben Sie Ihre Wahl getroffen, können Sie die COPY-Taste drücken. Diese dient sozusagen als "Annahmebestätigung" für Ihre Auswahl.

Sehen Sie jetzt das Zentrum des Bildschirms an, werden Sie entdecken, daß der Bildschirm hauptsächlich von drei großen Fenstern bedeckt ist, die alle die gleichen Abmessungen haben. Für jeden der drei Computertypen ist ein Fenster reserviert.

Disassemblieren Sie zum Beispiel das Basic-ROM des CPC-464, erscheint die Ausgabe im 464-Fenster. Listen Sie dagegen das Betriebssystem-ROM des 6128, gibt der Computer den Maschinencode im 6128-Fenster aus.

Am unteren Rand jedes Fensters ist jeweils ein weiteres (kleines) Fenster zu entdecken: Dieses dient als Eingabeort der Startadresse, von der ab disassembliert werden soll.

Wählen Sie mit den Cursortasten zur Übung bitte das 6128-Basic-ROM aus und bestätigen Sie Ihre Wahl mit der COPY-Taste. Der Cursor erscheint im kleinen Fenster unter dem 6128-Fenster und zeigt die Eingabebereitschaft an. Geben Sie hier die gewünschte Adresse an. Diese Angabe kann dezimal, hexadezimal oder auch binär gemacht werden. In unserem Fall nehmen wir &0030. Der Computer weist Ihre Eingabe zurück und erwartet eine neue Adresse. Warum? Weil das Basic-ROM im Adreßbereich &C000 bis &FFFF liegt.

Damit ist Ihre Eingabe "&0030" rund 50 Kilobytes von der richtigen Adresse entfernt, und der Computer moniert das. Wiederholen Sie Ihre Eingabe als "&C090", und Sie sehen, daß der Computer damit einverstanden ist.



Er beginnt sofort mit der Disassemblierung des Basic-ROMs ab der Adresse &C090:

```

&C090 CALL &C10D
&C093 JR   NC,&C058
&C095 CALL &DE4D
&C098 CALL &EECF
&C09B JR   NC,&C0A7
&C09D CALL &DE4D
&C0A0 OR   A
&C0A1 SCF

```

Anhalten können Sie die Auflistung durch einen Druck auf die Space-Taste. Drücken Sie eine beliebige Taste, geht es wieder weiter.

Bei besonders kniffligen Suchen kann es oft hilfreich sein, nur jeweils eine einzige Anweisung zu disassemblieren und in Ruhe zu studieren. Dazu bietet SYNOPSIS 6128 einen Einzelschritt-Modus. Er wird durch Drücken von Control-S ("S" für "Singlestep") eingeschaltet. Der Computer listet danach einen Maschinenbefehl auf und wartet, bis Sie eine Taste drücken. Er gibt den nächsten Befehl aus und wartet wieder.

So können Sie das endlos machen. Durch erneutes Drücken der Control-S-Taste schalten Sie den Singlestep-Modus wieder ab, und die Auflistung der Maschinencodes geschieht in der üblichen Geschwindigkeit. Control-C ist als Abbruchtaste vorgesehen. Sobald Sie diese Tastenkombination betätigen, friert der Computer die Bildschirmausgabe ein und erlaubt es Ihnen, mit den Cursor-Tasten in der Menüleiste der ersten Zeile ein anderes ROM und einen anderen Computer auszuwählen.

Tippen Sie bitte jetzt Control-C und COPY. Als Adresse nehmen Sie wieder &C090. Falls Sie jetzt gar nichts mehr sehen, kann das daran liegen, daß noch der Einzelschritt-Modus eingeschaltet ist. Durch Drücken der Control-S-Taste beseitigen Sie das nötigenfalls. Sobald der Computer die Disassemblierung aufgenommen hat, können Sie die Anzeige ablesen.

Wenn das Bildschirmfenster vollgeschrieben ist, drücken Sie bitte Control-C, um den Computer zu stoppen. In der untersten Zeile mußte jetzt "&C0B4 CALL &C398" stehen.

Wählen Sie nun mit Cursor-Links das CPC-464-Fenster an und lassen Sie den ROM-Typ weiterhin auf "BASIC" stehen. Weiter drücken Sie die COPY-Taste und geben als Startadresse wieder &C090 an.

Der entsprechende Ausschnitt des CPC-464-Basic-ROMs wird von SYNOPSIS 6128 gelistet.

Auch hier drücken Sie Control-C und wiederholen das Ganze für den CPC-664.

Der Bildschirm dürfte etwa so aussehen:

```

----- CPC-464 ----- CPC-664 ----- CPC-6128 -----
&C090 LD HL,&C0CC      &C090 CALL &C10D      &C090 CALL &C10D
&C093 CALL &C341      &C093 JR NC,&C058      &C093 JR NC,&C058
&C096 CALL &DDCB      &C095 CALL &DE52      &C095 CALL &DE4D
&C099 LD A,(&AC1C)    &C098 CALL &EED4      &C098 CALL &EECF
&C09C OR A           &C09B JR NC,&C0A7      &C09B JR NC,&C0A7
&C09D JR Z,&C0B0      &C09D CALL &DE52      &C09D CALL &DE4D
&C09F CALL &C102      &C0A0 OR A           &C0A0 OR A
&C0A2 JR NC,&C064      &C0A1 SCF           &C0A1 SCF
&C0A4 LD A,(HL)      &C0A2 CALL Z,&E869      &C0A2 CALL Z,&E864
&C0A5 OR A           &C0A5 JR NC,&C08A      &C0A5 JR NC,&C08A
&C0A6 JR Z,&C099      &C0A7 CALL NC,&C0DE      &C0A7 CALL NC,&C0DE
&C0A8 CALL &E6D2      &C0AA LD HL,&AC8A      &C0AA LD HL,&AC8A
&C0AB CALL &C17A      &C0AD JR &C0B7      &C0AD JR &C0B7
&C0AE JR &C099      &C0AF CALL &CAFC      &C0AF CALL &CAF9
&C0B0 CALL &CA3B      &C0B2 JR NC,&C0AF      &C0B2 JR NC,&C0AF
&C0B3 JR NC,&C0B0      &C0B4 CALL &C39B      &C0B4 CALL &C39B

-----
Startadresse: &C090      Startadresse: &C090      Startadresse: &C090
-----

```

Wie unschwer festzustellen, ist es also mit der vielbeschworenen Kompatibilität der Schneider-Computer nicht allzu weit her. Alle offiziell dokumentierten Routinen sind kompatibel, aber viele Sachen lassen sich mit diesen einfach nicht machen. Während das ROM des CPC-664 nur sehr wenig von dem des CPC-6128 abweicht, sind die Änderungen gegenüber dem CPC-464 doch sehr gravierend. Die Entwickler bei Amstrad haben für die beiden Nachfolgecomputer nahezu das gesamte ROM "umgekrempelt".

Mit SYNOPSIS 6128 können Sie auf die "Jagd nach den verlorenen Routinen" gehen und damit Programme schreiben, die sich selbstständig an den verwendeten Computer anpassen. Sie wissen doch noch, wie man den Computer ermittelt, auf dem das Programm läuft? Hier der Maschinencode:

```

CALL &B900
LD A,(&C002)
LD (&0160),A
RET

```

In einem kleinen Basic-Programm sieht das dann so aus:

```

100 POKE &160,&CD:POKE &161,&0:POKE &162,&B9
110 POKE &163,&3A:POKE &164,&2:POKE &165,&C0
120 POKE &166,&32:POKE &167,&60:POKE &168,&1
130 POKE &169,&C9
140 CALL &160:cpcversion=PEEK(&160)
150 IF cpcversion=0 THEN PRINT "Ein CPC-464!"
160 IF cpcversion=1 THEN PRINT "Ein CPC-664!"
170 IF cpcversion=2 THEN PRINT "Ein CPC-6128!"
180 END

```

Eine Möglichkeit von SYNOPTIS 6128 wurde noch nicht besprochen: Stellen Sie sich vor, Sie sind gerade dabei, eine ROM-Routine zu suchen.

Dazu haben Sie ein Stück der jeweiligen Basic-ROMs disassembliert. Nun glauben Sie, die korrespondierenden Routinen gefunden zu haben, wollen aber auch diejenigen Adressen disassemblieren, die hinter den auf dem Bildschirm gezeigten liegen.

Normalerweise müßten Sie jetzt wieder das passende ROM und den Computertyp selektieren und die Startadresse angeben. Nur wird dabei leider das Bildschirmfenster gelöscht, und die alte Anzeige verschwindet.

Drücken Sie dagegen die Tasten F1 bis F3, setzt der Computer die Disassemblierung des jeweiligen ROMs fort. Nach Control-C können Sie wieder eine der Funktionstasten betätigen und sich ein anderes ROM ansehen. Die Tastaturbelegung ist wie folgt:

Taste F1: CPC-464-Disassemblierung fortsetzen  
Taste F2: CPC-664-Disassemblierung fortsetzen  
Taste F3: CPC-6128-Disassemblierung fortsetzen

Viel Spaß und vor allen Dingen Erfolg bei der Benutzung des Programmes SYNOPTIS 6128! Für die "wahren" Computerfreaks hier noch Anmerkungen zum Aufbau von SYNOPTIS.BAS. Als Überblick zuerst die Speicheraufteilung:

Die zweite 64K-Speicherbank, die in den CPC-6128 eingebaut ist, kann eigentlich nur unter CP/M Plus und mit dem Bank-Manager BANKMAN genutzt werden. Da BANKMAN unverständlicherweise listgeschützt ist, ist es recht mühsam, auf die Methoden der Speicherumschaltung zu kommen. Doch hier sind sie: Die Umschaltung wird über OUT-Befehle gesteuert, die in ein bisher unbenutztes Register des Gate-Arrays Bytes übertragen. Um die übrigen Funktionen des Gate-Array nicht zu stören, müssen die beiden höchsten Bits des Werts immer auf logisch 1 stehen. Damit sind nur die Werte &C0 bis &C7 (192 bis 199) sinnvoll.

Über die Portadresse &7F00 können Sie die Bytes ausgeben. OUT &7F00,&C0 schaltet die Standard-Konfiguration ein.

Alle 16K-Blöcke sind in der normalen Reihenfolge, und die Speichererweiterung bleibt ungenutzt.

Die Werte &C1 bis &C3 werden unter CP/M Plus verwendet. In ihnen ist immer der Adreßbereich zwischen &C000 und &FFFF identisch, er enthält den letzten 16K-Block der Speichererweiterung.

Die übrigen Blöcke werden andauernd hin- und hergeschaltet, um einerseits eine TPA (Transient Program Area) von 61 Kilobyte zu realisieren und andererseits auch noch das komplette BDOS (Basic Disk Operating System), das BIOS (Basic Input/Output System), diverse Diskettenpuffer und den CCP (Console Command Processor) aufzunehmen. Der CCP liegt normalerweise am Beginn der TPA, wird aber beim Start eines Programms von dort in die Speichererweiterung wegkopiert und beim nächsten Warmstart des Computers wieder

von dort zurückübertragen. Die Werte &C1 bis &C3 sind unter Basic nicht brauchbar. Sie führen entweder zum Systemabsturz, weil der Speicherblock mit den lebenswichtigen RST-Restart-Befehlen weggeblendet wird (&C2) oder zu einem scheinbaren Absturz, bei dem in Wirklichkeit nur der Video-Prozessor nicht auf den neuen Speicher zugreifen kann, während die ROM-Routinen zur Bildschirmausgabe in diesen Speicher schreiben (&C1). Eine besondere Variante ist das Byte &C3. Bei diesem wird zuerst einmal der Speicher zwischen &C000 und &FFFF ersetzt, was die Bildschirmausgabe unterbindet. Zusätzlich aber wird der normale Video-RAM in den Adreßbereich &4000 bis &7FFF gebracht.

Für Basic von Bedeutung sind die Werte &C4 bis &C7. Sie ersetzen den Speicher im Block &4000 bis &7FFF der Reihe nach durch einen 16K-Block der Speichererweiterung. Wie das funktioniert, zeigt die Tabelle:

OUT &7F00,&C4	Block 1 der Speichererweiterung (&0000 bis &3FFF)
OUT &7F00,&C5	Block 2 der Speichererweiterung (&4000 bis &7FFF)
OUT &7F00,&C6	Block 3 der Speichererweiterung (&8000 bis &BFFF)
OUT &7F00,&C7	Block 4 der Speichererweiterung (&C000 bis &FFFF)

Mit diesen vier OUT-Befehlen kann sozusagen ein "Fenster" von 16 Kilobyte Größe über der Speichererweiterung verschoben werden. Mit allen normalen Befehlen wie PEEK und POKE können Sie auf dieses Fenster zugreifen. Ist erst einmal der Speicher eingeblendet, wird er von allen Systemkomponenten (Hard- und Software) bis auf den Video-Chip als vollwertiger Speicher anerkannt. OUT &7F00,&C0 stellt die normale Konfiguration wieder her.

Angenehmerweise laden die Diskettenroutinen auch Dateien in die Speichererweiterung, wenn sie in geeigneter Weise über die OUT-Befehle in den Adreßbereich eingeblendet wird.

So steht bei SYNOPSIS 6128 im ersten Block der Speichererweiterung die Datei BASIC.464, im zweiten SYSTEM.464, im dritten Block BASIC.664 und im vierten SYSTEM.664.

Selbstverständlich müssen die Speicherblöcke mit MEMORY &3FFF vor einem Überschreiben durch das Basic-Programm geschützt werden. Damit bleiben aber nur noch etwa 16K frei für das ganze Programm SYNOPSIS.BAS. Somit lassen sich die Disassembler-Codes weder in Strings noch in DATA-Zeilen unterbringen. Dafür ist der Speicher zwischen &8000 (32768) und 42619 noch völlig leer.

Beginnend bei &8000 stehen deshalb alle Disassembler-Codes im Klartext im Speicher. Sie werden beim Programmstart aus der Datei DISCODE.BIN geladen. Ab &A000 (40960) finden sich dann noch die drei kurzen Maschinenroutinen ROM\_PEEK, FLOPPY\_PEEK und BIN\_LOAD. Sie haben eine Länge von nicht ganz 200 Bytes.

Trotz dieser nahezu optimalen Speicherausnutzung stehen nach dem Laden und Starten von SYNOPSIS.BAS während des Einlesens der Dateien von der Diskette nur noch etwa 100 Bytes zur Verfügung. Dies kommt daher, weil die Diskettenoperationen immer über einen zwei Kilobytes großen Pufferspeicher ausgeführt werden müssen. Nach dem Laden kann der Speicher aber wieder freigegeben werden,

und die Basic-Variablen können ihn benutzen.

Eine ausführliche Erläuterung des Programms würde den Rahmen des Buches wohl sprengen. Dennoch gehen wir kurz auf den Aufbau von SYNOPSIS.BAS ein. Die Zeilenstruktur sieht so aus:

- Zeile 1060 bis 1120: Hier wird geprüft, auf welchem Computer das Programm gestartet wurde. Ein Programmstart auf dem CPC-464 oder CPC-664 hat die Ausgabe der Fehlermeldung "SYNOPSIS 6128 arbeitet mit diesem Computer nicht!" zur Folge.
- Zeile 1130 bis 1320: Diese Zeilen stellen die Bildschirmfarben ein und geben das Titelbild aus.
- Zeile 1330 bis 1530: Alle nötigen Dateien werden von der Diskette nachgeladen. Die Speicherobergrenze wird festgelegt, und die Variablen werden gelöscht.

Die Befehle der Zeilen 1180 bis 1530 werden übrigens nur einmal ausgeführt. Bei einem erneuten Programmstart werden sie übersprungen, weil die Dateien ja schon geladen sind. Das spart Ihnen enorm Zeit. Allerdings bedeutet es auch, daß Sie SYNOPSIS.BAS während des Eintippens nicht unbedingt starten sollten, weil einige Teile des Programms dann noch nicht arbeitsfähig sind und der Computer abstürzen könnte.

- Zeile 1540 bis 1910: Der Computer baut den Arbeitsbildschirm auf. Dies geschieht über Grafikbefehle wie DRAW und MOVE und über Textbefehle wie WINDOW, LOCATE und PRINT.

- Zeile 1920 bis 1980: Die wichtigsten Programmvariablen werden festgelegt. Die Variablen addr1 bis addr5 enthalten Zeiger auf die Disassembler-Tabellen im Speicher:

\* addr1 zeigt auf die Haupttabelle, die die Codes von &00 bis &FF aufnimmt. Alle Einträge sind 13 Bytes lang, wodurch der Computer mit einfachen Berechnungen an die passende Stelle kommt. Das erste Byte jedes Eintrags ist ein spezieller Code, der dem Disassembler sagt, welches Format der Befehl hat:

- 1 - Der Befehl besitzt keine Operanden.
- 2 - Der Befehl besitzt einen 16-Bit-Operanden.
- 3 - Der Befehl besitzt einen 8-Bit-Operanden.
- 4 - Der Befehl besitzt einen Operanden als 8-Bit-Displacement.
- 5 - Verweist auf die Tabelle der Befehle mit Vorsatzbyte &CB.
- 6 - Verweist auf die Tabelle der Befehle mit Vorsatzbyte &DD.
- 7 - Verweist auf die Tabelle der Befehle mit Vorsatzbyte &ED.
- 8 - Verweist auf die Tabelle der Befehle mit Vorsatzbyte &FD.

Die Position eines eventuellen Operanden wird durch ein Potenzierungs-Symbol (^) kenntlich gemacht, zum Beispiel sieht der Befehl LD HL,(Adresse) intern so aus:

2LD...HL, (^).

Die Punkte dienen hier -ebenso wie in CODEGEN.BAS- nur als Platz-

halter und werden im Speicher nicht verwendet.

\* addr2 zeigt auf die Tabelle, in der alle Befehle mit dem Vorsatzbyte &CB fein säuberlich aufgelistet sind. Das Kennbyte kann man sich hier sparen, weil alle Befehle ohnehin dasselbe Format besitzen.

\* addr3 verweist auf die Tabelle der Befehle mit dem Vorsatzbyte &ED. Da es nicht für jedes Byte einen Befehl im Z80-Befehlssatz gibt, muß zusätzlich zum Kennbyte jedem Eintrag der Prozessorcode vorangestellt werden. So sieht der Befehl OUT (C),B im Speicher so aus:

A1OUT..(C),B..

Das "A" besitzt den ASCII-Code 65 oder in hexadezimaler Schreibung &41. Damit ist der Prozessorcode &ED &41. Das Kennbyte "1" gibt an, daß der Befehl keine Operanden besitzt.

\* addr4 zeigt auf die Tabelle der Befehle mit den Vorsatzbytes &DD und &FD. Dabei handelt es sich ausschließlich um Kommandos, die mit den Indexregistern IX und IY arbeiten. Da der Befehlssatz der beiden Indexregister völlig symmetrisch ist, reicht es, die Codes einmal in den Speicher aufzunehmen. Die Codierung gleicht der unter addr3 beschriebenen.

\* addr5 enthält einen Zeiger auf die Tabelle mit den Doppelbyte-Codes &DD &CB und &FD &CB. Aufbau wie bei addr3.

- Zeile 1990 bis 2280: Die Cursor-Links- und Cursor-Rechts-Tasten werden abgefragt und in die entsprechende Bildschirmausgabe umgesetzt. Die Tasten "1", "2" und "3" behandelt das Programm gesondert in den Zeilen 2030 bis 2120.

- Zeile 2290 bis 2370: SYNOPSIS.BAS fordert vom Benutzer eine Startadresse an und überprüft den Wert.

- Zeile 2380 bis 4310: In eine WHILE-WEND-Schleife ist der komplette Disassembler "eingebettet". Durch die REM-Zeilen und die Einrückungen dürfte die Funktionsweise des Disassemblers deutlich werden.

Ab der Zeile 4330 beginnen die Unterprogramme:

- Zeile 4340 bis 4440: Anhand des Strings rom\$ wird das passende ROM ausgelesen. Die Adresse muß in start stehen, zurückgegeben wird der Wert in der Integervariablen x (DEFINT x in Zeile 1920).

- Zeile 4450 bis 4580: Dieses Unterprogramm zeichnet ein Kästchen der Menüleiste. Die Variablen müssen wie folgt gesetzt sein:

```
row      = Grafikzeile (0 bis 400)
column   = Grafikspalte (0 bis 640)
breite   = Dicke des senkrechten Strichs
```

breit2 = Dicke des waagerechten Strichs  
text\$ = Auszugebender Text

- Zeile 4590 bis 4690: Das Unterprogramm entfernt die Verstärkung eines Menükästchens. Die Nummer des Kästchens wird der Subroutine in sel übergeben.
- Zeile 4700 bis 4790: Hier speichert der Computer bei Control-C all die Variablen ab, die bei der Fortsetzung der Disassemblierung mit F1, F2 oder F3 wieder benötigt werden.
- Zeile 4800 bis 4840: Die sechs Window-Befehle setzen abhängig von der Variablen computer\$ das Ausgabefenster und das Fenster, in dem die Startadresse angefordert wird, fest.

## 28. Sparen Sie Farbbänder!

Mit diesem recht kurzen, aber dennoch sehr nützlichen Programm können Sie bares Geld sparen: Die Nutzungsdauer der Farbbänder Ihres Druckers wird drastisch verlängert!

Die Aufgabe des Programms besteht nun darin, Dateien mit Textzeilen einzulesen und dann auf dem Drucker auszugeben. Der ganze Trick besteht darin, daß jede Zeile zweimal auf dieselbe Stelle auf dem Papier gedruckt wird. Dadurch können Sie auch altersschwache Farbbänder für einen längeren Zeitraum verwenden.

Im Endeffekt müssen Sie sich weniger Farbbänder anschaffen und sparen dadurch Geld.

Damit das Programm korrekt arbeiten kann, muß der zum Einsatz kommende Drucker den Wagenrücklauf (CR-Carriage Return, ASCII-Code 13) als Zurückfahren des Druckkopfes zum linken Rand interpretieren, darf aber dabei keinen Zeilenvorschub (LF-Line Feed, ASCII-Code 10) auslösen. Korrekt verhalten sich fast alle an den Schneider-Computern funktionierenden Drucker, wie zum Beispiel der Schneider NLQ-401, DMP-2000 oder Brother M-1009.

Natürlich halbiert diese Druckmethode die Geschwindigkeit, mit der die Daten zu Papier gebracht werden. Besonders bei der NLQ-Briefqualität der Schneider-Drucker sinkt die Geschwindigkeit beinahe auf Typenraddrucker-Niveau. Das ist eben der Preis für das bessere Schriftbild.

Die Programmbedienung ist kinderleicht: Nach dem Programmstart gibt das Programm eine kurze Anleitung aus und will von Ihnen den Dateinamen der zu druckenden Datei wissen. Sie können das Unterdrückungszeichen "!" vor den Dateinamen setzen, müssen es aber nicht tun. Nach dieser Eingabe läuft alles automatisch, und das Programm meldet sich nach beendetem Ausdruck mit der READY-Meldung des Basic-Interpreters zurück.

Und natürlich finden Sie hier auch das Listing des Basic-Programms.

### 28.1 Listing: FBSAVE.BAS

```

100 ' *****
110 ' *                               *
120 ' *      FARBBÄNDER SPAREN      *
130 ' *                               *
140 ' *****
150 '
160 ' INITIALISIERUNG UND FARBEN *****
170 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
180 SYMBOL AFTER 250
190 SYMBOL 253,&66,&0,&78,&C,&7C,&CC,&76,&0
200 SYMBOL 254,&66,&0,&3C,&66,&66,&66,&3C,&0
210 SYMBOL 255,102,0,102,102,102,102,62,0
220 PAPER 0:BORDER 10:PEN 3:SPEED INK 30,20
230 ' TITELBILD UND ANLEITUNG *****

```



```

240 LOCATE 1,1:PRINT STRING$(40,210);
250 LOCATE 1,2:PRINT CHR$(24);SPACE$(11);"FARBBAENDER SPAREN";
260 PRINT SPACE$(11);CHR$(24);
270 LOCATE 10,5
280 PRINT STRING$(40,210)
290 PEN 1:LOCATE 2,10:PRINT "Dieses Programm hilft Ihnen, Drucker-
300 PRINT:PRINT " farbb"+CHR$(253)+"nder zu sparen: es druckt Texte"
310 PRINT:PRINT " aus Dateien zweifach "+CHR$(255)+"bereinander aus."
320 PEN 3:PRINT
330 PRINT:PRINT " Bitte schalten Sie den Drucker ein"
340 PRINT:PRINT " und legen Sie die Cassette/Disc ein!"
350 PEN 1
360 PRINT:PRINT:INPUT " Dateiname? ",file$
370 IF LEFT$(file$,1)<>"!" THEN file$="!" + file$
380 LOCATE 13,24
390 PEN 2:PRINT " BITTE WARTEN!";
400 ' AUSDRUCKEN DER DATEI *****
410 OPENIN file$
420 WHILE NOT EOF
430     LINE INPUT #9,a$
440     PRINT #8,a$;CHR$(13);a$;CHR$(13);CHR$(10);
450 WEND
460 CLOSEIN
470 MODE 1:PEN 1:END
480 ' *****

```

## 29. VARLIST – Variablen gelistet

Wer fremde Basic-Programme analysiert, steht meist vor dem großen Problem, keinen Überblick über die Variablen des Programms zu haben. Abhilfe schafft VARLIST, ein Variablenlister. Er eignet sich auch dazu, Ihre eigenen Programme zu dokumentieren.

VARLIST liest ein Programm von der Cassette oder Diskette in den Speicher ein. Dann durchsucht es jede Programmzeile nach Variablen und gibt – wahlweise auf dem Bildschirm oder Drucker – eine Liste aus.

Um VARLIST zu benutzen, sollten Sie das zu untersuchende Programm im ASCII-Format abspeichern. Dazu hängen Sie an den SAVE-Befehl ein „A“ an, zum Beispiel:

```
SAVE "PROGRAMM",A
```

Mit RUN "VARLIST" starten Sie den Variablenlister. Das Programm gibt kurze Hinweise zu seiner Funktion aus und fragt Sie, ob Sie die Ausgabe auf dem Bildschirm oder auf dem Drucker wünschen. Ebenso will es von Ihnen den Dateinamen des zu analysierenden Programms erfahren. Sie können diesen wahlweise mit oder ohne Unterdrückungszeichen "!" angeben, VARLIST fügt das Zeichen so- wieso ein.

Es erscheint die Meldung "Press PLAY or insert disc ...", worauf Sie Gelegenheit haben, die passende Diskette oder Cassette einzulegen. Nach einem Tastendruck beginnt VARLIST sofort mit dem Auflisten der Variablen. Sie sind nach den Nummern der Zeilen sortiert, in denen sie enthalten sind, etwa:

```
100    in$
      a%
```

---

```
110    a$()
```

---

```
120    l%()
```

Ist eine Variable mehrfach in einer Zeile vorhanden – `i=i+1`, `a$=LEFT$(a$,3)` –, wird die Mehrfachnennung unterdrückt, damit der Ausdruck übersichtlich bleibt.

Datenfelder werden durch folgende leere Klammern "()" sichtbar gemacht.

Das Programm verlangt allerdings von Ihnen, daß Sie alle Variablen in Kleinbuchstaben eingeben. Das fordert von Ihnen einen sauberen Programmierstil. Wenn Sie – wie die meisten Programmierer – alle Listings in Kleinbuchstaben eintippen, wandelt der Basic-Interpreter die Schlüsselworte in Großbuchstaben um und beläßt die Variablennamen in Kleinschrift.

## 29.1 Listing VARLIST.BAS

```

100 ' *****
110 ' *
120 ' *          VARLIST - Variablenliste          *
130 ' *
140 ' *****
150 '
160 ' Initialisierung *****
170 MODE 1:INK 0,13:INK 1,0:INK 2,0,3:INK 3,1
180 PAPER 0:PEN 3:BORDER 10:SPEED INK 30,20
190 SYMBOL AFTER 253
200 SYMBOL 253,&66,&0,&78,&C,&7C,&CC,&76,&0
210 SYMBOL 254,&66,&0,&66,&66,&66,&3E,&0
220 SYMBOL 255,&66,&0,&3C,&66,&66,&66,&3C,&0
230 LOCATE 1,1:PRINT STRING$(40,210);
240 LOCATE 1,2:PRINT CHR$(24)+SPACE$(17)+"VARLIST"+SPACE$(16)+CHR$(24)
250 PRINT STRING$(40,210)
260 PEN 1:PRINT:PRINT " Mit ";PEN 2:PRINT "VARLIST";PEN 1
270 PRINT " k"+CHR$(255)+"nnen Sie alle Variablen"
280 PRINT:PRINT " eines Basic-Programms ausgegeben lassen."
290 PRINT:PRINT " Das Programm mu"+CHR$(177)+" im ASCII-Format"
300 PRINT:PRINT " auf Cassette gespeichert sein."
310 PEN 3:PRINT:PRINT:PRINT " Bitte w"+CHR$(253)+"hlen Sie:":PRINT
320 PEN 1:PRINT " 1- Bildschirmausgabe"
330 PRINT:PRINT " 2- Druckerausgabe";
340 PEN 2:PRINT TAB(25);CHR$(143)
350 in$=UPPER$(INKEY$):IF in$="" THEN 350
360 IF in$="1" THEN device=0:GOTO 390
370 IF in$="2" THEN device=8:GOTO 390
380 PRINT CHR$(7);GOTO 350
390 LOCATE 25,20:PRINT " ";
400 PEN 3:PRINT:PRINT:PRINT " Dateiname: ";PEN 1
410 filename$=SPACE$(20):LINE INPUT filename$
420 PEN 1:LOCATE 1,24:PRINT STRING$(40,210);
430 PRINT CHR$(24);SPACE$(6)+"Press PLAY or insert disc ..."+SPACE$(5);
440 WHILE INKEY$="" :WEND
450 PRINT CHR$(24);
460 MODE 2:INK 1,0:INK 0,10:BORDER 10
470 ' Hauptprogramm -----
480 DIM vars.in.line$(255) ' Alle Variablen einer Zeile
490 ' Datei oeffnen und Programmzeilen einlesen *****
500 IF LEFT$(filename$,1)<>"!" THEN filename$="!" + filename$
510 OPENIN filename$
520 LINE INPUT #9,a$
530 FOR i=0 TO 255:vars.in.line$(i)="" :NEXT i
540 linenumber=VAL(a$)
550 vilont=0:quoteflag=0
560 ' Zeile byteweise zerlegen und analysieren *****
570 FOR i=1 TO LEN(a$)
580 seg$=MID$(a$,i,1):seg2$=MID$(a$,i+1,2)
590 IF seg$=CHR$(34) THEN quoteflag=quoteflag XOR 1
600 IF quoteflag THEN 640
610 IF seg$="!" THEN 650 ' Abkuerzung fuer REM: Rest der Zeile ueberlesen
620 IF seg$+seg2$="REM" THEN 650 ' REM: Rest der Zeile nicht beachten
630 IF ASC(seg$)>96 AND ASC(seg$)<123 THEN 720 ' VARIABLE GEFUNDEN!
640 NEXT i

```

```

650 IF vilcnt=0 THEN 700 ' Keine Variable in der Zeile
660 ' Die Variablen einer Zeile ausgeben *****
670 PRINT #device, USING "####: "; linenumber;
680 FOR x=1 TO vilcnt:PRINT #device, TAB(20); vars.in.line$(x):NEXT x
690 PRINT #device, STRING$(78,95):PRINT #device
700 IF NOT EOF THEN 520 ELSE END
710 ' Variable gefunden: Namen zusammenstellen *****
720 WHILE i<=LEN(a$)
730 i=i+1:var$=MID$(a$,i,1)
740 IF var$="" THEN 800 ELSE var=ASC(var$)
750 IF var>96 AND var<123 THEN 850
760 IF var>48 AND var<57 THEN 850
770 IF var$="!" OR var$=" $" OR var$="%" OR var$="." THEN 850
780 IF var$="(" THEN seg$=seg$+"()" ' Feldelemente, z.B. a$()
790 ' Variablenliste absuchen *****
800 FOR x=1 TO vilcnt
801 IF seg$=vars.in.line$(x) THEN 640
810 NEXT x
820 ' Wenn noch nicht in der Variablenliste, dann aufnehmen *****
830 vilcnt=vilcnt+1:vars.in.line$(vilcnt)=seg$:GOTO 640
840 ' Variablenende nicht erreicht: weiteren Buchstaben anhaengen *****
850 seg$=seg$+var$
860 WEND
870 IF NOT EOF THEN 520
880 CLOSEIN:END
890 ' -----

```

Für Interessierte der Grobaufbau des Programms:

- Zeile 100 bis 150: Kommentarzeilen
- Zeile 160 bis 300: Ausgabe des Titelbilds
- Zeile 310 bis 450: Hier fordert das Programm den Dateinamen an und will wissen, ob die Ausgabe an den Drucker oder an den Bildschirm gehen soll. Sie werden aufgefordert, den Datenträger einzulegen, auf dem sich die Datei befindet, und danach eine Taste zu drücken.
- Zeile 460 bis 890: Hauptprogramm
- \* 490 bis 550: Datei wird zum Lesen geöffnet, und alle Variablen werden initialisiert.
- \* 560 bis 650: Die Zeile wird byteweise eingelesen. Das Programm prüft, ob ein Anführungszeichen oder ein REM-Befehl zu finden ist.
- \* 660 bis 700: VARLIST gibt nach der Rückkehr aus dem Unterprogramm ab Zeile 710 alle Variablen einer Zeile aus. Die Variable "device" enthält den Wert 0 für die Bildschirmausgabe (PRINT #0) und 8 für die Ausgabe auf dem Drucker (PRINT #8).
- \* 710 bis 780: Hier stellt der Computer aus den einzelnen Bytes einen Variablennamen zusammen.

\* 790 bis 830: VARLIST prüft, ob die Variable schon in der Zeile gefunden ist. Wenn nicht, dann wird sie in die Liste im Array vars.in.line\$() aufgenommen.

Und hier zur Einstimmung die Variablenliste von VARLIST selbst:

350:	in\$
360:	in\$ device
370:	in\$ device
410:	filename\$
480:	vars.in.line\$()
500:	filename\$
510:	filename\$
520:	a\$
530:	i vars.in.line\$()
540:	linenumber a\$
550:	vilont quoteflag
570:	i a\$
580:	seg\$ a\$ i seg2\$
590:	seg\$ quoteflag
600:	quoteflag
610:	seg\$
620:	seg\$ seg2\$
630:	seg\$
640:	i

650:	vilont
670:	device linenumber
680:	x vilont device vars.in.line\$()
690:	device
720:	i a\$
730:	i var\$ a\$
740:	var\$ var
750:	var
760:	var
770:	var\$
780:	var\$ seg\$
800:	x vilont
801:	seg\$ vars.in.line\$() x
810:	x
830:	vilont vars.in.line\$() seg\$
850:	seg\$ var\$

### 30. Benutzertip MAXAM – mal anders

Der MAXAM-Assembler ist – besonders in der ROM-Version – ein äußerst leistungsfähiger Maschinensprache-Monitor und Assembler. Doch es gibt auch Probleme ...

Der Aufruf des ROM-Moduls ist nämlich von Basic aus ausschließlich über RSX-Kommandos möglich. !MAXAM und !M heißen die beiden (identischen) Befehle, die den Computer dazu veranlassen, in das Hauptmenü von MAXAM zu springen.

Es gibt aber immer wieder Situationen, in denen der Aufruf des MAXAM-Moduls auf diese Art nicht mehr funktioniert: Wenn irgend etwas bei der Definition neuer RSX-Befehle schiefgegangen ist, verstellen sich die internen Zeiger des Betriebssystems auf alle vorher definierten RSXen.

Das passiert vor allem dann, wenn Sie versehentlich eine Initialisierungs-Routine für einen RSX-Befehl zweimal aufrufen und die vier Bytes Hilfsspeicher, die Sie dem Betriebssystem zur Verfügung stellen müssen, unverändert an der gleichen Adresse stehen lassen. Das Betriebssystem reiht anhand dieser vier Bytes die definierte RSX-Erweiterung in die Kette der anderen RSX-Befehle ein.

Beim zweimaligen Aufruf an derselben Adresse zeigen diese vier Bytes nun auf sich selbst.

Versucht später der Basic-Interpreter, einen RSX-Befehl auszuführen, ruft er die Betriebssystem-Routine KL FIND COMMAND (Adresse &BCD4) auf, die anhand dieser verketteten Liste nach dem passenden RSX-Namen sucht.

Sie bleibt aber an der Stelle hängen, an der der Zeiger auf sich selbst zeigt, und läuft in eine Endlosschleife.

Der nun fällige Reset löscht alle Daten und Programme.

Gerade in dieser Situation wäre es aber interessant, mit einem Maschinensprache-Monitor wie MAXAM den Speicher analysieren zu können, um dem aufgetauchten Fehler auf die Schliche zu kommen!

Die RSX-Befehle !M und !MAXAM scheiden als Aufrufmöglichkeit aus, da sie den Computer entweder zum Aussteigen bringen oder den Basic-Interpreter nur mit einem lapidaren UNKNOWN COMMAND antworten lassen.

Folglich muß ein anderer Weg gefunden werden.

Am einfachsten ist der Aufruf mittels eines kleinen Maschinenprogramms.

Das erste Listing zeigt den Quellcode:

```
;
; MAXAM-ROM über CALL aufrufen
```

```

; -----
                                ORG      &0160

KL_ROM_SELECT EQU      &B90F ; KL ROM SELECT
MAXAM_ENTRY   EQU      &D469 ; Einsprungsadresse

CALL_MAXAM    LD        C,5
              CALL      KL_ROM_SELECT
              JP        MAXAM_ENTRY
              END

```

Das Maschinencode-Programm blendet das MAXAM-ROMs in den Adreßbereich &C000 bis &FFFF mit LD C,5 und CALL &B90F (KL ROM SELECT) ein. Der Wert 5 korrespondiert mit der ROM-Nummer von MAXAM. Der Computer springt dann direkt an die Adresse &D469. Das ist die Standard-Einsprungstelle die auch von der RSX !MAXAM verwendet wird.

Möglicherweise benutzen zukünftige ROM-Versionen andere Einsprünge. Für den Fall, daß der Aufruf anhand der Adresse &D469 bei Ihnen nicht klappt, können Sie es stattdessen mit &C009 versuchen: Dort ist der Eintrag in die RSX-Sprungtabelle zu finden. "MAXAM" steht in der Tabelle an der zweiten Position.

Das zweite Listing zeigt ein kurzes Basic-Programm, das das Maschinenprogramm an die Adressen &0160 bis &0167 POKet. Dieser Speicherbereich liegt zwischen dem Eingabepuffer für die Programmzeilen (&0040 bis &015F) und dem Basic-Programmspeicher (ab &0170).

Diese Bytes werden von keiner ROM-Routine benutzt und sind damit geschützt. Der Aufruf der Maschinenroutine kann mit CALL &0160 erfolgen. Natürlich lassen sich diese Befehle auch im Direktmodus eingeben.

#### Listing: MAXCALL.BAS

```

100 ' MAXCALL: MAXAM-Aufruf per
110 '       POKE und CALL
120 '
130 ' *****
140 '
150 POKE &160,14:POKE &161,5
160 POKE &162,&CD:POKE &163,15
170 POKE &164,&B9:POKE &165,&C3
180 POKE &166,&69:POKE &167,&D4
190 CALL &160

```



## 31. Anhang 1 – Firmware-Einsprünge

Diese Aufstellung wird für den "Analysierenden Disassemblier" in Abschnitt 10 benötigt.

```

1000 '245 FIRMWARE-ADRESSEN
1010 '
1020 'LOW-ADRESSES
1030 DATA 0000,RST0 =RESET ENTRY,0000,RST1 =LOW JUMP,000B,KL LOW PCHL
1040 DATA 000E,PCBC INSTRUCTION,0010,RST2 =SIDE CALL,0013,KL SIDE PCHL
1050 DATA 0016,PCDE INSTRUCTION,0018,RST3 =FAR CALL,001B,KL FAR PCHL
1060 DATA 001E,PCHL INSTRUCTION,0020,RST4 =RAM LAM,0023,KL FAR ICALL
1070 DATA 0028,RST5 =FIRM JUMP,0030,RST6 =USER RESTART
1080 DATA 0038,RST7 =INTERUPT ENTRY,003B,EXT INTERRUPT
1090 '
1100 'FIRMJUMPS
1110 DATA B900,KL U ROM ENABLE,B903,KL U ROM DISABLE,B906,KL L ROM ENABLE
1120 DATA B909,KL L ROM DISABLE,B90C,KL L ROM RESTORE,B90F,KL L ROM SELECT
1130 DATA B912,KL CURR SELECTION,B915,KL PROBE ROM,B918,KL ROM DESELECT
1140 DATA B91B,KL LDIR,B91E,KL LDDR,B921,KL POLL SYNCHRONOUS
1150 '
1160 'KM=Key Manager
1170 DATA BB00,KM INITIALIZE,BB03,KM RESET,BB06,KM WAIT CHAR
1180 DATA BB09,KM READ CHAR,BB0C,KM CHAR RETURN,BB0F,KM SET EXPAND
1190 DATA BB12,KM GET EXPAND,BB15,KM EXP BUFFER,BB18,KM WAIT KEY
1200 DATA BB1B,KM READ KEY,BB1E,KM TEST KEY,BB21,KM GET STATE
1210 DATA BB24,KM GET JOYSTICK,BB27,KM SET TRANSLATE
1220 DATA BB2A,KM GET TRANSLATE,BB2D,KM SET SHIFT,BB30,KM GET SHIFT
1230 DATA BB33,KM SET CONTROL,BB36,KM GET CONTROL,BB39,KM SET REPEAT
1240 DATA BB3C,KM GET REPEAT,BB3F,KM SET DELAY,BB42,KM GET DELAY
1250 DATA BB45,KM ARM BREAKS,BB48,KM DISARM BREAKS,BB4B,KM BREAK EVENT
1260 '
1270 'TXT=Text VDU
1280 DATA BB4E,TXT INITIALISE
1290 DATA BB51,TXT RESET,BB54,TXT VDU ENABLE,BB57,TXT VDU DISABLE
1300 DATA BB5A,TXT OUTPUT,BB5D,TXT WR CHAR,BB60,TXT RD CHAR
1310 DATA BB63,TXT SET GRAPHIC,BB66,TXT WIN ENABLE,BB69,TXT GET WINDOW
1320 DATA BB6C,TXT CLEAR WINDOW,BB6F,TXT SET COLUMN,BB72,TXT SET ROW
1330 DATA BB75,TXT SET CURSOR,BB78,TXT GET CURSOR,BB7B,TXT CUR ENABLE
1340 DATA BB7E,TXT CUR DISABLE,BB81,TXT CUR ON,BB84,TXT CUR OFF
1350 DATA BB87,TXT VALIDATE,BB8A,TXT PLACE CURSOR,BB8D,TXT REMOVE CURSOR
1360 DATA BB90,TXT SET PEN,BB9E,TXT GET PEN,BB96,TXT SET PAPER
1370 DATA BB99,TXT GET PAPER,BB9C,TXT INVERSE,BB9F,TXT SET BACK
1380 DATA BBA2,TXT GET BACK,BBA5,TXT GET MATRIX,BBA8,TXT SET MATRIX
1390 DATA BBAB,TXT SET M TABLE,BBAE,TXT GET M TABLE,BBB1,TXT GET CONTROLS
1400 DATA BBB4,TXT STR SELECT,BBB7,TXT SWAP STREAMS
1410 '

```

```

1420 'GRA=Grafik VDU
1430 DATA BBBA, GRA INITIALISE
1440 DATA BBBD, GRA RESET, BBC0, GRA MOVE ABSOLUTE, BBC3, GRA MOVE RELATIVE
1450 DATA BBC6, GRA ASK CURSOR, BBC9, GRA SET ORIGIN, BBCC, GRA GET ORIGIN
1460 DATA BBCF, GRA WIN WIDTH, BBD2, GRA WIN HEIGHT, BBD5, GRA GET W WIDTH
1470 DATA BBD8, GRA GET W HEIGHT, BBDB, GRA CLEAR WINDOW, BBDE, GRA SET PEN
1480 DATA BBE1, GRA GET PEN, BBE4, GRA SET PAPER, BBE7, GRA GET PAPER
1490 DATA BBEA, GRA PLOT ABSOLUTE, BBED, GRA PLOT RELATIVE, BBF0, GRA TEST ABSOLUTE
1500 DATA BBF3, GRA TEST RELATIVE, BBF6, GRA LINE ABSOLUTE, BBF9, GRA LINE RELATIVE
1510 DATA BBFC, GRA WR CHAR
1520 '
1530 'SCR=BILDSCHIRMPAKET
1540 DATA BBFF, SCR INITIALISE, BC02, SCR RESET, BC05, SCR SET OFFSET
1550 DATA BC08, SCR SET BASE, BC0B, SCR GET LOCATION, BC0E, SCR SET MODE
1560 DATA BC11, SCR GET MODE, BC14, SCR CLEAR, BC17, SCR CHAR LIMITS
1570 DATA BC1A, SCR CHAR POSITION, BC1D, SCR DOT POSITION, BC20, SCR NEXT BYTE
1580 DATA BC23, SCR PREV BYTE, BC26, SCR NEXT LINE, BC29, SCR PREV LINE
1590 DATA BC2C, SCR INK ENCODE, BC2F, SCR INK DECODE, BC32, SCR SET INK
1600 DATA BC35, SCR GET INK, BC38, SCR SET BORDER, BC3B, SCR GET BORDER
1610 DATA BC3E, SCR SET FLASHING, BC41, SCR GET FLASHING, BC44, SCR FILL BOX
1620 DATA BC47, SCR FLOOD BOX, BC4A, SCR CHAR INVERT, BC4D, SCR HW ROLL
1630 DATA BC50, SCR HW ROLL, BC53, SCR UNPACK, BC56, SCR REPACK, BC59, SCR ACCESS
1640 DATA BC5C, SCR PIXELS, BC5F, SCR HORIZONTAL, BC62, SCR VERTICAL
1650 '
1660 'cas=Kassettenverwaltung (und Disk!)
1670 DATA BC65, CAS INITIALISE
1680 DATA BC68, CAS SET SPEED, BC6B, CAS NOISY
1690 DATA BC6E, CAS START MOTOR, BC71, CAS STOP MOTOR, BC74, CAS RESTORE MOTOR
1700 DATA BC77, CAS IN OPEN, BC7A, CAS IN CLOSE, BC7D, CAS IN ABANDON
1710 DATA BC80, CAS IN CHAR, BC83, CAS IN DIRECT
1720 DATA BC86, CAS RETURN, BC89, CAS TEST EOF, BC8C, CAS OUT OPEN
1730 DATA BC8F, CAS OUT CLOSE, BC92, CAS OUT ABANDON, BC95, CAS OUT CHAR
1740 DATA BC98, CAS OUT DIRECT, BC9B, CAS CATALOG, BC9E, CAS WRITE
1750 DATA BCA1, CAS READ, BCA4, CAS CHECK
1760 '
1770 'Sound=Tonverwaltung
1780 DATA BCA7, SOUND RESET, BCAA, SOUND QUEUE
1790 DATA BCAD, SOUND CHECK, BCB0, SOUND ARM EVENT, BCB3, SOUND RELEASE
1800 DATA BCB6, SOUND HOLD, BCB9, SOUND CONTINUE, BCBF, SOUND AMPL ENVELOPE
1810 DATA BCBF, SOUND TONE ENVELOPE, BCC2, SOUND A ADRESS, BCC5, SOUND T ADRESS
1820 '
1830 'KL=Betriebssystemkern
1840 DATA BCC8, KL CHOKE OFF, BCCB, KL ROM WALK, BCCE, KL INIT BACK, BCD1, KL LOG EXT
1850 DATA BCD4, KL FIND COMMAND, BCD7, KL NEW FRAME FLY, BCDA, KL ADD FRAME FLY
1860 DATA BCDD, KL DEL FRAME FLY, BCE0, KL NEW FAST TICKER, BCE3, KL ADD FAST TICKER
1870 DATA BCE6, KL DEL FAST TICKER, BCE9, KL ADD TICKER, BCEC, KL DEL TICKER
1880 DATA BCEF, KL INIT EVENT, BCF2, KL EVENT, BCF5, KL SYNC RESET
1890 DATA BCF8, KL DEL SYNCHRONOUS, BCFB, KL NEXT SYNC, BCFF, KL DO SYNC
1900 DATA BD01, KL DONE SYNC, BD04, KL EVENT DISABLE, BD07, KL EVENT ENABLE
1910 DATA BD0A, KL DISARM EVENT, BD0D, KL TIME PLEASE, BD10, KL TIME SET
1920 '
1930 'MC=Maschinenpaket
1940 DATA BD13, MC BOOT PROGRAMM, BD16, MC START PROGRAMM
1950 DATA BD16, MC START PROGRAMM, BD19, MC WAIT FLYBACK, BD1C, MC SET MODE
1960 DATA BD1F, MC SCREEN OFFSET, BD22, MC CLEAR INKS, BD25, MC SET INKS
1970 DATA BD28, MC RESET PRINTER, BD2B, MC PRINT CHAR, BD2E, MC BUSY PRINTER
1980 DATA BD31, MC SEND PRINTER, BD34, MC SOUND REGISTER, BD37, JUMP RESTORE

```

1990 '  
2000 'INDIRECTIONS  
2010 DATA BDCD, TXT DRAW CURSOR, BDD0, TXT UNDRAW CURSOR, BDD3, TXT WRITE CHAR  
2020 DATA BDD6, TXT UNWRITE, BDD9, TXT OUT ACTION, BDDC, GRA PLOT, BDDF, GRA TEST  
2030 DATA BDE2, GRA LINE, BDE5, SCR READ, BDE8, SCR WRITE, BDEB, SCR MODE CLEAR  
2040 DATA BDEE, KM TEST BREAK, BDF1, MC WAIT PRINTER  
2050 '  
2060 'Neue Einspruenge fuer CPC 664/6128  
2070 DATA BD3A, KM SET LOCKS, BD3D, KM FLUSH, BD40, TXT ASK STATE, BD43, GRA DEFAULT  
2080 DATA BD46, GRA SET BACK, BD49, GRA SET FIRST, BD4C, GRA SET LINE MASK  
2090 DATA BD4F, GRA FROM USER, BD52, GRA FILL, BD55, SCR SET POSITION  
2100 DATA BD58, MC PRINT TRANSLATION, BDF4, KM SCAN KEYS, B92A, KL SCAN NEEDED

## 32. Anhang 2 – Die System-Adressen

Diese Aufstellung kann als Umsetzungstabelle beim Umschreiben von Programmen für die verschiedenen CPC-Typen dienen.

Um eine Tabelle zu erstellen, kann das auf die Aufstellung folgende Programm benutzt werden. Zwischenadressen müssen berechnet werden!

Außerdem wird diese Aufstellung für den

"Analysierenden Disassembler"

in Abschnitt 10, benötigt.

```

1000 ' *****
1010 ' *** 409 Systemadressen 6128/664/464 ***
1020 ' *****
1030 '
1040 DATA A67C,,AB80,HINEM DEFAULT
1050 DATA ABFF,,????,DEFAULT UPPER ROM BOUNDARY -1
1060 DATA AC00,,AC00,????
1070 DATA AC01,,AC1C,FLAG FOR AUTO
1080 DATA AC02,,AC1D,NEW LINE #
1090 DATA AC04,,AC1F,STEP FOR AUTO
1100 DATA AC06,,AC21,OUTPUT CHANNEL #
1110 DATA AC07,,AC22,INPUT CHANNEL #
1120 DATA AC08,,AC23,POS PRINTER
1130 DATA AC09,,AC24,WIDTH FOR PRINTER
1140 DATA AC0A,,AC25,POS (TAPE) # OF CHARS
1150 DATA AC0B,,AC26,????
1160 DATA AC0C,,AC26,FLAG USED BY FOR ??
1170 DATA AC0D,,AC27,FAC USED BY FOR ??
1180 DATA AC11,,AC29,????
1190 DATA AC12,,AC2C,USED BY FOR ??
1200 DATA AC13,,AC2D,????
1210 DATA AC14,,AC2E,USED BY WHILE/WEND
1220 DATA AC16,,AC30,USED BY ON
1230 DATA AC17,,AC31,USED BY ON
1240 DATA AC18,,AC32,????
1250 DATA AC19,,AC33,????
1260 DATA AC1A,,AC34,LINE # FOR ON BREAK GOSUB (1)
1270 DATA AC1B,,AC35,LINE # FOR ON BREAK GOSUB (2)

```

```

1280 DATA AC1C,,AC36,BASIC PC ON BREAK
1290 DATA AC1E,,AC38,SOUND CHANNEL 1 (BIT 0)
1300 DATA AC2A,,AC44,SOUND CHANNEL 2 (BIT 1)
1310 DATA AC36,,AC50,SOUND CHANNEL 3 (BIT 2)
1320 DATA AC42,,AC5C,TIMER BLOCK #0
1330 DATA AC48,,AC62,????
1340 DATA AC8A,,ACA4,EDIT BUFFER
1350 DATA AD8C,,ADA6,ERROR ADDRESS
1360 DATA AD8E,,ADA8,PC ON ERROR BREAK
1370 DATA AD90,,ADAA,LAST BASIC ERROR #
1380 DATA AD92,,ADAB,CONTINUE POINTER
1390 DATA AD94,,ADAD,BASIC PC ON STOP OR END
1400 DATA AD96,,ADAF,ON ERROR ADDRESS
1410 DATA AD98,,ADB1,FLAG ON ERROR
1420 DATA AD99,,ADB2,SOUND CHAN-STAT
1430 DATA AD9A,,ADB3,SOUND VOL-ENV
1440 DATA AD9B,,ADB4,SOUND TON-ENV
1450 DATA AD9C,,ADB5,SOUND PERIOD
1460 DATA AD9E,,ADB7,SOUND NOISE
1470 DATA AD9F,,ADB8,SOUND VOLUME
1480 DATA ADA0,,ADB9,SOUND TIME
1490 DATA ADA2,,ADBB,ENVELOPE TABLE ADDRESS
1500 DATA ADA3,,ADBC,SOUND ENVELOPE ADDRESS ??
1510 DATA ADB2,,ADCB,FAC USED BY POWER
1520 DATA ADB7,,ADD0,?????
1530 DATA ADEB,,AE04,?????
1540 DATA ADED,,AE06,?????
1550 DATA AE0D,,AE26,?????
1560 DATA AE0E,,AE27,POINTER TO BASIC STACK
1570 DATA AE10,,AE29,POINTER TO FN SUBPROG (1)
1580 DATA AE12,,AE2B,POINTER TO FN SUBPROG (2)
1590 DATA AE14,,AE2D,SAVE FOR SEMIKOLON ON PRINT
1600 DATA AE15,,AE2E,LAST DATA LINE #
1610 DATA AE17,,AE30,POINTER TO NEXT DATA
1620 DATA AE19,,AE32,TEMP STORAGE BASIC STACK POINTER
1630 DATA AE1B,,AE34,PROGRAMM COUNTER ON RUN
1640 DATA AE1D,,AE36,BASIC PROG COUNTER PC
1650 DATA AE1F,,AE38,FLAG TRON/TROFF
1660 DATA AE20,,AE39,FLAG USED ASSEMBLING A BASIC LINE
1670 DATA AE21,,AE3A,BASIC PROG LINE FORMAT
1680 DATA AE22,,AE3B,USED BY DELETE LINE #/LOWER ADR
1690 DATA AE24,,AE3D,USED BY DELETE LINE #/UPPER ADR
1700 DATA AE26,,AE3F,LOAD POINTER WHILE LOAD
1710 DATA AE28,,AE41,LOAD MERGE FLAG
1720 DATA AE29,,AE42,LOAD/CHAIN FLAG
1730 DATA AE2A,,AE43,USED BY LOAD CHAIN
1740 DATA AE2C,,AE45,FLAG FILE READ PROTECTED
1750 DATA AE2D,,AE46,NUMBER EDIT BUFFER (1)
1760 DATA AE3A,,AE53,NUMBER EDIT BUFFER (2)
1770 DATA AE3F,,AE58,NUMBER EDIT BUFFER (3)
1780 DATA AE4E,,AE67,NUMBER EDIT BUFFER INDEX ??
1790 DATA AE4F,,AE68,NUMBER EDIT BUFFER (4)
1800 DATA AE50,,AE69,NUMBER EDIT BUFFER INDEX ??
1810 DATA AE52,,AE6E,TEMP STORE FOR CHAR
1820 DATA AE53,,AE6F,?????

```

```

1830 DATA AE54,,AE70,NUMBER EDIT BUFFER INDEX
1840 DATA AE55,,AE72,ADDRESS OF CALLED ROUTINE
1850 DATA AE57,,AE74,ROM SELECTION ON CALL
1860 DATA AE58,,AE75,SAVE HL ON CALL
1870 DATA AE5A,,AE77,SAVE SP ON CALL
1880 DATA AE5C,,AE79,ZONE FOR TAB
1890 DATA AE5D,,AE7A,FLAG FOR PRINT USING
1900 DATA AE5E,,AE7B,HIMEM FOR BASIC POINTER
1910 DATA AE60,,AE7D,HIMEM FOR SYMBOL AFTER POINTER
1920 DATA AE62,,AE7F,LOW MEMORY BOUNDARY POINTER
1930 DATA AE64,,AE81,START OF BASIC-PRG -1
1940 DATA AE66,,AE83,END OF BASIC-PRG POINTER
1950 DATA AE68,,AE85,START VAR TABLE POINTER
1960 DATA AE6A,,AE87,START DIM'D VAR POINTER
1970 DATA AE6C,,AE89,UPPER END DIM POINTER
1980 DATA ????,,AE8B,START OF BASIC STACK
1990 '
2000 DATA ????,,B000,?????
2010 DATA AE6E,,B091,TAPE BUFFER FLAG ??
2020 DATA B06F,,B08B,BASIC STACK-POINTER
2030 DATA B071,,B08D,LOW END OF USED STRING SPACE POINTER
2040 DATA B073,,B08F,UPPER BOUND FOR STRING SPACE POINTER
2050 DATA B075,,B091,TAPE BUFFER FLAG
2060 DATA B076,,B092,POINTER TO TAPE BUFFER/LOWER END
2070 DATA ????,,B094,POINTER TO TAPE PUFFER/UPPER END
2080 DATA ????,,B096,HIMEM FOR SYMBOL AFTER
2090 DATA B07A,,B098,USED BY GARBAGE COLLECT
2100 DATA ????,,B09A,POINTER TO START OF STRING STACK
2110 DATA ????,,B09C,STRING STACK
2120 DATA ????,,B0BA,TEMPORARY STRING DESCRIPTOR
2130 DATA ????,,B0BD,USED ON GARBAGE COLLECT
2140 DATA ????,,B0BF,SAVE ON GARBAGE COLLECT
2150 DATA B09F,,B0C1,VARTYPE
2160 DATA B0A0,,B0C2,FLDATING POINT ACU/FAC
2170 '
2180 DATA B02D,,B100,KL INTERRUPT SERVICE QUEUE
2190 DATA B02E,,B101,KL INTERRUPT PENDING QUEUE
2200 DATA B02F,,B102,KL INTERRUPT SERVICE CHAIN
2210 DATA B031,,B104,KL INTERRUPT SERVICE CLASS
2220 DATA B032,,B105,KL SAVE SP ON INTERR SERVICE
2230 DATA ????,,B107,KL PRIVATE INTERRUPT STACK
2240 DATA B0B4,,B187,KL TIME BYTE 0+1
2250 DATA B0B6,,B189,KL TIME BYTE 2+3
2260 DATA B0B8,,B18B,KL TIME BYTE 4
2270 DATA B0B9,,B18C,KL FRAME FLY LIST POINTER
2280 DATA B0BB,,B18E,KL FAST TICKER LIST POINTER
2290 DATA B0BD,,B190,KL POINTER TO TICK LIST
2300 DATA B0BF,,B192,KL SLOW TICKER COUNT
2310 DATA B0C0,,B193,KL SYNC EVENT QUEUE
2320 DATA B0C1,,B194,KL SYNC EVENT QUEUE +1
2330 DATA B0C2,,B195,KL EVENT CLASS
2340 DATA B0C3,,B196,KL TEMP STORE EXT COMMAND NAME
2350 DATA B0D3,,B1A6,KL RSX QUEUE
2360 '
2370 DATA B0D6,B0D5,B1A8,KL ROM SELECT ADDRESS

```

```

2380 DATA B8D7,B8D6,B1A9,KL ROM START
2390 DATA B8D9,B8D8,B1AB,KL ROM STATE TO CALL
2400 DATA ????,,B1AC,KL USED FOR RST 3/FAR CALL
2410 DATA B8DA,B8D9,B1AA,START OF ROM HIGH BYTE (C0)
2420 DATA ????,,B1BA,KL USED FOR RST 3 +14
2430 '
2440 DATA B7C3,,B1C8,SCR SCREEN MODE
2450 DATA B7C4,,B1C9,SCR SCREEN START
2460 DATA B7C5,,B1CA,SCR OFFSET TO SCREEN START
2470 DATA B7C6,,B1CB,SCR BASE OF RAM FOR SCREEN
2480 DATA B7C7,,B1CC,SCR PIXELS WRITE
2490 DATA B7C8,,B1CD,SCR PIXELS WRITE SET JUMP ADDRESS
2500 DATA ????,,B1CF,SCR CURRENT PIXEL BIT MAP
2510 DATA B7D2,,B1D7,SCR TIME FLASHING PERIOD 1
2520 DATA B7D3,,B1D8,SCR TIME FLASHING PERIOD 2
2530 DATA B7D4,,B1D9,SCR COLOR TABLE FLASH 1 (BORDER)
2540 DATA B7E5,,B1EA,SCR COLOR TABLE FLASH 2 (BORDER)
2550 DATA B7F6,,B1FB,SCR FLAG WHICH FLASH PERIOD (1 OR 2)
2560 DATA B7F7,,B1FC,SCR FLAG
2570 DATA B7F8,,B1FD,TIME COUNT/CURRENT FLASH PERIOD
2580 DATA B7F9,,B1FE,SCR FRAME FLY LIST
2590 DATA B6B5,,B20C,TXT CURRENT TEXT STREAM
2600 DATA B6B6,,B20D,TXT TABLE (STREAM-PARAMETER 8*15)
2610 '
2620 DATA ????,,B21C,TXT PARAM WINDOW #1
2630 DATA ????,,B22B,TXT PARAM WINDOW #2
2640 DATA ????,,B23A,TXT PARAM WINDOW #3
2650 DATA ????,,B249,TXT PARAM WINDOW #4
2660 DATA ????,,B258,TXT PARAM WINDOW #5
2670 DATA ????,,B267,TXT PARAM WINDOW #6
2680 DATA ????,,B276,TXT PARAM WINDOW #7
2690 '
2700 DATA B726,,B285,TXT CURSOR COLUMN/ROW
2710 DATA B728,,B287,TXT WINDOW FLAG
2720 DATA B729,,B288,TXT ROW-WINDOW LEFT UPPER CORNER
2730 DATA B72A,,B289,TXT COLUMN-WINDOW LEFT UPPER CORNER
2740 DATA B72B,,B28A,TXT ROW WINDOW RIGHT BOTTOM CORNER
2750 DATA B72C,,B28B,TXT COLUMN-WINDOW RIGHT BOTTOM CORNER
2760 DATA B72D,,B28C,TXT ROLL COUNT
2770 DATA B72E,,B28D,TXT CURSOR ENABLE FLAG (USER)
2780 DATA ????,,B28E,TXT FLAG VDU ENABLE
2790 DATA B72F,,B28F,TXT PEN INK
2800 DATA B730,,B290,TXT PAPER INK
2810 DATA B731,,B291,TXT ADDRESS OF BACK-/FOREGROUND ROUTINE
2820 DATA B733,,B293,TXT FLAG GRAPHIC CHAR WRITE
2830 DATA B734,,B294,TXT FIRST CHAR OF USER MATRIX TABLE
2840 DATA B735,,B295,TXT FLAG USER MATRIX TABLE
2850 DATA B736,,B296,TXT START OF USER MATRIX TABLE
2860 DATA B738,,B298,TXT BUFFER FOR UNPACKED CHAR MATRIX
2870 DATA B758,,B2B8,TXT CONTROL CODE PUFFER INDEX
2880 DATA B759,,B2B9,TXT CONTROL CODE BUFFER
2890 DATA B763,,B2C3,CONTROL CODE TABLE
2900 '
2910 DATA ????,,B323,?????
2920 '

```

```

2930 DATA B693,,B328,GRA USER ORIGIN X
2940 DATA B695,,B32A,GRA USER ORIGIN Y
2950 DATA B697,,B32C,GRA CURSOR X
2960 DATA B699,,B32E,GRA CURSOR Y
2970 DATA B69B,,B330,GRA WINDOW WIDTH X-LEFT
2980 DATA B69D,,B332,GRA WINDOW WIDTH X-RIGHT
2990 DATA B69F,,B334,GRA WINDOW HEIGHT Y-TOP
3000 DATA B6A1,,B336,GRA WINDOW HEIGHT Y-BOTTOM
3010 DATA B6A3,,B338,GRA PEN INK
3020 DATA B6A4,,B339,GRA PAPER INK
3030 DATA B6A5,,B33A,GRA TEMP STORE 1
3040 DATA B6A7,,B33C,GRA TEMP STORE 2
3050 DATA B6A9,,B33E,GRA TEMP STORE 3
3060 '
3070 DATA B6AA,,????,?????
3080 '
3090 DATA B6AB,,B340,GRA TEMP STORE 4
3100 '
3110 DATA B6AC,,????,?????
3120 '
3130 DATA B6AD,,B342,GRA TEMP STORE X ON DRAW
3140 DATA B6AE,,????,?????
3150 DATA B6AF,,B344,GRA TEMP STORE Y ON DRAW
3160 DATA B6B0,,????,?????
3170 DATA B6B1,,B346,GRA TEMP FLAG
3180 DATA B6B2,,????,?????
3190 DATA B6B3,,B348,?????
3200 DATA B6B4,,B349,?????
3210 '
3220 DATA B496,,B34C,KM KEY NORMAL ENTRY
3230 DATA B4E6,,B39C,KM KEY SHIFT ENTRY
3240 DATA B536,,B53C,KM KEY CONTROL ENTRY
3250 DATA B586,,B43C,KM KEY REPEAT MAP
3260 DATA B590,,B446,KM KEY EXP BUFFER
3270 DATA B5D6,,????,?????
3280 '
3290 DATA B628,,B4DE,KM EXPANSION STRING FLAG/COUNT
3300 DATA B629,,B4DF,KM EXP BUFFER FLAG
3310 DATA B62A,,B4E0,KM KEYBOARD PUT BACK CHAR
3320 DATA B62B,,B4E1,KM POINTER TO F-KEY EXP BUFFER
3330 DATA B62D,,B4E3,KM POINTER TO END OF EXP B+1
3340 DATA B62F,,B4E5,KM EXP PUFFER POINTER
3350 DATA B630,,B4E6,EXP-BUFFER POINTER HI-BYTE
3360 DATA B631,,B4E7,KM CAPS LOCK STATE
3370 DATA B632,,B4E8,KM SHIFT LOCK STATE
3380 DATA B633,,B4E9,KM KEY REPEAT SPEED
3390 DATA B634,,B4EA,KM KEY STARTUP DELAY
3400 DATA B635,,B4EB,KM KEY STATE MAP
3410 DATA B637,,B4ED,CONTAINS CTRL AND SHIFT BITS
3420 DATA B63B,,B4F1,JOYSTICK 2
3430 DATA B63D,,B4F3,BIT 2 = BREAK KEY
3440 DATA B63E,,B4F4,JOYSTICK 1
3450 DATA B63F,,B4F5,KM KEY CHANGE STATE MAP
3460 DATA B649,,B4FF,KM KEY LAST CYCLE STATE MAP
3470 DATA B653,,B509,KM TIME COUNT FOR REPEAT SPEED

```



```

3480 DATA B654,,B50A,?????
3490 DATA B655,,B50B,?????
3500 DATA ????,B50C,KM BREAK ENABLE FLAG
3510 DATA ????,B50D,KM EVENT BLOCK BREAK
3520 DATA B686,,B53C,?????
3530 DATA B688,,B53E,?????
3540 DATA B68A,,B540,?????
3550 DATA B68B,,B541,KM TRANSLATE NORMAL ENTRY POINTER
3560 DATA B68D,,B543,KM TRANSLATE SHIFT ENTRY POINTER
3570 DATA B68F,,B545,KM TRANSLATE CONTROL ENTRY POINTER
3580 DATA B691,,B547,KM REPEAT KEY POINTER TO TABLE
3590 '
3600 DATA B1EE,,B550,SOUND FLAG ?? NICHT EINDEUTIG
3610 DATA B1ED,,B551,SOUND SAVE FOR AKTIVE SOUND
3620 DATA ????,B552,SOUND CHANNEL BITS OF AKTIVE SOUND
3630 DATA ????,B553,SOUND TIMER
3640 DATA B1F0,,B554,SOUND RENDEZVOUS BYTE ?
3650 DATA ????,B555,SOUND EVENT BLOCK
3660 DATA B1F8,,B55C,SOUND QUEUE CHANNEL A
3670 DATA ????,B55F,SOUND QUEUE +3/TONE PERIOD
3680 DATA ????,B561,SOUND QUEUE +5/NOISE PERIOD
3690 DATA B237,,B59B,SOUND QUEUE CHANNEL B
3700 DATA B276,,B5DA,SOUND QUEUE CHANNEL C
3710 DATA B2A6,,B60A,SOUND AMPLITUDE ENVELOPE
3720 '
3730 DATA B2B5,,B619,?????
3740 '
3750 DATA B396,,B6FA,SOUND TONE ENVELOPE
3760 '
3770 DATA ????,B800,CAS MESSAGE FLAG
3780 DATA ????,B802,CAS INPUT BUFFER STATUS
3790 DATA ????,B803,CAS START ADDRESS INPUT BUFFER
3800 DATA ????,B805,CAS POINTER INPUT BUFFER
3810 DATA ????,B807,CAS FILE HEADER INPUT
3820 DATA ????,B847,CAS OUTPUT BUFFER STATUS
3830 DATA ????,B848,CAS ADDRESS START OUTPUT BUFFER
3840 DATA ????,B84A,CAS POINTER OUTPUT BUFFER
3850 DATA ????,B84C,CAS FILE HEADER OUTPUT
3860 DATA ????,B8D1,CAS SPEED
3870 '
3880 DATA B802,,????,?????
3890 DATA B804,,????,?????
3900 '
3910 DATA B114,,B8DC,EDI CURSOR ON FLAG
3920 DATA B115,,B8DD,EDI INSERT/OVERWRITE FLAG
3930 DATA B116,,B8DE,EDI COPYCURSOR POS/COLUMN AND ROW
3940 DATA B118,,B800,CAS IN FLAG
3950 DATA B119,,B801,CAS IN FLAG ?
3960 DATA B11A,,B802,CAS IN FILE TYPE ON READ
3970 DATA B11B,,B803,CAS IN BUFFER POINTER LOW
3980 DATA B11D,,B805,CAS IN BUFFER POINTER HI
3990 DATA B11F,,B807,CAS IN FILE NAME
4000 DATA B12F,,B817,CAS IN BLOCK NUMBER
4010 DATA B130,,B818,CAS IN LAST BLOCK FLAG
4020 DATA B131,,B819,CAS IN FILE TYPE

```

```

4030 DATA B132,,B81A,CAS IN DATA LENGTH
4040 DATA B134,,B81C,CAS IN DATA LOCATION
4050 DATA B136,,B81E,CAS IN FIRST BLOCK FLAG
4060 DATA B137,,B81F,CAS IN USER FIELDS
4070 DATA B15F,,B847,CAS OUT DIRECT FILE TYPE ON WRITE
4080 DATA B160,,B848,CAS OUT DIRECT/POINTER TO DATA LD
4090 DATA B162,,B84A,CAS OUT DIRECT/POINTER TO DATA HI
4100 DATA B164,,B84C,CAS OUT DIRECT/FILENAME
4110 DATA B174,,B85C,CAS OUT DIRECT/BLOCK NUMBER
4120 DATA B175,,B85D,CAS OUT DIRECT/LAST BLOCK FLAG
4130 DATA B176,,B85E,CAS OUT DIRECT/FILE TYPE
4140 DATA B177,,B85F,CAS OUT LEN OF DATA
4150 DATA B179,,B861,CAS OUT DIRECT/DATA LOCATION
4160 DATA B17B,,B863,CAS OUT DIRECT/FIRST BLOCK FLAG
4170 DATA B17C,,B864,CAS OUT DIRECT/TOTAL LENGTH OF DATA
4180 DATA B17E,,B866,CAS OUT DIRECT/ENTRY FOR HEADER
4190 DATA B1A4,,B88C,CAS OUT FILENAME
4200 DATA B1B4,,B89C,CAS OUT BLOCK NUMBER
4210 DATA B1B5,,B89D,CAS OUT FILE TYPE
4220 DATA B1B7,,B89F,CAS OUT DATA LENGTH
4230 DATA B1B9,,B8A1,CAS OUT DATA LOCATION
4240 '
4250 DATA B1BB,,B8A3,CAS OUT FIRST BLOCK FLAG
4260 DATA B1BC,,B8A4,CAS OUT USER FIELDS LOGICAL LENGTH
4270 DATA B1BC,,B520,TONE ENV TO USE ON CHAN A (?)
4280 DATA B1BE,,B522,NOISE PERIOD CHANNEL A (VORSICHT !!!)
4290 DATA B1BE,,B8A6,CAS OUT USER FIELDS ENTRY ADDRESS FOR MC
4300 '
4310 DATA B1D5,,B539,?????
4320 DATA B1E4,,B8CC,?????
4330 DATA B1E5,,B8CD,?????
4340 DATA B1E6,,B8CE,?????
4350 DATA B1E7,,????,?????
4360 DATA B1E8,,B8D0,?????
4370 DATA B1E9,,B8D1,CAS WRITE SPEED
4380 DATA B1EA,,B8D2,?????
4390 DATA B1EB,,B8D3,?????
4400 '
4410 DATA ????,,B8E0,?????
4420 DATA ????,,B8E4,RANDOM NUMBER BYTE 0+1
4430 DATA ????,,B8E6,RANDOM NUMBER BYTE 2+3
4440 DATA ????,,B8E8,FAC 1
4450 DATA ????,,B8ED,FAC 2
4460 DATA ????,,B8F2,FAC 3
4470 DATA ????,,B8F7,FLAG DEG/RAD
4480 '
4490 DATA B941,,B939,INTERRUPT ENTRY RST 7
4500 DATA B984,,B97C,LOW ROM OR RAM
4510 DATA B9B9,,B9B1,JP FAR CALL
4520 DATA B98A,,B982,LOW JUMP
4530 DATA B9C1,,B9B9,KL FAR ICALL
4540 DATA B9C7,,B9BF,FAR CALL
4550 DATA BA17,,BA10,JP TO A SIDEWAYS ROM
4560 DATA BA1D,,BA16,CALL TO A SIDEWAYS ROM
4570 DATA BA35,,BA2E,FIRM JUMP LOWER ROM

```

```

4580 DATA ????,,BA4A,KL L ROM ENABLE CONT D
4590 DATA ????,,BA54,KL L ROM DISABLE CONT D
4600 DATA BA5F,,BA5E,KL CURRENT UPPER ROM ENABLE
4610 DATA ????,,BA68,KL CURRENT UPPER ROM DISABLE
4620 DATA ????,,BA72,KL ROM RESTORE CONT D
4630 DATA BA79,,BA7E,KL SELECT AN UPPER ROM
4640 DATA BA7E,,????,KL ASK CLASS/VERSION MARK
4650 DATA ????,,BA83,KL PROBE ROM CONT D
4660 DATA BA87,,BA8C,KL STORE PREVIOUS ROM SELECTION
4670 DATA BAA1,,BAA6,KL LDIR ROMS DISABLED
4680 DATA BAA7,,BAAC,KL LDDR ROMS DISABLED
4690 DATA ????,,BAA2,KL CURR SELECTION CONT D
4700 DATA ????,,BAAB,KL LDIR CONT D
4710 DATA ????,,BAAC,KL LDDR CONT D
4720 DATA BAC6,,BACB,RAM LAM
4730 DATA BAD7,,BADC,RAM LAM (IX)
4740 '
4750 DATA BC0C,,????,CRTC-ADDRESS
4760 DATA BC0F,,????,CRTC ADDRESS
4770 '
4780 DATA BD5E,BD5B,BD3A,EDI LINE EDITOR
4790 DATA BD61,BD5E,BD3D,COPY 5 BYTES
4800 DATA ????,,BD40,REAL ARITH CREAL
4810 DATA ????,,BD43,REAL ARITH CREAL 4 BYTE INT TO REAL
4820 DATA ????,,BD46,REAL ARITH CINT
4830 DATA ????,,BD4C,REAL ARITH FIX
4840 DATA ????,,BD4F,REAL ARITH INT
4850 DATA ????,,BD55,REAL ARITH ADD
4860 DATA ????,,BD5B,REAL ARITH SUB
4870 DATA ????,,BD61,REAL ARITH MULT
4880 DATA ????,,BD64,REAL ARITH DVD
4890 DATA ????,,BD67,REAL ARITH ??
4900 '
4910 DATA BD64,BD61,????,?????
4920 DATA BD67,BD64,????,?????
4930 DATA BD6A,BD67,????,?????
4940 DATA BD6D,BD6A,????,?????
4950 DATA BD70,BD6D,????,?????
4960 DATA BD73,BD70,????,?????
4970 DATA BD76,BD73,????,?????
4980 DATA BD79,BD76,BD55,REAL ARITH ???
4990 DATA BD7C,BD79,????,?????
5000 DATA BD7F,BD7C,BD9D,REAL ARITH RANDOMIZE
5010 DATA BD82,BD7F,????,?????
5020 DATA ????,,BD82,REAL ARITH LOG10
5030 DATA ????,,BD85,REAL ARITH EXP
5040 DATA BD85,BD83,????,?????
5050 DATA BD88,BD85,BD94,REAL ARITH
5060 DATA BD8B,BD88,BDA0,REAL ARITH GET LAST RANDOM NUMBER
5070 DATA BD8E,BD8B,????,?????
5080 DATA BD91,BD8E,????,?????
5090 DATA BD94,BD91,BD70,REAL ARITH SGN
5100 DATA BD97,BD94,BD73,REAL ARITH SET DEG/RAD
5110 DATA BD9A,BD97,BD76,REAL ARITH/PI
5120 DATA BD9D,BD9A,BD79,REAL ARITH SQUARE

```

```

5130 DATA BDA8,BD9D,BD7C,REAL ARITH EXP
5140 DATA BDA3,BDA8,BD7F,REAL ARITH LOG
5150 DATA BDA6,,BD82,REAL ARITH LOG10
5160 DATA BDA9,BDA6,BD85,REAL ARITH GET EXP
5170 DATA BDAC,BDA9,BD88,REAL ARITH SIN
5180 DATA BDAF,BDAC,BD8B,REAL ARITH COS
5190 DATA BDB2,BDAF,BD8E,REAL ARITH TAN
5200 DATA BDB5,BDB2,BD91,REAL ARITH ATN
5210 DATA BDB8,BDB5,????,????
5220 DATA BDBE,BDB8,BD9A,REAL ARITH SEED RANDOM NUMBER
5230 DATA BDF4,,????,KM UPDATE KEY STATE MAP
5240 '
5250 DATA ????,,BDCD,TXT DRAW CURSOR INDIRECTION
5260 DATA ????,,BDD0,TXT UNDRAW CURSOR INDIRECTION
5270 DATA ????,,BDD3,TXT WRITE CHAR INDIRECTION
5280 DATA ????,,BDD6,TXT UNWRITE INDIRECTION
5290 DATA ????,,BDD9,TXT OUT ACTION INDIRECTION
5300 DATA ????,,BDDC,GRA PLOT INDIRECTION
5310 DATA ????,,BDDF,GRA TEST INDIRECTION
5320 DATA ????,,BDE2,GRA LINE INDIRECTION
5330 DATA ????,,BDE5,SCR READ INDIRECTION
5340 DATA ????,,BDE8,SCR MODE CLEAR INDIRECTION
5350 DATA ????,,BDEE,KM TEST BREAK INDIRECTION
5360 DATA ????,,BDF1,MC WAIT PRINTER INDIRECTION
5370 '
5380 DATA B0FF,,????,UPPER ROM BOUNDARY-1
5390 DATA B100,,????,START OF FIRMWARE RAM
5400 DATA B101,,????,START OF FIRMWARE RAM+1
5410 DATA B104,,????,????
5420 DATA B10E,,????,????

```

#### Listing: Programm zur Ausgabe der Tabelle

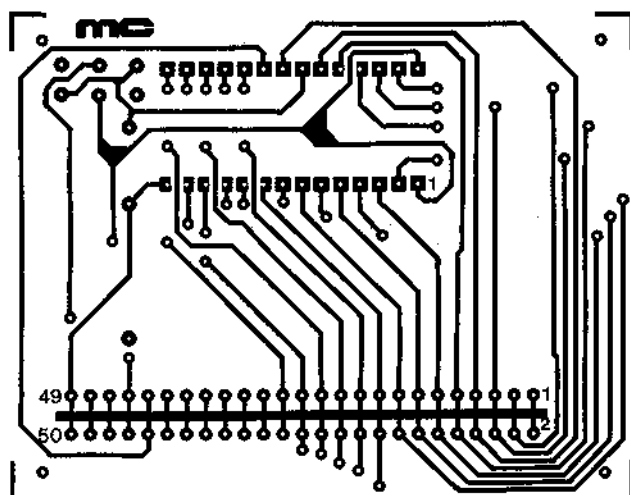
```

100 MODE 2:output=0
110 PRINT#output,"Vergleichstabelle der Systemadressen":PRINT
120 PRINT #output,"464      664      6128      Funktion
130 PRINT #output,STRING$(80,"-")
140 FOR i = 0 TO 409: READ a$,b$,c$,d$
150 IF b$="" THEN b$=a$
160 PRINT #output,c$;"      ";b$;"      ";a$;"      ";d$
170 NEXT i
180 PRINT #output,STRING$(80,"-")
190 END

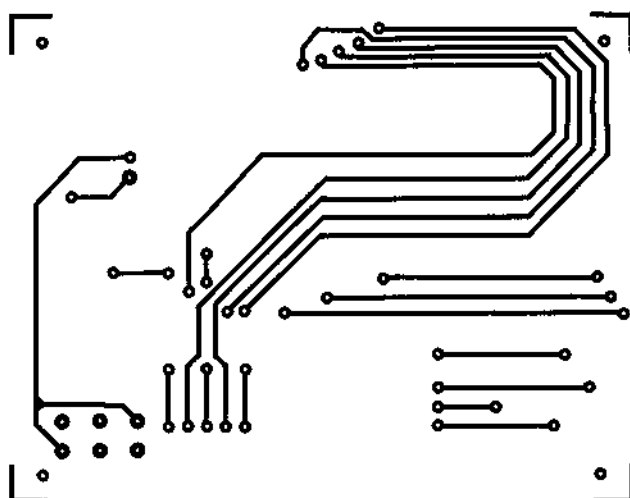
```

(Die Tabelle selbst muß angehängt werden! In der Tabelle stehen die Angaben in der Reihenfolge 6128, 664, 464. Das Programm gibt in der Folge 464, 664, 6128 aus!)

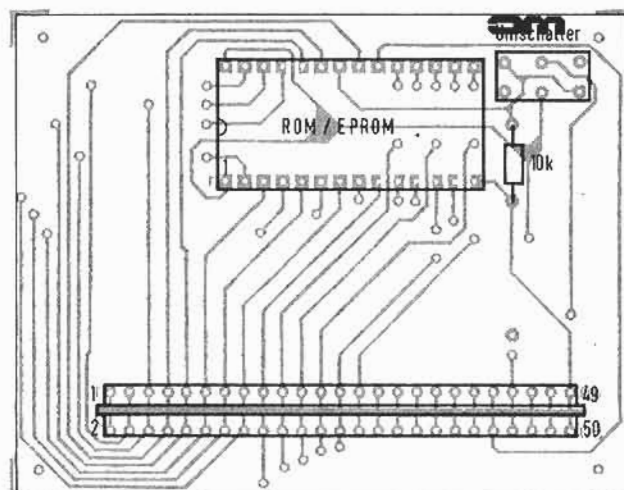
### 33. Anhang 3 – Die Betriebssystem-Erweiterungsplatine



Die Lötseite der ROM-/EPROM-Platine

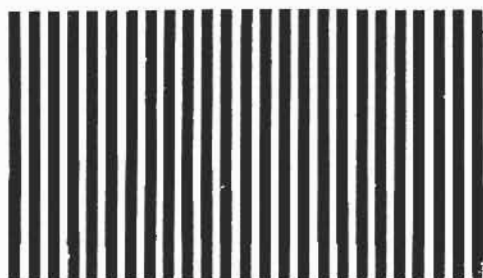


Die Bestückungsseite der Hauptplatine



M 2:1

Der Bestückungsplan der Erweiterungsplatine.  
Die Platine muß im Bereich des Busanschlusses ausgefräst und in diesen Ausschnitt muß dann die Busplatine eingesteckt und mit den Anschlußpins des Steckers verlötet werden.

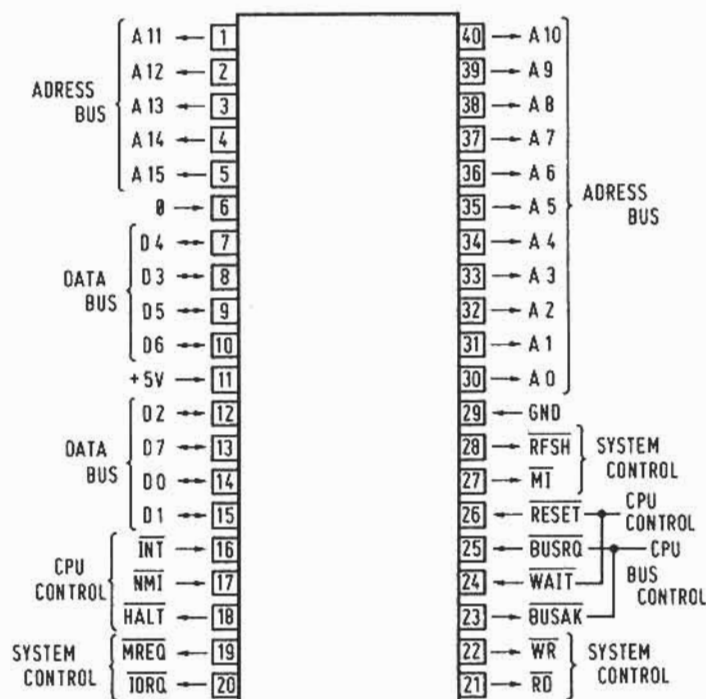


Die Busplatine ist auf Vorder- und Rückseite gleich. Die Bahn zum Anschlußpin 43 muß unterbrochen werden. An der dem CPC gegenüberliegenden Seite ist eine Drahtbrücke vom Mittelkontakt des Schalters (auf der Lötseite der Hauptplatine als alleinstehende Lötinsel über Pin 43 zu sehen) zur weiterführenden Bahn zu legen.

## 34. Anhang 4 – PINOUTs und Anschlußbelegungen

### 1. PINOUT des Prozessors Z80-A

(Nach Service-Unterlagen der Schneider Computer Division)

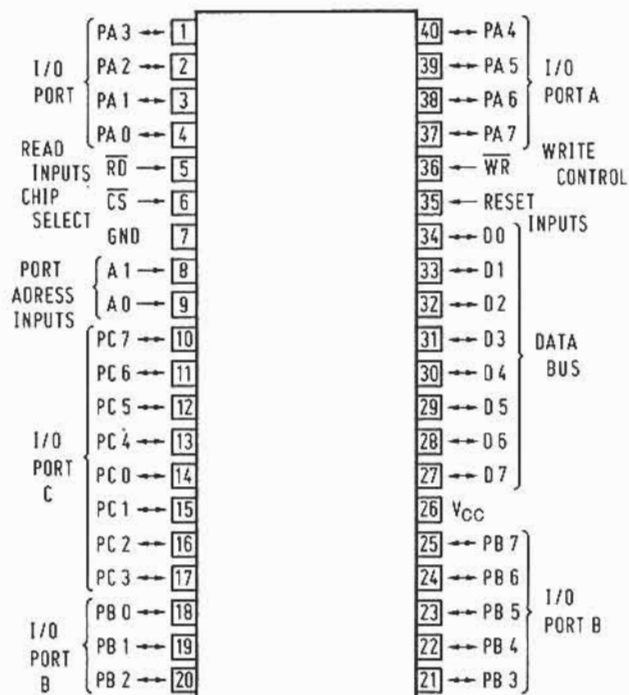






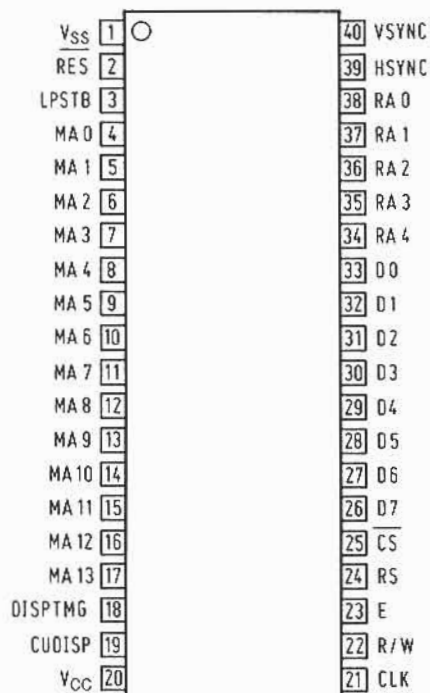
## 3. PINOUT des Schnittstellenbaustein 8255

(Nach Service-Unterlagen der Schneider Computer Division)



## 4. PINOUT des VDU-Controllers 6845

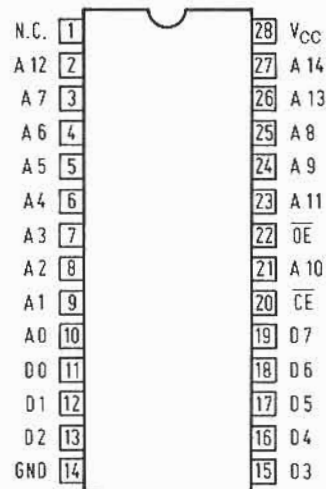
(Nach Service-Unterlagen der Schneider Computer Division)



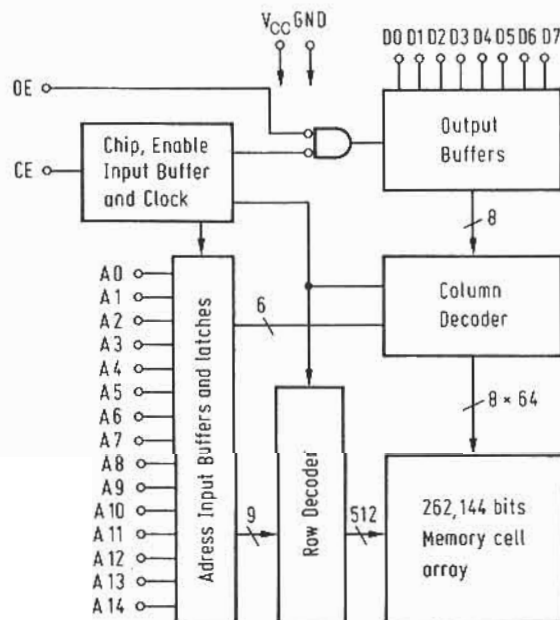
5. PINOUT des Gate Arrays 40010 (Typ HSG3130 oder HSG3170)  
 (Nach Service-Unterlagen der Schneider Computer Division)

D5	1	40	D4
D6	2	39	D3
D7	3	38	D2
CCLK	4	37	D1
NSYNC SYNC	5	36	VSS3
VDD1	6	35	D0
NRESET	7	34	NRAS
B	8	33	NMWE
DISPEN	9	32	NINTERRUPT J105
G	10	31	NCASAD IC-SOCKET
HSYNC	11	30	A14
R	12	29	NRAMRO
VSNC	13	28	A15
NCPU	14	27	NROMEN
VSS1	15	26	VSS2
NCAS	16	25	VDD2
NMREQ	17	24	CK16
NIORQ	18	23	N244EN
NPH1	19	22	READY
NM1	20	21	NRO

### 6.A PINOUT des Firmware-ROMs TMM 23256P (Nach Unterlagen von Toshiba MOS Memory Products)

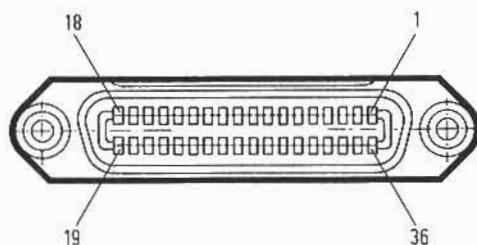


### 6.B Blockdiagramm des ROMs TMM 23256P (Nach Unterlagen von Toshiba MOS Memory Products)



## 7. Der Druckeranschluß beim CPC 6128

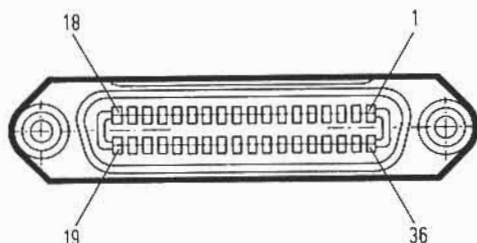
(Nach Service-Unterlagen der Schneider Computer Division)



Pin 1	STROBE	Pin 19	GND
Pin 2	D 0	Pin 20	GND
Pin 3	D 1	Pin 21	GND
Pin 4	D 2	Pin 22	GND
Pin 5	D 3	Pin 23	GND
Pin 6	D 4	Pin 24	GND
Pin 7	D 5	Pin 25	GND
Pin 8	D 6	Pin 26	GND
Pin 9	GND	Pin 27	GND
Pin 11	BUSY	Pin 28	GND
Pin 14	GND	Pin 33	GND
Pin 16	GND	Alle weiteren Stifte NC	

## 8. Anschluß für die zweite Diskettenstation beim CPC 6128

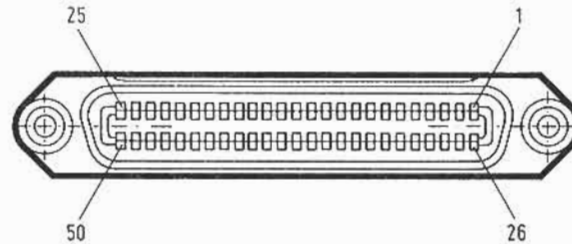
(Nach Service-Unterlagen der Schneider Computer Division)



Pin 1 - 19	GND	Pin 21	NC	Pin 22	NC
Pin 20	NC	Pin 24	NC	Pin 25	DRIVE 1 SELECT
Pin 23	INDEX	Pin 27	MOTOR ON	Pin 28	DIREKTION SELECT
Pin 26	NC	Pin 30	WRITE DATA	Pin 31	WRITE GATE
Pin 29	STEP	Pin 33	WRITE PROTECT	Pin 34	READ DATA
Pin 32	TRACK 0	Pin 36	READY		
Pin 35	SIDE 1 SELECT				

## 9. Der Erweiterungsanschluß beim CPC 6128

(Nach Service-Unterlagen der Schneider Computer Division)



Pin 1	SOUND	Pin 18	$\overline{\text{INT}}$	Pin 35	D 6
Pin 2	A 15	Pin 19	$\overline{\text{BUSRQ}}$	Pin 36	D 4
Pin 3	A 13	Pin 20	$\overline{\text{READY}}$	Pin 37	D 2
Pin 4	A 11	Pin 21	$\overline{\text{RESET}}$	Pin 38	D 0
Pin 5	A 9	Pin 22	$\overline{\text{ROMDIS}}$	Pin 39	$\overline{\text{MREQ}}$
Pin 6	A 7	Pin 23	$\overline{\text{RAMDIS}}$	Pin 40	$\overline{\text{RFSH}}$
Pin 7	A 5	Pin 24	$\overline{\text{LPEN}}$	Pin 41	$\overline{\text{RD}}$
Pin 8	A 3	Pin 25	GND	Pin 42	$\overline{\text{HALT}}$
Pin 9	A 1	Pin 26	GND	Pin 43	$\overline{\text{NMI}}$
Pin 10	D 7	Pin 27	A 14	Pin 44	$\overline{\text{BUSAK}}$
Pin 11	D 5	Pin 28	A 12	Pin 45	$\overline{\text{BUSRESET}}$
Pin 12	D 3	Pin 29	A 10	Pin 46	$\overline{\text{ROMEN}}$
Pin 13	D 1	Pin 30	A 8	Pin 47	$\overline{\text{RAMRD}}$
Pin 14	+ 5 VOLT	Pin 31	A 6	Pin 48	$\overline{\text{CURSOR}}$
Pin 15	$\overline{\text{M1}}$	Pin 32	A 4	Pin 49	$\overline{\text{EXP}}$
Pin 16	$\overline{\text{IORQ}}$	Pin 33	A 2	Pin 50	TAKT
Pin 17	$\overline{\text{WR}}$	Pin 34	A 0		

## Diskette zum Buch

Zu diesem Buch

gibt es eine **Sammeldiskette** von den abgedruckten Programmen.

Sie können diese Diskette bei unserem Franzis-Software-Service,  
Postfach 37 01 20, 8000 München 37, direkt bestellen, oder verwenden  
Sie bitte die beiliegende Bestellkarte.

Bestell-Nr.: 3-8178-0232-3

Preis: 49,- DM

Liefermöglichkeit und Preisänderung vorbehalten.

**Plötzlich auftretende Fragen finden  
in diesem Band eine gründliche Antwort**

Dipl.-Ing. (FH) Herwig Feichtinger

# **Arbeitsbuch Mikrocomputer**

Funktion und Anwendung von Mikrocomputern, Peripherie und Software. 2., neu bearbeitete und erweiterte Auflage. 624 Seiten mit 350 Abbildungen. Lwstr-gebunden mit Schutzumschlag DM 108.—. ISBN 3-7723-8022-0

Im Arbeitsbuch Mikrocomputer konzentriert sich die Theorie und die Praxis der letzten Jahre wie in einem Brennglas zu einem Punkt und gibt den Ausblick auf die Zukunft.

Das Arbeitsbuch Mikrocomputer faßt die weitverstreute Basis-Literatur zusammen, filtert das unumstößlich Wichtige heraus und bereitet es so auf, daß der Benutzer des Werkes optimal informiert wird.

Das Arbeitsbuch Mikrocomputer ist in erster Linie ein Nachschlagewerk. Es beantwortet die Fragen der täglichen Praxis. Z. B. Befehlssätze von Mikroprozessoren und Betriebssystemen, Anschlußbelegungen von Bauelementen, Normen von Schnittstellen, Bedienung von Assemblern und Compilern. Die höheren Programmiersprachen gehören auch dazu.

Das Arbeitsbuch Mikrocomputer ist auch ein Lehrbuch. Neben den reinen Fakten, Zahlen und Tabellen sind reichlich Erklärungen und Hinweise zum Wieso und Warum angesiedelt. Das reicht von einfacher digitaler Logik über den internen Aufbau von Mikroprozessoren bis hin zu den Betriebssystemen MS-DOS und Unix.

Das Arbeitsbuch Mikrocomputer ist dazu noch eine moderne Datenbank auf dem handsamsten Medium, dem Papier. Über das umfangreiche Inhaltsverzeichnis oder das aufgeschlüsselte Stichwortregister stößt der Benutzer ganz schnell auf die Stelle, die ihm die Information serviert, die er braucht und die ihm weiterhilft.

Das Arbeitsbuch Mikrocomputer bietet also eine Arbeitserleichterung und eine Literaturersparnis, die gar nicht hoch genug angesetzt werden kann.

Preisänderungen und Liefermöglichkeit vorbehalten.

**Franzis-Verlag, München**



Das große CPC-Arbeitsbuch fängt dort an, wo die Handbücher zu den Schneider CPCs aufhören.

Das große CPC-Arbeitsbuch ist das Bindeglied zwischen Handbuch und praktischem Einsatz. Mit Hilfe der Demonstrations- und Hilfsprogramme wird hier ein Wissen vermittelt, das es in sich hat. Programmiertricks werden aufgezeigt, die erst so richtig zeigen, was in den CPCs alles steckt und was durch geschickte Programmierung aus ihnen herausgeholt werden kann. Ganz gleich, ob es um den nackten CPC oder um die trickreiche Handhabung der Diskettenstation geht. Das große CPC-Arbeitsbuch bietet viele Lösungen für den täglichen Umgang mit dem CPC.

Für Programmierer, die nicht nur Software von der Stange auf ihrem CPC laufen lassen wollen, ist das große CPC-Arbeitsbuch ein Muß!



ISBN N 3-7723-8421-8 DM +068.00