BRÜCKMANN · SCHIEB

DAS GROSSE

FLOPPY BUCH

DISKETTENPROGRAMMIERUNG FÜR EINSTEIGER, FORTGESCHRITTENE UND PROFIS MIT DEM CPC

EIN DATA BECKER BUCH

BRÜCKMANN · SCHIEB

DAS GROSSE

FLOPPY BUCH

DISKETTENPROGRAMMIERUNG FÜR EINSTEIGER, FORTGESCHRITTENE UND PROFIS MIT DEM CPC

EIN DATA BECKER BUCH

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

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

Vorwort

Dieses große Floppybuch zum CPC in der zweiten überarbeiteten Auflage beschäftigt sich mit den Floppy-Laufwerken des CPC 464, des CPC 664 und des CPC 6128 gleichermaßen. Es werden Fehler im Betriebssystem angesprochen und Unzulänglichkeiten des Laufwerkes bzw. des AMSDOS ausgebügelt (Fehlermeldungen auf den Bildschirm; relative Dateiverwaltung).

Für den Anfänger sind zahlreiche Informationen über die sequentielle Datenspeicherung mit reichlich Beispielprogrammen und Abbildungen vorhanden. Der technisch interessierte Programmierer findet darüber hinaus Wissenswertes über "seine" Floppy in diesem Buch (Controller-Programmierung etc.).

Bei der Erstellung dieses Buches wurden die Autoren von dem Team des Lektorates maßgeblich unterstützt; insbesondere *Brigitte Witzer*, die Lektorin, hat sich hier verdient gemacht, weswegen wir ihr danken wollen.

Wir wünschen Ihnen, lieber Leser, noch viel Spaß mit diesem Buch und vor allem Erfolg bei der Arbeit mit Ihrem Schneider-Rechner und Ihrem Diskettenlaufwerk.

Düsseldorf, im Februar 1986

Rolf Brückmann
Jörg Schieb

Inhaltsverzeichnis

Kapitel 1
Einführung in die Programmierung der DDI-1

1.1	Was bietet eine Floppy?	13
1.2	Die Diskette	16
1.2.1	Die Entwicklung	16
1.2.2	Die mitgelieferte Diskette	20
1.3	Einiges über CP/M	21
1.3.1	Was ist CP/M?	21
1.3.2	Laden und Starten	22
1.3.3	Formatieren von Disketten	24
1.3.4	Kopieren unter CP/M	27
1.3.5	Kopieren mit 2 Laufwerken	29
1.3.6	DISCCHK - Überprüfen von Disketten	30
1.3.7	Spezielle Formatierungsformate	31
1.3.8	Transiente und residente Befehle	34
1.3.9	Kopieren von Files mittels Filecop	35
1.3.10	Wild Cards	38
1.3.11	Das DIR-Kommando	39
1.3.12	Das ERA-Kommando	40
1.3.13	Das REN-Kommando	41
1.3.14	Das TYPE-Kommando	41
1.3.15	Umschalten des Standardlaufwerkes	41
1.3.16	Das PIP-Kommando	42
1.3.17	Das STAT-Kommando	43
1.3.18	BOOTGEN	45
1.3.19	MOVCPM und SYSGEN	45
1.3.20	SETUP	46
1.3.21	AMSDOS	46
1.4	Arbeiten mit AMSDOS	47
1.4.1	Was ist AMSDOS?	47
1.4.2	LOAD und RUN	49
1.4.3	Die Dateibefehle	54
1.4.4	Die zusätzlichen AMSDOS-Befehle	62

1.5 Sequentielle Datenspeicherung 74
1.5.1 Was ist sequentielle Datenspeicherung? 74
1.5.2 Die Programmierung sequentieller Files 79
1.5.3 Sequentielle Dateien und Feldvariablen 87
1.5.4 Unterschiede zwischen PRINT# und WRITE# 98
1.5.5 Unterschiede zwischen INPUT# und LINE INPUT# 103
1.6 Relative Datenspeicherung 108
1.6.1 Was ist relative Datenspeicherung? 108
1.6.2 Das Einrichten einer relativen Datei 111
1.6.3 Wie arbeite ich mit einer relativen Datei? 114
1.6.4 Wie funktioniert die relative Datenspeicherung? 116
1.6.5 Wann ist die relative Datenspeicherung sinnvoll? 118
1.6.6 Die ISAM-Dateien 119
1.7 Die verschiedenen Schneider-Rechner 121
1.7.1 Der CPC 664 und der CPC 6128 und Diskettenfehler 122
1.7.2 Weitere Unterschiede des CPC 664 und des CPC 6128 123

Kapitel 2
Disketten-Programmierung für Fortgeschrittene

2.1 Die DOS-Vektoren 125
2.1.1 DISK IN OPEN 128
2.1.2 DISK IN CLOSE 131
2.1.3 DISK IN ABANDON 132
2.1.4 DISK IN CHAR 132
2.1.5 DISK IN DIR 133
2.1.6 DISK RETURN 134
2.1.7 DISK TEST EOF 135
2.1.8 DISK OUT OPEN 136
2.1.9 DISK OUT CLOSE 139
2.1.10 DISK OUT ABANDON 140
2.1.11 DISK OUT CHAR 140
2.1.12 DISK OUT DIRECT 141
2.1.13 DISK CATALOG 142
2.1.14 Das Patchen der Vektoren 143

2.2 Die Befehlserweiterungen des AMSDOS 145
2.2.1 Programmierung der Erweiterungen in Assembler 146
2.2.2 Die "versteckten" Befehle der Befehlserweiterung 151
2.2.2.1 Der Befehl &81 Message on/off 151
2.2.2.2 Der Befehl &82 Drive Parameter 152
2.2.2.3 Der Befehl &83 Disk Format Parameter 155
2.2.2.4 Der Befehl &84 Read Sector 156
2.2.2.5 Der Befehl &85 Write Sector 160
2.2.2.6 Der Befehl &86 Format Track 160
2.2.2.7 Der Befehl &87 Seek Track 163
2.2.2.8 Der Befehl &88 Test Drive 164
2.2.2.9 Der Befehl &89 Retry Count 164

Kapitel 3
Technik der Floppy und der Diskette

3.1 Der Floppy Controller 167
3.1.1 Beschreibung der Hardware des Floppy Controllers 169
3.1.2 Der FDC 765 170
3.1.2.1 Die Anschluß-Belegung des FDC 172
3.1.2.2 Die Programmierung des FDC 765 180
3.1.2.3 Die Statusregister des FDC 765 198
3.1.2.4 Einsatz des FDC 765 im CPC 207
3.2 Der Aufbau der Diskette 209
3.2.1 Die drei Diskettenformate 209
3.2.1.1 Das Standard-CPC- oder System-Format 210
3.2.1.2 Das Datenformat 211
3.2.1.3 Das IBM-Format 211
3.2.2 Der Aufbau der Directory 212
3.2.3 Der Aufbau der Dateien 217
3.2.4 Die physikalische Aufzeichnung der Daten 220
3.2.4.1 MF, MFM, Bits und Magnetismus 220
3.2.4.2 Von Gaps, IDs und Adresse-Marks 228

Kapitel 4
Das ROM und RAM des AMSDOS

4.1 Das System-RAM des AMSDOS 235
4.2 Das AMSDOS-ROM 247

Kapitel 5
Programme und Tricks für die DDI-1

5.1 Fehler bei MERGE und CHAIN MERGE 329
5.2 Fehlermeldungen 334
5.2.1 Erläuterungen zur Fehlerabfangroutine
- wie wird es gemacht? 346
5.3 Relative Dateiverwaltung 350
5.3.1 Erläuterungen zum Maschinenprogramm 360
5.4 Das Dateiverwaltungsprogramm 378
5.5 Der Diskmonitor 395
5.6 Der Diskettenmanager 403

Stichwortverzeichnis 413

Kapitel 1:
Einführung in die Programmierung der DDI-1

Die CPC-Rechner sind beliebte Geräte und zeichnen sich durch ein gutes Preis/Leistungsverhältnis aus. Wie Sie wissen, gibt es bereits drei verschiedene CPC-Rechner auf dem Markt. Das Floppybuch soll sich mit allen diesen drei CPC-Rechnern beschäftigen: dem **CPC 464**, dem **CPC 664** und dem **CPC 6128**.

Ist das äußere Erscheinunsgbild der drei Rechner auch verschieden (andere Tastatur etc.), so ist die Bedienung und die Handhabung der Diskettenlaufwerke für alle Rechner (beinahe) vollkommen identisch. Wegen der verschiedenen BASIC-Versionen ist es manchmal notwendig, Unterscheidungen zu machen. Ist keine besondere Anmerkung gemacht, so gilt das Gesagte für alle drei CPC-Rechner gleichermaßen.

1.1 Was bietet eine Floppy?

Alle CPC-Rechner (wie auch jeder andere Computer) verfügen nur über einen sogenannten *flüchtigen Speicher* - die in den Rechner eingegebenen Daten und Programme gehen also nach dem Ausschalten verloren. Um diese Informationen aufzubewahren, sie quasi zu konservieren, benötigt man externe Speicher wie z.B. ein Kassettenlaufwerk oder eben ein Diskettenlaufwerk.

Der CPC 464 hat noch ein eingebautes Kassettenlaufwerk - das Diskettenlaufwerk ist nur ein optionales (also zusätzliches) externes Speichermedium. Der CPC 664 und der CPC 6128 haben das Diskettenlaufwerk standardmäßig im Gehäuse der Tastatur eingebaut. Die eingebauten Diskettenlaufwerke unterscheiden sich vom *optionalen* Laufwerk überhaupt nicht.

Sofern Sie einen CPC 464 besitzen, haben Sie sicherlich in Ihrem Handbuch entdeckt, daß Sie mit zwei verschiedenen Geschwindigkeiten auf Kassette schreiben können. Beim Lesen stellt sich der Rechner automatisch in seiner

Lesegeschwindigkeit darauf ein. Mit dem Befehl **SPEED WRITE** können Sie das Kassettenlaufwerk veranlassen, entweder mit 1000 *Baud* oder mit 2000 *Baud* zu schreiben. Was bedeutet *Baud*? *Baud* steht für Bits in der Sekunde. 1000 Baud bedeutet also, daß der Rechner 1000 Bits pro Sekunde auf Kassette schreibt und später auch wieder liest. 1000 Bits sind 1000/8=125 Bytes pro Sekunde. Bei 2000 Baud werden entsprechend 250 Bytes pro Sekunde geschrieben und gelesen. Warum dann nicht immer mit 2000 Baud arbeiten, fragen Sie sich? Ganz einfach: Dann ist die Gefahr eines Schreib-/Lesefehlers größer, d.h. daß möglicherweise beim Lesen eines Programmes nicht dasselbe gelesen wird, was geschrieben wurde. Sie können sich denken, was das bedeutet.

Bei einer Floppy besteht gegenüber dem Kassettenlaufwerk der Vorteil, daß man nach einem Lesefehler einen weiteren Leseversuch starten kann (bei einem Kassettenlaufwerk müßte man per Hand oder per komplizierter Mechanik das Band wieder zurückspulen).

Normalerweise werden bis zu zehn Versuche unternommen, eine Informationseinheit von Diskette zu lesen, bevor die Meldung gegeben wird, daß diese Einheit *nicht* lesbar ist. Beim Kassettenlaufwerk muß diese Fehlermeldung **direkt** nach Erkennen des Fehlers gegeben werden.

Für den professionellen Gebrauch ist ein Kassettenlaufwerk daher wohl kaum geeignet. Hierfür gibt es zwei Hauptgründe:

1. Ein Kassettenlaufwerk ist viel zu langsam. Professionelle Programme werden immer umfangreicher und damit länger, nicht selten sind sie 25 KBytes lang, was bedeutet, daß man lange Wartezeiten für den Ladevorgang in Kauf nehmen muß. Des weiteren müssen dann meist noch Daten geschrieben und gelesen werden, wobei wir auch schon beim zweiten Nachteil wären:

2. Es ist sehr schwierig, ganz bestimmte Dateien von Kassette einzuladen. Eine Datei stellen beispielsweise ein gespeichertes Programm oder gespeicherte Daten dar. Wenn

Sie mehrere Programme auf Kassette gespeichert haben, so müssen Sie entweder umständlich und ungenau vorspulen und dann versuchen zu laden, oder Sie lassen den Computer die Datei suchen - was aber bedeutet, daß er jede andere Datei überlesen muß.

Diese Nachteile des Kassettenbetriebes lassen sich vermeiden: Schnelleres Arbeiten und auch leichtere Handhabung bieten Disketten, die scheibenförmigen Speichermedien, die in eine Floppy-Station eingeführt werden.

Bei großen Anlagen verwendet man Platten bzw. Plattenstapel, die größeren Schwestern der Disketten. Eine Platte kann bis zu 50 MBytes (50 Mega = 50 Millionen) fassen. Allerdings werden heute noch wichtige Daten auf die langsameren Magnetbänder kopiert, meist dann, wenn man Daten archivieren will. Magnetbänder sind hier sehr gut geeignet, weil sie weniger Platz beanspruchen und billiger sind.

Doch zurück zum CPC und dem Diskettenbetrieb:

"Willkommen im Kreis der stolzen Besitzer eines Schneider DDI-1. Sie werden schnell sehen, daß Ihre Entscheidung richtig war und daß sich Ihre Investition gelohnt hat."

Dies verspricht Ihnen das Handbuch zu Ihrer Schneider DDI-1 Disketten-Station. Besitzen Sie einen CPC 664 oder einen CPC 6128, so dürfen Sie dieses Zitat auch auf sich beziehen. Wir wollen Ihnen mit diesem Buch dabei helfen, diesen Einleitungssatz zu verwirklichen, indem wir zeigen, welche Möglichkeiten in diesem Floppy-Laufwerk stecken.

1.2 Die Diskette

1.2.1 Die Entwicklung

Noch vor wenigen Jahren waren die meisten Disketten 8 Zoll groß. Diese Disketten aber erwiesen sich als wenig praktikabel; etwas später wurden dann die 5-1/4-Zoll-Disketten entwickelt, so wie sie heute bei fast allen gängigen Home- und Personal-Computern benutzt werden (IBM, Apple, Commodore 64, Sirius etc). Eine Revolution bahnte sich mit Apples LISA bzw. mit dem kleinerem MacIntosh an, der 3-1/2-Zoll-Disketten verwendet - die Schneider-Laufwerke aber arbeiten sogar mit 3-Zoll-Disketten. Die Miniaturisierung schreitet also auch merklich auf dem Gebiet der externen Speicher voran.

Da noch sehr wenige der Drei-Zoll-Disketten produziert werden, sind sie bisher relativ teuer. Aber ein Preisgefälle wird nicht lange auf sich warten lassen, so daß sie in einem Jahr wahrscheinlich nur noch die Hälfte kosten dürften.

Sie kennen sicherlich die 5-1/4-Zoll-Disketten, die aus einer flexiblen weichen Plastikhülle und einer innenliegenden Scheibe bestehen. Bei Ihren Disketten sind diese Scheiben aber aus Sicherheitsgründen durch ein stabiles Plastikgehäuse geschützt. Sie können auch nicht mehr mit dem Finger in das für den Schreib/Lesekopf vorgesehene Loch (Abb. 1, Punkt 1) geraten. Dieses Loch ist mit einem metallischen Verschluß versehen, der sich erst öffnet, wenn die Diskette in das Laufwerk eingeführt wird.

Dasselbe gilt für das *Indexloch* (Punkt 2). Disketten haben ein Indexloch, damit das Laufwerk immer "weiß", wann wieder eine Umdrehung getan ist. Die Indexlöcher werden durch eine Lichtschranke erkannt. Dem Laufwerk dient dieses Loch also praktisch zur Orientierung. Es gibt auch Diskettenlaufwerke, die dieses Indexloch nicht benötigen, allerdings werden diese Laufwerke dann meist langsamer, da man per Software dieses Problem lösen muß.

Beide Öffnungen sind durch einen Klappverschluß gesichert, den Sie einmal vorsichtig mit dem Finger wegschieben können. Nehmen Sie Ihre Diskette in die Hand, mit der B-Seite nach oben, das Schreib-/Lesekopf-Loch von Ihnen abgewendet. Dann fahren Sie mit dem Fingernagel in der linken Laufschiene entlang, bis Sie hier einen weißen Plastikschieber (3) erkennen. Den ziehen Sie bis an den Anschlag zu sich hin. Sie sehen, daß sowohl das Indexloch als auch das Loch für den Schreib-/Lesekopf freigelegt werden. Nun schauen Sie direkt auf die eigentliche Diskette, auf der die Informationen gespeichert werden. Berühren Sie diese aber nicht mit den Fingern! In der Mitte (4) ist das Führungsloch. Hier wird die Diskette gepackt und gedreht. Wenn Sie nun vorsichtig mit der anderen Hand die Scheibe drehen, können Sie auch das Indexloch sehen. Schließen Sie nun die Diskette wieder vorsichtig, indem Sie den weißen Plastikschieber loslassen.

Zum Schluß wären noch die zwei Öffnungen zu erwähnen, die dem *Schreibschutz* dienen (5). Am linken Rand jeder Diskette befindet sich ein Loch, das mit einem kleinem Pfeil versehen ist. Es dient dazu, eine Diskette vor Überschreiben zu schützen. Sie kennen dies auch von Ihrem Kassettenrecorder. Wenn das Loch geschlossen ist, können Sie auf die Diskette schreiben. Sie können aber auch den Plastikschieber verschieben, so daß das Loch offen ist. In diesem Fall kann auf die Diskette nicht mehr geschrieben werden, was besonders sinnvoll ist, wenn Sie auf der Diskette wertvolle Programme haben, die nicht versehentlich überschrieben werden sollen. Bei der mitgelieferten Diskette von Schneider, auf der sich **CP/M** und **LOGO** befinden, müssen Sie den Schieber von oben nach unten verschieben. Bei Disketten von anderen Herstellern geschieht dies etwas anders, aber genauso einfach. Bei den von uns verwandten Disketten müssen Sie den Schieber von rechts nach links z.B. mit einem Kuli bewegen.

Ihre Disketten können beidseitig verwendet werden. Bei den 5-1/4-Zoll-Disketten war diese Version extra mit der Aufschrift

"Double Sided" versehen; ein Luxus, den man entsprechend bezahlen mußte. Bei Ihrem Schneider ist es aber selbstver-ständlich, daß Sie beide Seiten Ihrer Disketten nutzen können.

Auf jede der Diskettenseite passen 180 KBytes, so daß insgesamt 360 KBytes bzw. 360.000 Zeichen auf einer Diskette speicherbar sind.

Um eine Diskette in das Laufwerk einzulegen, müssen Sie lediglich die Diskette in die Hand nehmen und vorsichtig in Pfeilrichtung in das Laufwerk einführen, bis Sie ein Klackgeräusch hören: Dann ist die Diskette fest in das Laufwerk eingerastet. Um sie wieder aus dem Laufwerk zu entfernen, drücken Sie auf den Knopf, der sich rechts vom Einführungsschlitz befindet. Diesen Knopf nennt man auch Disketten-Auswurf-Knopf. Betätigen Sie diesen Knopf aber nie, während Ihr Laufwerk arbeitet! Dies kann zu absolutem Datenverlust führen.

Beachten Sie auch, daß sich sowohl beim Ein- als auch beim Ausschalten keine Diskette mehr in Ihrem Laufwerk befindet - auch dann kann es aufgrund von Spannungsspitzen am Schreib-/Lesekopf zu Datenverlust kommen.

Eine gekaufte Diskette ist noch "roh", ein unbeschriebenes Blatt gewissermaßen. Bevor man eine Diskette benutzen kann, muß sie formatiert werden, ähnlich einem weißen Blatt Papier, das zur korrekten Beschriftung erst mit Linien versehen wird.
Wie Sie eine Diskette neu formatieren können, wird Ihnen in Kapitel 1.3.3 (CP/M) erklärt.

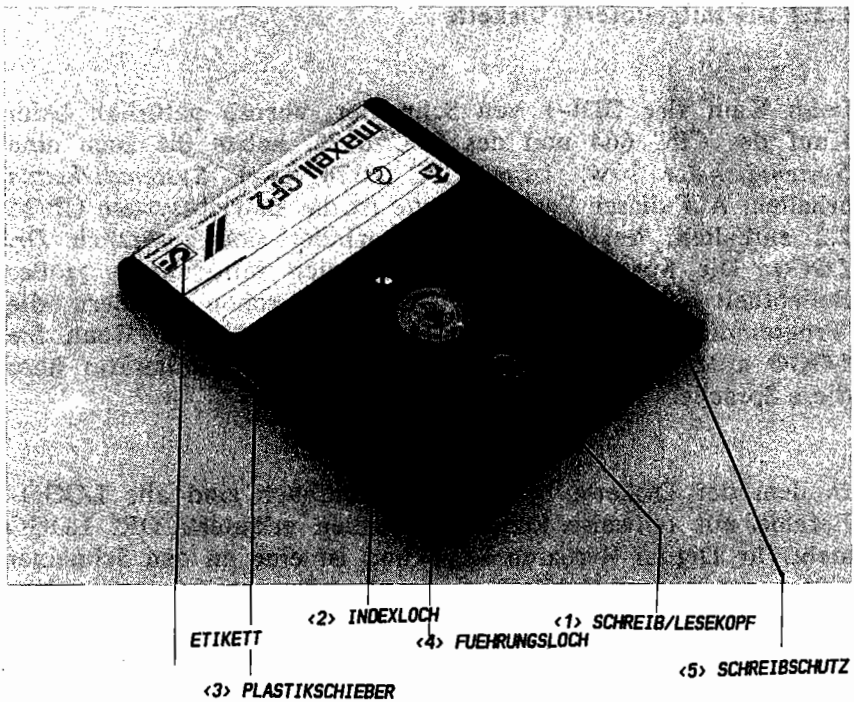


Abb. 1 3-Zoll-Diskette

1.2.2 Die mitgelieferte Diskette

Beim Kauf der DDI-1 von Schneider, ebenso natürlich beim Kauf des CPC 664 und des CPC 6128, haben Sie auch eine *Systemdiskette* bzw. beim CPC 6128 zwei *Systemdisketten* erhalten. Auf dieser *Systemdiskette* ist das Betriebssystem CP/M 2.2 enthalten, auf der B-Seite befindet sich zusätzlich *Dr. LOGO*. Die Sprache *LOGO* erfreut sich mittlerweile großer Beliebtheit und wurde entwickelt, um schon Kindern die "Programmierung" von Computern zu ermöglichen. Wenn Sie **LOGO** nutzen wollen, so gibt es einiges an Fachliteratur über diese Sprache.

In dem der Diskette beigefügten Handbuch sind alle LOGO-Befehle mit teilweise kurzen Beispielen erläutert. DR. LOGO steht für Digital Research Logo und ist eine an den Schneider CPC angepaßte LOGO-Version. Man hat die Programmiersprache LOGO um einige Sound-Befehle erweitert, um auch diese Möglichkeiten des Schneider CPC nutzen zu können. Weiterhin wurden die Cursortasten zur Editierung des Programmes mit einbezogen.

1.3 Einiges über CP/M

1.3.1 Was ist CP/M?

Auf Seite *A* Ihrer Diskette befindet sich das Betriebssystem *CP/M*. *CP/M* steht für *C*ontrol *P*rogram for *M*icrocomputers, das es nur auf Rechnern mit den weit verbreiteten Prozessoren 8080, 8085 und Z-80 gibt. Wer einen Rechner hat, der unter CP/M arbeitet, der kann auf eine große Bibliothek von Anwendersoftware zurückgreifen. CP/M hat den Vorteil, daß nur sehr wenige Änderungen (wenn überhaupt) an den bestehenden Programmen nötig sind, um sie auf einem anderen Rechner lauffähig zu machen.

Mittlerweile gibt es schon diverse Anbieter von CP/M-Software, so wird beispielsweise bereits WORDSTAR oder DBASE angeboten; leider sind die Programme noch etwas langsam auf den Schneider-Rechnern.

Es existieren unter CP/M für bestimmte Routinen (etwa für Ausgabe auf Bildschirm etc.) Sprungtabellen, die man fest definiert hat. Diese werden von den CP/M-Programmen angesprungen, so daß nur ein Minimum an Anpassung notwendig wird. Die entsprechenden Grundroutinen werden bei jedem Rechner mit dem Booten von CP/M geladen.

Ein weiterer Vorteil besteht darin, daß ein Anwender, der die Bedienung von CP/M gelernt hat, auf jedem anderen Rechner unter diesem Betriebssystem arbeiten kann. Wer BASIC auf einem Rechner X gelernt hat, der kann noch lange nicht auf dem Rechner Y mit BASIC erfolgreich programmieren. Anders bei CP/M, das stark standardisiert ist. Auf Ihrer Diskette befindet sich CP/M 2.2. Sie benötigen es auf jeden Fall zum Formatieren oder Kopieren von Disketten, können es aber auch anderweitig nutzen - wenn Sie zu diesem Zweck CP/M erlernen möchten, dann bietet Ihnen der Buchmarkt reichlich Auswahl.

1.3.2 Laden und Starten von CP/M

Folgendes sei vorausgeschickt: Wenn Sie mit Ihrer Floppy arbeiten, so schalten Sie grundsätzlich zuerst das Laufwerk ein und erst dann Monitor und Rechner. Sonst wird bei einem internen Testlauf, bei dem alle ausgeschalteten Geräte erkannt werden, das Laufwerk als nicht angeschlossen registriert und alle Befehle, die sonst an die Floppy gingen, wie früher an das Kassettenlaufwerk gerichtet.

Als Besitzer des Schneider CPC 6128 haben Sie neben CP/M 2.2 noch CP/M 3.0 (PLUS) erhalten. Der Unterschied von CP/M 2.2 und CP/M 3.0 ist hauptsächlich das Alter: CP/M 3.0 ist die neuere, überarbeitete Version von CP/M 2.0. Ferner ist es mit CP/M 2.2 "lediglich" möglich, 64 KBytes zu adressieren, wogegen Sie mit CP/M 3.0 128 KBytes adressieren können. Weiterhin ist CP/M 3.0 etwa komfortabler als sein älterer Bruder. Selbstverständlich können Sie auf Ihrem CPC 6128 auch CP/M 2.2 benutzen - weshalb wir im Floppybuch ausschließlich auf diese etwas "einfachere" CP/M-Version eingehen wollen. Eine zusätzliche Erläuterung zu CP/M 3.0 würde den Rahmen dieses Buches sicherlich sprengen. Wenn wir im folgenden von der *Systemdiskette* sprechen, so meinen wir die CP/M-2.2-Systemdiskette.

Führen Sie nun die Systemdiskette so in die Floppy-Station ein, daß ihr Etikett von außen zu lesen ist und der Pfeil mit der Beschriftung "CP/M" nach oben zeigt. Jetzt geben Sie ICPM ein.

(Anmerkung: I steht für den senkrechten Balken, den Sie erhalten, indem Sie Shift und die @-Taste gleichzeitig betätigen.)

Nun wird CP/M von der Diskette in den Rechner geladen.

Sie erhalten die folgende Meldung:

```
CPM 2.2 - Amstrad Consumer Electronics plc.
A>
```

Ab sofort sind alle BASIC-Kommandos unwirksam. Probieren Sie es ruhig aus, indem Sie beispielsweise einmal folgendes eingeben:

```
run
```

CP/M wird sich mit einem

```
RUN?
```

zurückmelden, was soviel bedeutet, wie "Das Kommando RUN kenne ich aber nicht". Sie haben sicher schon bemerkt, daß in der ersten Spalte ein "A>" steht. Dies ist das sogenannte *Promptsymbol*. Es wird Ihnen hier angezeigt, daß der Computer Ihre Befehle erwartet, und daß auf Laufwerk A geschaltet ist. Wenn Sie nur ein Laufwerk besitzen, so werden Sie immer diese Meldung erhalten, wenn Sie Besitzer von zwei Laufwerken sind, so gibt es noch das Promptsymbol B>.

Doch versuchen Sie es einmal mit einen für CP/M verständlichen Befehl:

```
dir
```

Sie werden augenblicklich das Disketteninhaltsverzeichnis auf den Bildschirm bekommen. dir steht hier für das englische Wort für Inhaltsverzeichnis, *Directory*.

1.3.3 Formatierung von Disketten

Formatieren bedeutet, eine rohe Diskette für das Diskettenlaufwerk zu präparieren. Eine unformatierte Diskette ist für die Floppy wie ein Blatt Papier, auf dem mit weißer Tinte geschrieben wurde.

Um eine Diskette zu formatieren, müssen Sie unter CP/M folgenden Befehl eingeben:

```
format
```

Auf dem Bildschirm erscheint:

```
Please insert disc to be formatted into drive A
then press any key
```

Entfernen Sie nun die CP/M-Diskette, und legen Sie die Diskette ein, die Sie formatieren wollen. Beim Formatieren werden alle eventuell auf einer Diskette gespeicherten Informationen gelöscht, d.h. wenn Sie eine Diskette formatieren, die schon formatiert war und auf der Sie Programme o.ä. gespeichert haben, so sind die Programme verloren. Darum ist beim Formatieren Eile mit Weile geboten, denn haben Sie erst einmal eine Diskette formatiert, ist das Kommando nicht mehr zu widerrufen.

Wenn Sie die Disketten gewechselt haben, so betätigen Sie irgendeine beliebige Taste. Die Formatierung beginnt sofort. Jede Diskettenseite wird mit 40 *Spuren*, die konzentrisch um das Führungsloch angeordnet sind, formatiert, von Spur 0 bis Spur 39. Spur 0 liegt außen, Spur 39 innen. Neben den Spuren gibt es aber noch weitere Unterteilungen der Diskette, die sogenannten *Sektoren*. Die Diskette ist zusätzlich unterteilt in 9 Sektoren, vergleichbar mit den Stücken einer Torte.

Man kann nur Sektor für Sektor von der Diskette lesen. Die Aufteilung in Spuren und Sektoren findet man bei den meisten Diskettensystemen.

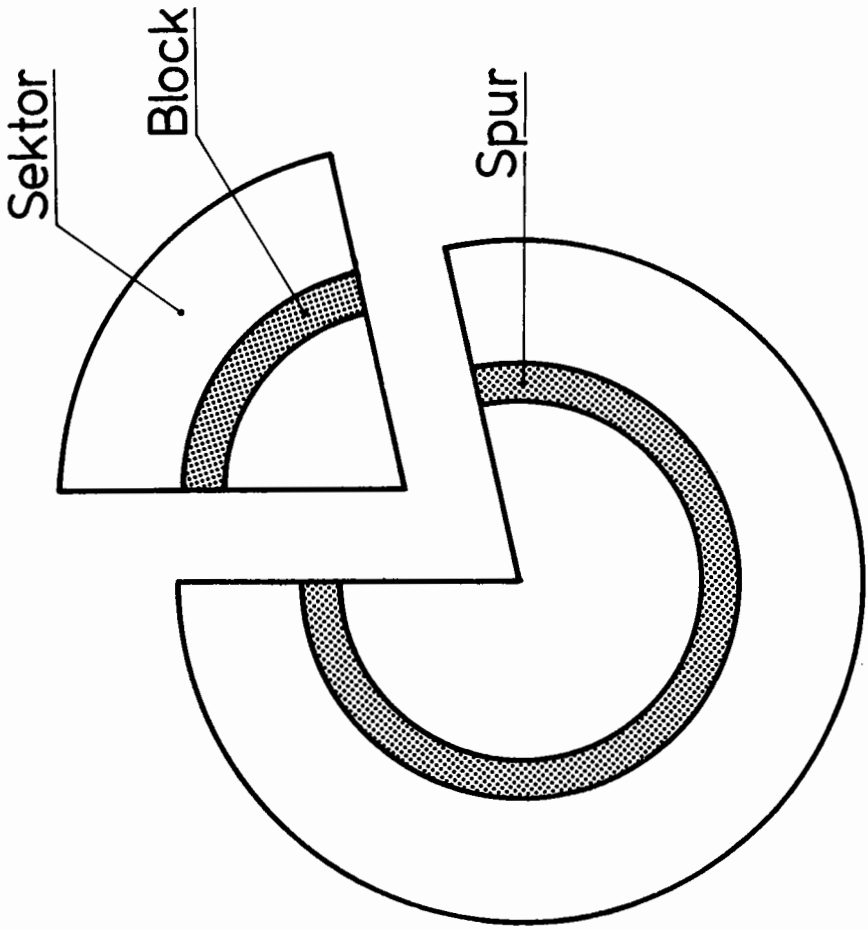


Abb.2 Spuren, Blöcke und Sektoren

Generell muß man folgende Begriffe unterscheiden: **Spur, Sektor, Block und Record.**

Eine Spur ist, wie beschrieben, ein vierzigstel der Diskette. Jede Spur hat 9 Sektoren, von denen jeder 512 Bytes umfaßt.
Für CP/M gibt es jetzt aber noch eine weitere Unterteilung, die sogenannten *Records*. Ein Record umfaßt nicht mehr und nicht weniger als 128 Bytes. Jeder Sektor umfaßt also 4 Records. Diese Unterteilung ist aus Kompatibilitätsgründen zu CP/M nötig.

AMSDOS kennt dann eine weitere, letzte Unterteilungsmöglichkeit in *Blöcke*. Ein Block umfaßt 1024 Bytes, besteht also aus 2 Sektoren. Ein Block ist die kleinste ansprechbare Einheit in BASIC.

Doch nun zurück zu unserem eigentlichen Thema, dem **Formatieren.**

Nachdem also alle 40 Spuren formatiert wurden, meldet sich das System wieder mit:

```
Do you want to format another disc (Y/N):
```

Wenn Sie also beispielsweise die andere Seite der Diskette formatieren wollen, so antworten Sie mit **y** für "Yes(=Ja)"; Sie können aber auch jede andere gewünschte Diskette formatieren.

Die Formatierung kann beliebig oft wiederholt werden, bis auf die Wiederholungsfrage mit **n** für "No(=Nein)" geantwortet wird. Daraufhin wird das System Sie auffordern:

```
Please insert a CP/M system disc into drive A
then press any key:
```

Sie können jetzt eine beliebige Taste betätigen, müssen also nicht zuvor die Diskette wechseln, denn das System hat beim

Formatieren auf den äußeren zwei Spuren das System CP/M abgelegt (nicht aber die Kommandofiles wie z.b. den Befehl **format**).

Wenn Sie eine Diskette formatieren wollen, so darf diese nicht schreibgeschützt sein. Anderenfalss erhalten Sie die Meldung:

```
Drive A: disc is write protected
Retry, Ignore or Cancel?
```

Diese Meldung teilt Ihnen mit, daß wegen des Schreibschutzes nicht ordnungsgemäß formatiert werden kann. Es gibt noch ähnliche Fehlermeldungen, bei denen Sie ebenfalls auswählen können zwischen:

a) Retry Wenn Sie es noch einmal versuchen wollen, drücken Sie die Taste "r".

b) Ignore Wollen Sie die Fehlermeldung der Floppy ignorieren, wird im Arbeitslauf fortgeschritten. Betätigen Sie hierzu die "i"-Taste. Dies ist allerdings nur selten ratsam, da dann bei den folgenden Kommandos unerwartete Effekte auftreten können.

c) Cancel Mit Betätigung der Taste "c" wird der Arbeitsablauf abgebrochen. Dies sollten Sie auch in unserem Falle tun und erst einmal, wenn geboten, den Schreibschutz entfernen.

1.3.4 Kopieren unter CP/M

Es ist sehr einfach, unter CP/M den Inhalt einer ganzen Diskette auf eine andere zu kopieren. Wenn nur ein Diskettenlaufwerk zur Verfügung steht, so geben Sie das Kommando

disccopy

ein. Sollten Sie zwei Diskettenlaufwerke haben, so können Sie den Befehl copydisc verwenden, der schneller arbeitet.

Nachdem Sie **disccopy** eingegeben haben, meldet sich der Computer mit:

```
Please insert source disc into drive A
then press any key
```

Sollten Sie allerdings die Meldung

```
FILECOPY?
```

auf dem Bildschirm erhalten, so bedeutet dies, daß Sie nicht die CP/M-Diskette im Laufwerk haben.

Nehmen Sie nun die CP/M-Diskette aus dem Laufwerk heraus. Wollen Sie aber die CP/M-Diskette selbst kopieren, was Sie unbedingt mindestens einmal tun sollten, so lassen Sie die CP/M-Diskette im Laufwerk. Betätigen Sie nun eine beliebige Taste.

```
Copying started
Reading track 0 to 7
```

Nun liest der Rechner die ersten 8 Spuren (= tracks) in den Speicher. Wenn dies getan ist, erhalten Sie die Meldung:

```
Please insert destination disc into drive A
then press any key
```

Entnehmen Sie nun die Quelldiskette dem Laufwerk und legen Sie die Diskette ein, auf die die Files kopiert werden sollen. Sollte Ihre Zieldiskette noch nicht oder fehlerhaft formatiert

sein, so wird diese vorher formatiert. Auch werden alle auf der Diskette befindlichen Daten überschrieben, da eine vollständige Kopie der Quelldiskette angefertigt wird. Man kann theoretisch und praktisch die Kopie nicht mehr vom Original unterscheiden.

Nach dem Einlegen der Zieldiskette und Betätigung einer beliebigen Taste erhalten Sie die Mitteilung:

```
Writing track 0 to 7
```

Denselben Arbeitsgang wiederholen Sie für die Spuren (tracks) 8 bis 15, 16 bis 23, 24 bis 31 und 32 bis 39. Dann ist die Diskette fertig kopiert, woraufhin Sie die Meldung

```
Do you want to copy another disc (Y/N):
```

erhalten. Wenn Sie keine weitere Diskette kopieren wollen, so geben Sie N ein und folgen den Anweisungen auf dem Bildschirm. Bei einer weiteren Kopie sieht der Arbeitsgang exakt gleich aus.

1.3.5 Kopieren mit zwei Laufwerken

Den Befehl **COPYDISC** können Sie nur verwenden, wenn Sie zwei Laufwerke haben. Der Arbeitsverlauf ist ähnlich dem des vorgehend beschriebenen Kommandos **DISCCOPY**. Der Vorteil des Befehles ist aber, wie bereits erwähnt, daß Sie nicht immer die Quell- und Zieldisketten zu wechseln brauchen.

Auch bei dem Kommando **COPYDISC** wird, falls nötig, automatisch die Zieldiskette formatiert. Ansonsten müssen Sie lediglich den Anweisungen auf dem Bildschirm folgen.

1.3.6 DISCCHK - Überprüfen von Disketten

Nach Erstellung einer Kopie können Sie auch überprüfen, ob alles gut kopiert worden ist; es wäre ja ärgerlich, wenn Ihre Originaldiskette einmal kaputt geht, und Sie dann feststellen müssen, daß Ihre Kopie auch nicht in Ordnung ist, (was immer mal passieren kann). Um Quell- und Kopiediskette zu überprüfen, geben Sie das Kommando

```
discchk
```

ein. Folgen Sie dann den Kommandos auf dem Bildschirm, die Sie auffordern, jeweils die Original- und Zieldiskette einzulegen. Wenn Unterschiede zwischen Original- und Zieldiskette festgestellt werden, so erhalten Sie die Meldung

Failed to verify destination disc correctly:
track x sector y

Es wird trotzdem im Vergleich der beiden Disketten fortgefahren, so daß eventuelle weitere Unstimmigkeiten ebenfalls angezeigt werden. Es werden immer 8 Spuren nacheinander verglichen. Wenn Sie zwei Laufwerke haben, so geht der Vergleich deutlich schneller und einfacher. Benutzen Sie dann das Kommando

```
chkdisc
```

Sie brauchen nun nur noch Quell- und Kopiediskette entsprechend den Anweisungen auf dem Bildschirm einzulegen, und es werden die beiden Disketten genauso wie beim Kommando **discchk** verglichen.

Sie können übrigens alle CP/M-Kommandos abbrechen, indem Sie [Ctrl] und die Taste "C" gleichzeitig niederhalten. Wenn Sie beispielsweise versehentlich den Befehl **FORMAT** eingegeben

haben, so kommen Sie mit [Ctrl]-C wieder in den Eingabestatus von CP/M.

Weitere Kontrollfunktionen sind hier aufgelistet :

[Ctrl] C	Abbrechen des aktuellen Befehls.
[Ctrl] S	Hält die Bildschirmausgabe an. Durch Drücken einer beliebigen Taste können Sie mit der Ausgabe fortfahren.
[Ctrl] P	Es wird auf Druckerausgabe umgeschaltet, d.h. die Bildschirmausgabe kommt auf den Drucker.
[Ctrl] Z	Textende. Wird beispielsweise bei Texteingaben verwandt.

1.3.7 Spezielle Formatierungsformate

Wenn Sie beispielsweise eine reine Datendiskette formatieren wollen, oder wenn Sie eine Diskette ausschließlich zum Abspeichern von BASIC-Programmen benötigen, so müssen Sie das Betriebssystem CP/M nicht mitkopieren. CP/M wird auf den Spuren 0 und 1 abgelegt. Auf einer Datendiskette brauchen Sie aber kein CP/M, so daß Sie zwei Spuren an Platz verschenkt hätten. Um dies zu umgehen, kann man sich eines besonderen Kommandos bedienen:

```
format d
```

Wenn Sie dieses Kommando statt **FORMAT** eingeben, so wird die Diskette ohne das Betriebssystem CP/M formatiert. Sie haben also mehr Raum für Daten und/oder Programme zur Verfügung.

Es existieren folgende Formatierungsmöglichkeiten:

format	Formatierung mit Kopieren von CP/M.
format d	Formatierung in Datenformat ohne CP/M
format i	Formatierung in IBM-Format
format v	Formatierung in Vendor-Format

Es wird jeweils die im Standardlaufwerk befindliche Diskette formatiert. Leider ist es nicht möglich, eine Diskette, die sich in einem anderen Laufwerk befindet, zu formatieren.

Das für Sie gebräuchlichste Format ist das erste, wenn Sie mit CP/M bzw. das zweite, wenn Sie fast nur unter AMSDOS in BASIC arbeiten. Hier werden 9 Sektoren pro Spur formatiert. Im Systemformat werden die Sektoren mit #41 bis #49 (hexadezimal) durchnumeriert. Die reservierten Spuren werden wie folgt belegt:

Spur 0,Sektor #41	Boot-Sektor
Spur 0,Sektor #42	Konfigurations-Sektor
Spur 0,Sektor #43-#47	unbenutzt
Spur 0,Sektor #48,#49	sowie:
Spur 1,Sektor #41-#49	CCP und BDOS

CCP=Console Command Processor.
BDOS=Basic Disk Operating System.

Ferner befindet sich auf den ersten Sektoren der Spur zwei das Inhaltsverzeichnis der Diskette. Wie die ersten Blöcke einer Diskette belegt sind, können Sie dieser Grafik entnehmen:

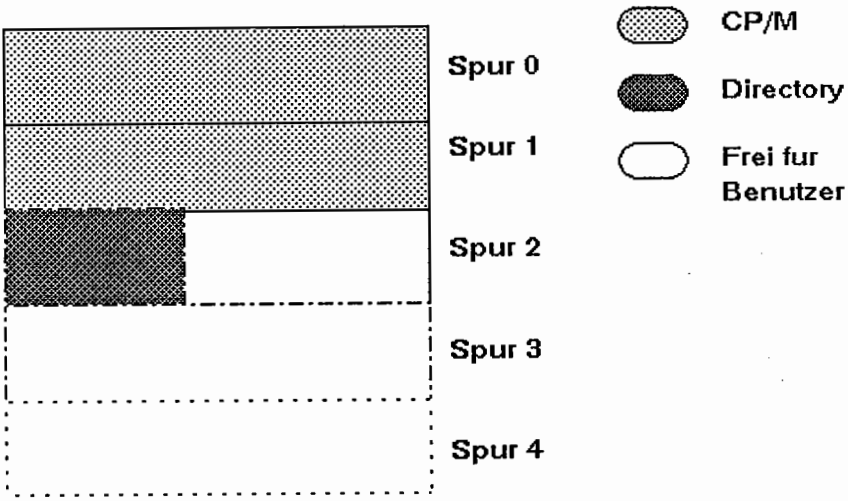


Abb. 3 Belegung Diskette

Das Vendor-Format ist dem System-Format gleich, es wird lediglich das CP/M-Betriebssystem nicht mitkopiert. Dieses Format wird beim Software-Vertrieb verwendet, da man CP/M nicht mitverkaufen darf.

Das DATA-ONLY-Format

Die (physikalische) Formatierung ist dieselbe wie beim System-Format. Zum Formatieren einer Datendiskette verwenden Sie das FORMAT D-Kommando. Es werden 40 Spuren mit je 9 Sektoren formatiert. Die Sektoren werden mit #C1 bis #C9 durchnumeriert. CP/M wird nicht mitkopiert, so daß die oberen beiden Spuren, Spur 0 und Spur 1, dem Benutzer zur Verfügung stehen. So haben Sie 2*9*512=9216 Bytes mehr zur Verfügung.

Das IBM-Format

Es wird mit 8 Sektoren (#1-#8) pro Spur formatiert. Eine Spur wird reserviert. Dieses Format ist logisch dasselbe Format wie das beim IBM-PC unter CP/M verwendete Disketten-Format.

Dieses Format sollte man nur zum speziellen Gebrauch verwenden (beispielsweise um Daten zu IBM-Rechnern zu übertragen o.ä), es wird ansonsten nicht empfohlen.

1.3.8 Transiente und residente Befehle

Man muß bei CP/M unterscheiden zwischen den sogenannten *transienten* und den *residenten* Befehlen. Die residenten Befehle stehen automatisch beim Booten (d.i. das Hochfahren des Systems) von CP/M zur Verfügung. Sie residieren also förmlich im Speicher wie früher die Könige in Schlössern.

Es handelt sich dabei um folgende Befehle:

SAVE, A:, B:, DIR, ERA, REN, USER und TYPE.

Der weitaus größere Teil gehört zu der Gruppe der transienten Befehle. Transiente Befehle müssen, bevor der Rechner sie ausführen kann, erst von der Diskette nachgeladen werden. Es handelt sich hierbei um die Befehle:

AMSDOS, BOOTGEN, CHKDISK, CLOAD, COPYDISK, CSAVE, DISCCKK, DISCCOPY, FILECOPY, FORMAT, SETUP und SYSGEN.

Diese Befehle arbeiten ausschließlich auf dem Schneider CPC, andere Anbieter verwenden aber ähnliche Hilfsprogramme unter ähnlichen oder gleichen Namen. Weitere standardisierte transiente CP/M-Kommandos sind:

ED, MOVCPM, PIP und STAT.

Sie finden diese Files auf Ihrer CP/M-Systemdiskette. Die Filenamen werden mit einem .COM markiert. Beispielsweise können Sie das File FILECOPY.COM auf Ihrer Diskette vorfinden

(durch das DIR-Kommando). Die transienten Befehle stehen nicht zur Verfügung, wenn sie nicht mit DISCCOPY, COPYDISC, FILECOPY oder PIP kopiert worden sind.

Um eine schlüsselfertige Kopie Ihrer CP/M-Diskette zu haben, müssen Sie also entweder mit

DISCCOPY (für Computer mit einem Laufwerk) oder
COPYDISC (für Besitzer von zwei Laufwerken)

kopieren, oder Sie kopieren mit

FORMAT und dann FILECOPY *.*

alle transienten Files. Eine weitere Möglichkeit ist es, daß Sie nur die transienten Files kopieren, die Sie am häufigsten benötigen. So könnten Sie beispielsweise die Befehle PIP, COPYDISC und CHKDISC weglassen, wenn Sie nur ein Laufwerk haben, so daß Ihnen mehr Platz auf Ihrer Diskette bleibt.

1.3.9 Kopieren von Files mittels Filecopy

Um einzelne Files zu kopieren, bedienen Sie sich des Kommandos

```
FILECOPY
```

Sollten Sie allerdings zwei Laufwerke haben, so können Sie sich auch des CP/M-Befehls PIP bedienen.
Wenn Sie alle Files einer Diskette kopieren wollen, so geben Sie

```
FILECOPY *.*
```

ein. Ein Filename besteht aus zwei Teilen, die durch einen Punkt "." voneinander getrennt sind. Der erste Teil ist der eigentliche Dateiname, der zweite Teil die Dateibezeichnung. Der erste Teil darf bis zu 8 Zeichen lang sein, der zweite bis zu

3 Zeichen. Diese Regel gilt nicht nur unter CP/M auf Ihrem DDI-1 Laufwerk, sondern auch unter AMSDOS. Folgende Typenbezeichnungen gibt es beispielsweise:

Nicht näher definierter Typ. Es könnte sich beispielsweise um ein in BASIC erstelltes File handeln, das ohne Dateitypen mittels OPENOUT geöffnet worden ist.

.BAS Hier handelt es sich um ein BASIC-Programm, das durch das BASIC-Kommando SAVE "Programm" oder SAVE"Programm",P oder SAVE"Programm.bas",a gesichert worden ist.

.BIN Speicherung eines Speicherbereichs oder eines Maschinenprogramms.

.BAK Hier handelt es sich um ein sogenanntes Backup, eine Sicherheitskopie. Wenn Sie beispielsweise ein Programm unter AMSDOS abspeichern und einen Namen verwenden, unter dem bereits ein Programm abgespeichert war, so erstellt AMSDOS automatisch eine Backup-Datei. Dies geschieht aus Sicherheitsgründen, damit Sie nicht versehentlich eine Datei überschreiben. Wenn Sie aber ein weiteres Mal das Programm unter demselben Namen abspeichern, so wird die alte Backup-Datei gelöscht, das letzte File wird zum Backup und Ihr neues Programm wird unter "Name.Bas" abgespeichert.

.COM Hier handelt es sich um Befehlsdateien, auf denen Befehle gespeichert sind. So sind alle Hilfsprogramme von CP/M von diesem Dateityp.

.DAT Es handelt sich um eine Datei, die beispielsweise von einem Dateiverwaltungsprogramm

(wie in diesem Buch) unter AMSDOS erstellt worden ist.

.SEQ — Das File ist vom Typ "sequentiell". Das Thema der sequentiellen Dateien wird später noch genauer abgehandelt.

Des weiteren ist jede andere Dateibezeichnung denkbar, da hier jede erdenkliche drei-stellige Buchstabenkombination erlaubt, d.h. frei wählbar ist. Die hier aufgeführten Dateitypen sind allerdings die am häufigsten verwendeten.

Es gibt im Grunde nur zwei Speicherungsarten, auf denen alle anderen basieren. Dies ist

a) die sequentielle und
b) die relative Datenspeicherung.

Standardmäßig verfügt die DDI-1 lediglich über die einfachere sequentielle Datenspeicherung. Was Sie unter *relativer Datenspeicherung* zu verstehen haben, und wie sie funktioniert, wird im Kapitel 1.6 erklärt.

1.3.10 Die Wild Cards

Die im Kommando FILECOPY *.* verwendeten Sternchen (*) sind sogenannte *Wild Cards* (= Wilde Karten). Diese Wild Cards können einem häufig das Leben erleichtern. Was aber bitte sind Wild Cards? Nun, es gibt zwei Formen von Wild Cards, die Sternchen und das Fragezeichen. Wenn Sie ein File mit dem Kommando FILECOPY MATHE.BAS kopieren, so lädt das Programm FILECOPY das File MATHE.BAS in den Speicher und schreibt es nachher auf die Zieldiskette mit exakt demselben Namen. Sie haben also genau spezifiziert, welches File kopiert werden sollte. Nun gibt es aber oft Anwendungen, bei denen Sie viele verschiedene Files mit ähnlichem Namen kopieren (bzw. ansprechen) wollen. Nehmen wir einmal an, Sie wollten ausschließlich alle Basic-Programme kopieren, also alle Programme, die mit einem .BAS enden. Dafür müßten Sie sich das Inhaltsverzeichnis ansehen, alle Files notieren, die mit einem .BAS enden und diese nacheinander mit dem Kommando FILECOPY Name.BAS kopieren. Das ist sicherlich sehr umständlich. Um dies zu vermeiden, gibt es zum Glück die Wild Cards, die Sie nicht nur unter CP/M finden.

In unserem Beispiel müßten Sie nur noch folgendes Kommando geben:

```
FILECOPY *.BAS
```

und alle BASIC-Programme würden kopiert. Das Sternchen bedeutet, "egal wie der erste Teil des Files lautet, wenn es sich um ein BASIC-Programm handelt, dann kopiere".

Es werden also alle BASIC-Programme kopiert.

Es ist auch möglich, ein Sternchen erst nach einem oder mehreren Zeichen anzuhängen. Beispielsweise bedeutet:

```
FILECOPY f*.*
```

kopiere alle Files, die mit "f" beginnen, egal welchen Dateityp sie haben. Die Files

```
"faulenze.bas"
"frau.seq"
```

würden somit anstandslos kopiert. Jetzt gibt es noch eine weitere Wild Card, das Fragezeichen. Ein Fragezeichen an beliebiger Stelle ersetzt genau einen Buchstaben.

So hätte man auch anstatt FILECOPY f*.* schreiben können FILECOPY f???????.??? oder FILECOPY f*.??? oder aber auch FILECOPY f????????.*. Wenn Sie beispielsweise einen Terminkalender auf Diskette gespeichert haben, so könnten Sie sich mit dem Befehl

```
FILECOPY 05-??.*
```

alle Termine des Monats Mai in den vergangenen Jahren kopieren lassen, vorausgesetzt, das Format des Filenamen wäre entsprechend.

FILECOPY 1?1 würde alle Files kopieren, die an erster und an dritter Stelle die Ziffer eins haben, in der Mitte wären alle erlaubten Zeichen denkbar.

Bei der Verwendung des Kommandos FILECOPY *.* werden Sie gefragt, ob Sie alle Files kopieren oder ob Sie selektieren wollen. Es wird Ihnen Filename für Filename angezeigt, und Sie können entscheiden (mit *y* für ja und *n* für nein), ob das jeweilige kopiert werden soll.

1.3.11 Das DIR-Kommando

Mit dem DIR-Kommando können Sie sich das Disketten-inhaltsverzeichnis ausgeben lassen. Ferner können Sie dabei eine Selektion betreiben, d.h. Sie können bestimmte Files ausschließen. Dies erreicht man durch die Benutzung von Wild

Cards. Beim Weglassen von Parametern wird *.* angenommen. Das DIR-Kommando hat folgende Auswirkungen:

DIR	Zeigt alle Dateien an
DIR B:	Zeigt alle Dateien von Laufwerk B: an.
DIR *.BAS	Zeigt nur die *BAS*IC-Files an
DIR FILE.COM	Zeigt nur das File FILE.COM an, sofern vorhanden.

Die Files werden in der Reihenfolge angezeigt, in der sie auch eingerichtet worden sind, d.h. das erste File, das man auf Diskette gespeichert hat, wird auch als erstes im Inhaltsverzeichnis aufgeführt.

1.3.12 Das ERA-Kommando

Mit dem Kommando ERA (für *ERA*se) können Sie Files auf Diskette löschen. Es werden allerdings nur die Einträge auf der Diskette gelöscht, die Daten des Files jedoch bleiben erhalten - allerdings kann man nicht mehr darauf zugreifen. Sie können die Wild Cards benützen, um mehrere Files zu löschen. Wenn Sie *.* angeben, so muß der Befehl bestätigt werden, da alle Einträge der Diskette gelöscht würden. Sollte eine Datei gefunden werden, die nur lesbar ist (siehe auch unter 1.3.16), so wird der Befehlsablauf unterbrochen.

Mögliche Kommandos:

ERA DISCCOPY.COM	Löscht das File DISCCOPY.COM
ERA *.SEQ	Löscht alle Dateien mit Dateityp SEQ

1.3.13 Das REN-Kommando

Mit dem Kommando REN (für *REN*ame) werden Dateien umbenannt.

```
REN neuer Name=alter Name.
```

Wenn entweder der neue Name schon existiert oder der alte Name nicht existiert, so wird eine Fehlermeldung ausgegeben.

1.3.14 Das TYPE-Kommando

Mit dem TYPE-Befehl können Sie den Inhalt von Files auf dem Bildschirm ausgeben lassen. Wenn es sich nicht um ASCII-Files handelt, wie etwa bei BASIC-Programmen, so können Grafiksymbole und ähnliches auf dem Bildschirm erscheinen. Des weiteren könnte sich beispielsweise die Hintergrundfarbe oder der Anzeigemodus ändern.

`TYPE EX1.BAS` Darstellen des Beispielprogramms EX1

1.3.15 Umschalten des Standardlaufwerks

Mit den Kommandos A: und B: können Sie, sofern Sie zwei Laufwerke besitzen, zwischen den beiden Laufwerken hin- und herschalten.

B: Laufwerk B ist Standardlaufwerk.

Entsprechend wird auch das Promptsymbol geändert.

1.3.16 Das PIP-Kommando

Mit dem PIP-Kommando können Sie den Transfer zwischen Computer und Peripherie erstellen, oder Sie benutzen das Kommando zum Kopieren, falls Sie zwei Laufwerke besitzen.

Die Syntax lautet:

```
PIP <Ziel>=<Quelle>.
```

Die *<Quelle>* und das *<Ziel>* können Files oder Peripheriebausteine sein. Sie können folgende Peripherie-Einheiten ansprechen:

Als Quelle:

CON:	Konsole = Tastatur
RDR:	Serielles Interface

Als Ziel:

CON:	Konsole = Bildschirm
PUN:	serielles Interface
LST:	Drucker

Beispiel:

Sie wollen ein File erstellen, das Sie per Tastatur eingeben wollen:

```
PIP Text.Txt=CON:
```

Es wird dann alles, was Sie eingeben, auf das File Text.Txt gespeichert, bis Sie die Taste **[Ctrl]-Z** betätigen. Dann wird das File geschlossen.

Sie können das Kommando PIP nicht zum Kopieren von Dateien mit einem Laufwerk verwenden. Bedienen Sie sich dann des Kommandos FILECOPY.

Weitere Beispiele:

PIP LST:=EX2.BAS	Gibt das BASIC-File EX2.BAS auf Drucker aus.
PIP CON:=EX2.BAS	Gibt das BASIC-File EX2.BAS auf Bildschirm aus. Dieser Befehl ist ähnlich dem Kommando TYPE EX2.BAS

1.3.17 Das STAT-Kommando

STAT steht für Status. Sie können mit dem STAT-Kommando einerseits eine komfortable Ausgabe von Dateiinformationen erzielen, ähnlich dem DIR-Kommando.

```
STAT
STAT B:
STAT *.BAS
```

wären Beispiele dafür. Andererseits bietet Ihnen dieses Kommando eine sehr wichtige und praktische Option. Sie können beispielsweise ein File "nur lesbar" machen, d.h. Sie können dieses File nicht mehr versehentlich löschen oder überschreiben. Mit dem Kommando

```
STAT *.BAS $R/O
```

erreichen Sie, daß alle BASIC-Files mit dem Read-Only-Status versehen werden. Versehentliches Löschen oder Überschreiben dieses Files ist ab sofort ausgeschlossen.

Zur Aufhebung dieses Status fügen Sie die Zeichen für Schreib/Lesestatus ($R/W) an den Ausgangsbefehl. Zum Aufheben des o.g. Kommandos geben Sie konkret ein:

```
STAT *.BAS $R/W
```

Sie können weiterhin ein File mit dem System-Status versehen. Dann wird dieses File beim DIR-Kommando nicht mehr aufgelistet, es ist praktisch unsichtbar. Weiterhin kann dieses File auch nicht mehr kopiert werden. Lediglich im STAT-Kommando wird es noch aufgelistet. Wenn Sie beispielsweise folgendes eingeben:

```
STAT *.COM $SYS
```

so werden alle COM-Files mit dem Systemstatus versehen, Sie können diese Files also nicht mehr mit dem DIR-Kommando auflisten und nicht kopieren. Allerdings können diese Kommandos weiterhin aufgerufen werden.

Die Umkehrung dieses Befehls lautet:

```
STAT *.COM $DIR
```

Dieses Kommando setzt den "Inhaltsverzeichnis-Status".

Eindrucksvolle STAT-Kommandos sind:

STAT dsk: Es werden Ihnen alle wichtigen Informationen über das Diskettenformat gegeben.

STAT val: Gibt Ihnen eine Liste der Abkürzungen aus, und wie diese momentan belegt sind. Beispielsweise CON:=TTY etc.

oder **STAT dev:** und **STAT usr:**

1.3.18 BOOTGEN

Mit BOOTGEN ist es möglich, die beiden Spuren 0 und 1 auf eine andere Diskette zu kopieren. Dieses Kommando können Sie verwenden, wenn Sie eine im Vendor-Format formatierte Diskette mit CP/M versehen bzw. einen neuen Konfigurationssektor auf mehrere Disketten kopieren wollen.

1.3.19 MOVCPM und SYSGEN

Mit dem Kommando **MOVCPM** können Sie CP/M in einen anderen Speicherbereich verschieben. Oft ist dies notwendig, weil sich CP/M mit irgendwelcher Software überschneidet. Sie können CP/M in 256-Bytes-Schritten verschieben. Die Syntax lautet

```
MOVCPM <Größe>*
```

Die Größe hat Werte zwischen 64 bis 179. Das standardmäßige CP/M ist mit der Größe 179 erzeugt worden. Mit MOVCPM 178* beispielsweise verschieben Sie CP/M um 256 Bytes nach unten.

Darauf folgend können Sie das neu erstellte CP/M entweder mit dem Kommando SYSGEN sichern, oder Sie sichern dieses CP/M unter einer Datei. Diese Möglichkeit wird Ihnen ebenfalls unter MOVCPM gegeben.

Mit SYSGEN schreiben Sie dann das Ergebnis eines MOVCPM-Kommandos auf die System-Spur. Hierbei gibt es dann noch drei unterschiedliche Kommandos:

```
SYSGEN*
```

Dieses Kommando schreibt das durch ein unmittelbar vorangehendes MOVCPM-Kommando erzeugte CP/M auf die System-Spuren.

SYSGEN <Dateiname>

Mit diesem Kommando wird das mit MOVCPM erstellte File unter dem Namen *<Dateiname>* gelesen und auf die System-Spuren kopiert. Beispiel: SYSGEN CPMEXTRA.COM

SYSGEN

Ohne Parameter werden Quell- und Zieldiskette erfragt und entsprechend das CP/M auf die Zieldiskette kopiert. Mit diesem Befehl können Sie auf eine VENDOR-Diskette (siehe auch unter FORMAT V) CP/M kopieren.

1.3.20 SETUP

Mit diesem Kommando können Sie ganz entscheidend das Aussehen und den Ablauf des CP/M verändern. Dieses Kommando erlaubt Ihnen Eingriffe in die Tastaturbelegung, in den Diskettenzugriff und vieles mehr. Sie sollten dieses Kommando nur dann verwenden, wenn Sie sich ausreichend über die Auswirkungen informiert haben. In Kapitel 3.7.3.2 und Kapitel 1.5 des Benutzerhandbuches zur DDI-1-Floppy ist der Arbeitsablauf entsprechend dargelegt.

1.3.21 AMSDOS

Mit dem Kommando AMSDOS schalten Sie CP/M ab und geraten ins AMSDOS. Sie können jetzt wie gewohnt in BASIC programmieren. Es stehen Ihnen die AMSDOS-Befehle zur Verfügung.

1.4 Arbeiten mit AMSDOS

1.4.1 Was ist AMSDOS?

AMSDOS steht für "*AM*Strad *D*isc *O*perating *S*ystem" und unterstützt das Arbeiten mit der Diskette unter BASIC. AMSDOS ist nur dann aktiv, wenn das Diskettenlaufwerk am Rechner angeschlossen und eingeschaltet ist - dies ist beim CPC 664 und dem CPC 6128 natürlich immer der Fall.

Beachten Sie beim Einschalten unbedingt die Reihenfolge! Es muß zuerst die Floppy eingeschaltet werden, dann erst dürfen Sie den Monitor und den Rechner einschalten. Das hat den Grund, daß beim Einschalten des Rechners automatisch getestet wird, welche Peripherie (Drucker, Floppy etc.) eingeschaltet ist. Sollte die Floppy also noch aus sein, wenn Sie den Rechner einschalten, so werden spätere Befehle, die an die Floppy gehen sollen, trotzdem an das Kassettenlaufwerk gesendet. Dies gilt natürlich nur für den CPC-464-Besitzer, der CPC 664 und der CPC 6128 erkennt natürlich sofort das eingebaute Diskettenlaufwerk.

Alle im folgenden aufgelisteten Befehle, die beim CPC 464 sonst an die Kassette gehen, werden an die Floppy gesandt, wenn keine entsprechenden anderen Befehle gegeben werden:

```
load "Filename"
run "Filename"
chain "Filename"
merge "Filename"
chain merge "Filename"
save "Filename"
openin "Filename"
closein
openout "Filename"
closeout
```

```
cat
eof
input #9
line input #9
print #9
write #9
list #9
```

Der Befehl SPEED WRITE bezieht sich (beim CPC 464) aber nach wie vor auf das Kassettenlaufwerk, da beim Diskettenlaufwerk Schreib- bzw. Lesegeschwindigkeiten nicht variabel sind.

Zusätzlich zu den oben aufgelisteten - im "normalen" BASIC vorhandenen Kommandos - gibt es noch zusätzlich die folgenden sogenannten *externen* Befehle.

Es handelt sich hier um externe Befehle, da die Befehle im ROM des AMSDOS gespeichert sind, also in der Floppy. Diese Befehle können beim CPC 464 im Kassetten-BASIC überhaupt nicht verwendet werden, da sie erst beim Einschalten des Diskettenlaufwerkes zur Verfügung stehen. Die Befehle beginnen mit einem I (Shift & @), dieses Zeichen leitet alle sogenannten *RSX*-Kommandos ein, also BASIC-Erweiterungsbefehle. Die Floppy-Kommandos sind allerdings keine eigentlichen RSX-Kommandos, sondern bilden eine Ausnahme.

```
Ia
Ib
Idir
Idisc
Idisc.in
Idisc.out
Idrive
Iera
Iren
Itape
```

```
Itape.in
Itape.out
Iuser
```

Sicherlich erkennen Sie die Parallelen zu verschiedenen CP/M-Kommandos. Dies ist auch nicht weiter verwunderlich, da bei der Ausführung dieser RSX-Kommandos exakt dieselben Funktionen ausgeführt werden wie unter CP/M 2.2 oder CP/M 3.0. Die RSX-Kommandos werden noch einmal an speziellen Beispielen erläutert.

1.4.2 LOAD und RUN

Einer der wesentlichen Funktionen eines Diskettenlaufwerkes ist das Abspeichern und Laden von Programmen. Wenn Sie ein BASIC-Programm von Diskette laden wollen, sei es nun zum Starten oder zum Programmieren, so müssen Sie das Kommando

```
LOAD "Filename"
```

verwenden. Sie laden mit diesem Kommando BASIC-Programme von der im Laufwerk befindlichen Diskette.

Der Schneider-Rechner kennt verschiedene sogenannte *Dateitypen* (wir haben die verschiedenen Dateitypen bereits einmal kurz angesprochen). Verschiedene Dateitypen sind notwendig, um BASIC-Programme von Maschinenprogrammen zu unterscheiden, und um Programme generell von Dateien (beispielsweise eine Adreß-Datei) unterscheiden zu können. Dies dient nicht zuletzt auch Ihrer Übersicht; das Diskettenlaufwerk kann diese Unterscheidungen *intern* auch ohne Hilfe des im Dateinamen angegebenen Dateitypes treffen.

So wissen Sie, daß der Dateityp BAS ein BASIC-Programm sein muß. Der Dateityp BIN signalisiert ein Maschinenprogramm und der Dateityp SEQ verrät sequentielle Dateien.

Haben Sie das LOAD-Kommando eingegeben, so versucht AMSDOS zunächst das File "Filename. " zu laden. Wenn dieses File nicht existiert, so wird versucht, das File "Filename.BAS" zu laden. Sollte auch dieses File nicht auf Ihrer Diskette vorhanden sein, so wird noch ein Ladeversuch des Files "Filename.BIN" unternommen. Erst wenn auch dieses File nicht gefunden werden kann, wird die Fehlermeldung "File not Found" ausgegeben. Sie können aber auch

```
LOAD "Filename.BAS"
```

eingeben, um auszuschließen, daß AMSDOS ein eventuell vorhandenes File unter dem Namen "Filename. " lädt.

Sollten Sie glücklicher Besitzer eines zusätzlichen zweiten Laufwerkes sein, so wäre es natürlich schön, Programme auch von diesem Laufwerk in den Speicher des Rechners einzulesen. Um dies zu erreichen, ist es erforderlich, die Laufwerksangabe im Dateinamen anzugeben. Das erste (eingebaute) Laufwerk hat die Laufwerksbezeichnung **A:**, das zweite (externe) Laufwerk hat die Laufwerksbezeichnung **B:**. Will man nun ein Programm vom zweiten Laufwerk einladen, so sieht dieses Ladekommando wie folgt aus:

```
LOAD "B:Filename"
```

Wenn die Diskette gar nicht oder nicht korrekt im Laufwerk eingelegt ist, so erscheint die Fehlermeldung:

```
Drive A: disc missing
Retry, Ignore or Cancel?
```

auf dem Bildschirm. Legen Sie dann die Diskette korrekt ins Laufwerk ein und betätigen Sie die "R"-Taste für "*Retry*", also wiederholen.

Beachten Sie, daß es nicht erforderlich ist, den Dateityp näher zu spezifizieren. AMSDOS hängt automatisch die Endung ".BAS" beim Laden und Speichern von BASIC-Programmen an den Filenamen an. Wenn Sie allerdings ein Programm unter einem anderen Namen mit einer anderen Kennung als ".BAS" abgespeichert haben (was durchaus zulässig ist), so können Sie auch dieses File mit dem Kommando:

```
LOAD "Filename.xxx"
```

laden. Für *xxx* können Sie hier den entsprechenden Dateityp eintragen.

Sollte das File "Filename" allerdings nicht existieren (weil Sie beispielsweise den Filenamen falsch eingegeben haben oder die falsche Diskette sich im Laufwerk befindet), so erhalten Sie die Fehlermeldung:

```
Filename.xxx not found.
```

Überprüfen Sie dann den eingegebenen Filenamen und versuchen Sie es nötigenfalls noch einmal.

Erhalten Sie die Fehlermeldung *Type mismatch*, so deutet dies darauf hin, daß Sie vergessen haben, die Anführungszeichen am Anfang des Filenamens einzugeben.

Bei der Fehlermeldung

```
Drive A: read fail
Retry, Ignore or Cancel
```

ist ein Lesefehler auf der Diskette aufgetreten. Dann ist wahrscheinlich Ihre Diskette defekt (oder Sie haben eine falsche Diskette eingelegt). Es kann auch sein, daß Ihre Diskette nicht korrekt im Schneider-Format formatiert worden ist.

Die Meldung

Press PLAY then any key:

bedeutet, daß die Verbindung vom Rechner zum Interface bzw. der Floppy nicht einwandfrei ist. Weiterhin ist es möglich, daß Sie das Diskettenlaufwerk nicht als erstes eingeschaltet haben. Sollte diese Meldung bei einem CPC 664 oder einem CPC 6128 erscheinen, so wäre dies schon sehr bedenklich! Eine weitere Möglichkeit ist, daß Sie den Itape(.in) Befehl benutzt haben, geben Sie dann ein:

```
Idisc
```

und versuchen Sie es erneut. Sollten Sie dieselbe Fehlermeldung erhalten, so schalten Sie Ihren Rechner noch einmal aus und wieder ein.

Es gibt wie im Kassetten-BASIC die Möglichkeit, Programme nach dem Laden automatisch starten zu lassen. Man verwendet auch hierfür das Kommando:

```
RUN "Filename"
```

Es existiert nicht mehr die Möglichkeit wie im Kassetten-BASIC, das Kommando RUN " zu verwenden, da Sie den Filenamen des zu ladenden Files genau bestimmen müssen. Es gelten für den Befehl RUN "Filename" dieselben Fehlermeldungen und deren Konsequenzen wie unter LOAD.

Sie werden als Besitzer des CPC 464 gerade beim Laden von BASIC-Programmen merken, wie schnell Ihre Schneider-Floppy im Gegensatz zum Kassettenlaufwerk ist.

Genauso selbstverständlich können Sie auch Gebrauch von den anderen BASIC-Ladebefehlen machen. Wie Sie die Kommandos

```
CHAIN "Filename",
MERGE "Filename" und
CHAIN MERGE "Filename"
```

verwenden können, entnehmen Sie am besten dem Manual, da beim Ladevorgang verschiedene Dinge mit dem im Speicher befindlichen Programm passieren.

Eine sehr nützliche Möglichkeit ist, daß man unter AMSDOS wie unter CP/M sogenannte *User-Bereiche* definieren kann. USER kommt aus dem Englischen und steht für das deutsche Wort "Benutzer". Wenn Sie sich schon einmal ein Inhaltsverzeichnis angesehen haben, so haben Sie sicherlich festgestellt, daß die erste Zeile

```
Drive A: user 0
```

lautet. Sie können User-Nummern von 0 bis 15 definieren, Standard ist User-Nummer 0. Durch die User-Nummern können Sie verschiedene Inhaltsverzeichnisse ansprechen, d.h. Sie könnten beispielsweise unter User 0 nur die CP/M-Befehle ablegen und unter User 1 Ihre BASIC-Programme und Daten. Wenn Sie ein Programm von User 1 laden wollen, so gibt es dafür zwei Wege. Der erste ist der, daß Sie mit dem IUSER-Befehl den User definieren, mit

```
IUSER,1
```

ist dies erreicht. Dann können Sie mit LOAD "Filename" das entsprechende Programm laden. Nach dem Laden des Programmes befinden Sie sich dann noch solange im User-Bereich 1, bis Sie ihn umdefinieren. Die zweite Möglichkeit ist, daß Sie den User-Bereich im Filenamen angeben. Mit

```
LOAD "1:Filename"
```

laden Sie ein Programm vom User-Bereich 1, unabhängig von Ihrem momentanen User-Status. Sie müssen also die User-Nummer und dann den Doppelpunkt vor den eigentlichen Filenamen setzen, so wie Sie es schon vom Laufwerk kennen. Wollen Sie das Laufwerk und die User-Nummer wählen, so müssen Sie zuerst die User-Nummer angeben, dann das Laufwerk und schließlich den Doppelpunkt. Beispiel:

```
LOAD "8B:Nim"
```

lädt das Spiel "Nim" von Laufwerk B, User ist 8. Die Syntax lautet also:

```
LOAD "<Usnr><Drive>:Filename"
```

<Usnr> bedeutet beliebige User-Nummer zwischen 0 und 15, <Drive> steht für die Laufwerksbezeichnung, also A oder B.

1.4.3 Die Dateibefehle

Um mit dem Kassettenlaufwerk oder dem Floppy-Laufwerk Dateien zu bearbeiten, benötigt man die Befehle:

```
OPENOUT "Filename"
OPENIN "Filename"
CLOSEOUT
CLOSEIN
INPUT #9
LINE INPUT #9
PRINT #9
WRITE #9
```

Folgendes Programm schreibt die ersten 100 Zahlen auf ein File auf Kassette (CPC 464) oder auf Diskette (CPC 664 & 6128):

```
10 REM ====================================
20 REM    Beispielprogramm für Datenfiles
30 REM ====================================
40 :
50 OPENOUT "zahlen"
60 FOR I=1 TO 100
70   PRINT #9,I
80 NEXT I
90 CLOSEOUT
100 :
110 REM ====================================
120 REM           Jetzt Lesen
130 REM ====================================
140 :
150 MODE 1 : PEN 1
160 PRINT "Spulen Sie das Band zurueck"
170 PRINT "und druecken Sie eine Taste"
180 D$ = INKEY$ : REM *** Loesche Buffer ***
190 IF INKEY$ = "" THEN 190
200 :
210 OPENIN "zahlen"
220 WHILE NOT EOF
230   INPUT #9,I
240   PRINT I,
250 WEND
260 CLOSEIN
270 END
```

Achtung! Dieses Programm schreibt nur dann auf Kassette, wenn Sie einen CPC 464 besitzen und das Diskettenlaufwerk nicht am Rechner angeschlossen haben. Sollten Sie jedoch einen CPC 464 besitzen und Ihre Floppy-Station am Rechner angeschlossen haben, so geben Sie vor Ausführung des Programmes noch das Kommando:

```
ITAPE
```

ein. Jetzt gehen alle Dateibefehle wieder an das Kassettenlaufwerk. Sie können einmal die Zeit stoppen, die das Programm zur Ausführung benötigt. Wenn Sie fertig sind, geben Sie

```
Idisc
```

ein, und starten das Programm erneut. Jetzt wird die Datei "zahlen" auf der Floppy-Station eröffnet. Da Sie bei einer Diskette nicht zurückspulen können, können Sie die Anweisung "Spulen Sie das Band zurück" übergehen und direkt eine Taste betätigen.

Legen Sie bitte eine Diskette ein, die nicht schreibgeschützt ist, am besten eine, die Sie sich zum Üben formatiert haben. Dann geben Sie **RUN** ein.

Sie werden bemerken, daß dieser Arbeitsvorgang deutlich schneller abläuft. Prinzipiell brauchen Sie sich also nicht umzustellen, wenn Sie sequentielle Files mit der Diskette verarbeiten wollen und dies auch schon in BASIC mit dem Kassettenlaufwerk getan haben.

Auch können Sie durch das Kommando ITAPE wieder auf das altbekannte Kassetten-BASIC umschalten, doch näheres unter Kapitel 1.4.4.

Die Funktion EOF, wie sie in Zeile **220** unseres Beispielprogrammes vorkommt, steht für *E*nd *O*f *F*ile, also Ende des Files. Diese Funktion ist unwahr(=0), wenn noch weitere Zeichen aus dem zum Lesen geöffneten File gelesen werden können. EOF wird wahr(=-1), wenn das letzte Zeichen aus diesem File gelesen wurde, es können nun keine weiteren Zeichen mehr aus diesem File geholt werden. Dieser Befehl ist sehr wichtig, wenn man sequentielle Files ausliest und die Anzahl der zu lesenden Zeichen vorher nicht bekannt ist. EOF gibt Ihnen also eine

Möglichkeit in die Hand, sehr flexibel mit sequentiellen Files zu arbeiten.

Wir beschäftigen uns jetzt ausschließlich mit Diskettenbefehlen! Die Auswirkungen auf den Kassettenbetrieb sind adäquat, sollen aber nicht ausführlich in einem Floppy-Buch erläutert werden.

Geben Sie jetzt einmal den Befehl

CAT

ein. Sie erhalten das Disketteninhaltsverzeichnis auf den Bildschirm. Unter Diskettenverzeichnis versteht man die Auflistung der Files, die auf einer Diskettenseite gespeichert sind. Die Diskette muß organisiert werden; dazu gibt es spezielle Blocks auf der Diskette, auf denen die Informationen gespeichert sind: wie die Files heißen, wie lang sie sind, welchem Typ sie angehören und wo das DOS, das Disk Operating System, diese Files finden kann.

Wenn Sie sich mit dem Befehl CAT oder mit IDIR das Inhaltsverzeichnis ansehen, so erhalten Sie Informationen darüber, welche Files sich auf Ihrer Diskette befinden und wieviel Speicherplatz Ihnen noch zur Verfügung steht.

Bei der Benutzung des Kommandos CAT wird Ihnen zusätzlich noch die Länge der Files in KBytes ausgegeben, dies entfällt bei dem Kommando IDIR. Die kleinste Einheit für ein File ist ein KByte.

Die Befehle IDIR und CAT sind nicht identisch. Beide zeigen zwar das Disketteninhaltsverzeichnis an, das IDIR-Kommando zeigt jedoch die Files in der Reihenfolge an, wie sie auf Diskette abgelegt worden sind. Der CAT-Befehl zeigt die Filelänge an und sortiert zusätzlich die Einträge alphabetisch, bevor sie ausgegeben werden. Zwei wichtige Unterschiede also.

Sie werden in diesem Diskettenінhaltsverzeichnis auch das von uns eben erstellte File "zahlen" finden:

```
zahlen  .  1k
```

Da wir keinen Dateityp angegeben haben, erscheint nach dem Punkt auch kein Dateityp. Die Filenamen werden immer automatisch bis zur achten Stelle mit Leerzeichen aufgefüllt.

Sie sehen, daß unsere 100 Zahlen 1 KByte belegen, also 1024 Bytes. Aber selbst wenn wir nur eine Zahl in dieses File geschrieben hätten, so würde das File "zahlen" 1 KByte belegen, da dies die kleinste Einheit unter AMSDOS ist, also seien Sie nicht verwundert.

Doch starten Sie Ihr Programm erneut und sehen Sie, was mit dem Inhaltsverzeichnis passiert. Geben Sie RUN ein, danach CAT.

Sie werden sehen, daß es jetzt zwei Files mit dem Namen "zahlen" gibt. Eines der Files ist ohne Dateibezeichnung. In diesem File stehen die zuletzt geschriebenen hundert Zahlen. Dann gibt es aber noch ein File "zahlen.BAK". Sie erinnern sich sicherlich noch aus dem CP/M-Teil, daß BAK die Abkürzung für *BAcK*up bedeutet. Beim Öffnen des Files "zahlen" durch das Kommando OPENOUT "zahlen" hat das AMSDOS überprüft, ob es das File "zahlen" nicht schon gibt. Da wir unser Programm bereits schon einmal gestartet haben, existiert dieses File tatsächlich schon. Das haben wir auch aus dem Inhalts-verzeichnis gesehen. Da AMSDOS Ihnen dieses File nicht überschreiben will, fertigt es zuvor noch eine Sicherheitskopie an, unter dem Namen "Filename.BAK". Dann erst wird das alte File praktisch gelöscht, damit unter gleichem Namen das neue File eröffnet werden kann. Sie werden diese Eigenschaft von AMSDOS noch zu schätzen lernen. Wenn Sie die Backup-Datei mit Sicherheit nicht mehr brauchen, dann können Sie sie auch löschen.

Auch wenn Sie mit dem Kommando SAVE "Filename" oder SAVE "Filename.BAS",A ein BASIC-Programm abspeichern, wird ein eventuell vorhandenes Programm zuvor als Backup-File gesichert, so daß es relativ wirkungsvoll vor Überschreiben geschützt ist. Probieren Sie folgendes einmal aus, indem Sie NEW eingeben:

```
NEW     (RETURN)
```

```
10 PRINT "Erster Teil."
20 PRINT
```

```
SAVE "Prog"
```

Das Programm wird unter dem Namen "Prog .BAS" auf Diskette gespeichert. Fügen Sie nun noch die Zeilen

```
30 PRINT "wurde ergaenzt."
40 PRINT
```

Geben Sie nun erneut den Befehl zum Abspeichern von Programmen ein:

```
SAVE "Prog"
```

Sie verwenden also exakt denselben Filenamen wie zuvor. Sie können jetzt einmal LOAD "Prog.BAK" eingeben. (Beachten Sie, daß bei der Eingabe von Filenamen nicht tatsächlich acht Zeichen vor dem Punkt stehen müsen.) Wenn Sie sich jetzt Ihr Programm auslisten lassen, so erkennen Sie, daß Sie die erste Version unseres kleinen Programmes geladen haben. Geben Sie nun

```
LOAD "Prog"
```

ein, so wird die letzte Version mit vier Zeilen in den Speicher geladen. Angenommen, Sie ergänzen das Programm noch durch die Zeile

```
50 END
```

und speichern es ein weiteres Mal auf Diskette ab, so ist die allererste Version des Programmes mit nur zwei Kommandozeilen verschwunden. Unser Programm mit vier Zeilen finden Sie nun unter dem Namen "Prog.BAK", das aktuelle Programm hat den Namen "Prog.BAS".

Wenn Sie nun das Programm unter dem Namen "zahlen" abspeichern würden, so würde keine Backup-Datei erstellt, weil es kein File mit Namen "zahlen.BAS" gibt. So bleibt Ihnen also Ihre Datei "zahlen" und Ihre Datei "zahlen.BAK" erhalten. Wenn Sie das Programm allerdings ein weiteres Mal abspeichern, so wird

die Datei "zahlen.BAK" überschrieben von einer Kopie des BASIC-Programmes.

Folgende weitere SAVE-Kommandos sind möglich:

```
SAVE "Filename",A
```

sichert das Programm als ASCII-Datei. Diese Programmfiles sind etwas länger als die "normalen" BASIC-Programm-Files, da die einzelnen Befehle wie etwa "PRINT" Zeichen für Zeichen auf der Diskette gesichert werden, und nicht als Token. (Ein Token besteht aus einem Zeichen und repräsentiert ein BASIC-Schlüsselwort.) Programme, die mit diesem Kommando gesichert wurden, können auch zeichenweise durch ein Programm ausgelesen werden.

```
SAVE "Filename",P
```

sichert ein Programm unter geschütztem Status, d.h. man kann das Programm nicht mehr LISTen und nach einer Unterbrechung mit (ESC)(ESC) weder erneut starten noch sichern.

```
SAVE "Filename",B,<Startadresse>,<Länge>
```

Mit diesem Kommando sichern Sie einen Speicherbereich, beispielsweise den Bildschirmbereich, als Binärdatei. Sie geben die Startadresse und die Länge des Speicherbereiches an. Bei Laden mit dem LOAD "Filename.BIN"-Kommando wird das File dann entsprechend wieder in die Originaladresse geladen. Die Abb. x verdeutlicht diesen Arbeitsvorgang.

Abspeichern eines Speicherbereiches

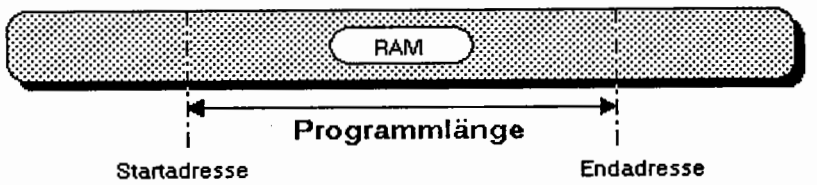


Abb. 4 Abspeichern Speicherausschnitt

Natürlich können Sie auch beim SAVE-Befehl genauso wie beim LOAD-Befehl Angaben über das Laufwerk und die User-Nummer im Filenamen angeben.

Sie haben sicherlich bemerkt, daß Sie mit dem Befehl SAVE genauso arbeiten können, wie Sie es vom Kassetten-BASIC gewohnt sind. Die Benutzung der weiteren BASIC-Befehle zum Laden und Schreiben von Daten werden Ihnen im folgenden Kapitel erklärt, wo auch das Arbeiten mit sequentiellen Files anhand von Beispielen erläutert wird.

1.4.4 Die zusätzlichen AMSDOS-Befehle

Durch das Anschließen des Diskettenlaufwerkes stehen Ihnen noch weitere Diskettenbefehle zur Verfügung, die im *ROM* (*R*ead *O*nly *M*emory) der Floppy gespeichert sind. Diese Befehle nennt man auch *externe Befehle*, da sie in der Floppy definiert sind.

Ohne Floppylaufwerk stehen Ihnen diese Befehle nicht zur Verfügung. Sie können diese Befehle daran erkennen, daß sie mit dem I-Zeichen beginnen (Shift & @-Taste). Sie können diese Befehle sowohl direkt (im sogenannten Direktmodus) eingeben als auch in Programme einbinden. Hier ist die Liste der externen Befehle mit anschaulichen Beispielen.

Wenn ein Parameter mit den Zeichen < und > eingeschlossen ist, so bedeutet das, daß es sich um ein Synonym handelt. Beispielsweise finden Sie häufig den Ausdruck <Stringausdruck>, was bedeutet, daß Sie an dieser Stelle in das Kommando eine Stringvariable einfügen müssen. Ist der <Stringausdruck> aber in eckige Klammern gesetzt [<Stringausdruck>], so zeigt Ihnen das an, daß der Stringausdruck auch weggelassen werden kann, er ist also Option.

IA Dieser Befehl setzt das Standardlaufwerk auf Laufwerk A. Sie können diesen Befehl bei nur einem Laufwerk nicht gebrauchen, da dieses Laufwerk A automatisch das Standardlaufwerk ist. Unter Standardlaufwerk versteht man das Laufwerk, das bei Floppybefehlen ohne spezifizierte Laufwerksangabe automatisch angesprochen wird.

Der Befehl ist identisch mit dem Kommando

```
a$="a"
IDRIVE,@a$
```

IB Setzt entsprechend das Laufwerk B als Standardlaufwerk (s.o.). Dieser Befehl ist identisch mit

```
a$="b"
IDRIVE,@a$
```

ICPM Es wird das Betriebssystem CP/M (siehe auch entsprechendes Kapitel) von Diskette geladen. Wenn Sie dieses Kommando verwenden, muß sich eine Diskette, die mit CP/M auf den beiden System-Spuren formatiert worden ist, im Laufwerk A befinden. Das ist bei der mitgelieferten Diskette der Fall.

IDIR Dieses Kommando IDIR,[<*Stringausdruck*>] zeigt Ihnen den Disketteninhalt und ist weitestgehend identisch mit dem Standard-BASIC-Befehl CAT.

Es wird Ihnen das Disketteninhaltsverzeichnis angezeigt sowie der auf der Diskette noch verfügbare Speicherplatz.

Sollte der optionale <Stringausdruck> fehlen, so wird *.* angenommen, das heißt, alle Files werden ausgegeben (ohne Selektion). Wie schon erläutert, sind Sterne und Fragezeichen (*/?) Wild Cards und repräsentieren beliebige Zeichen. Zur Erinnerung: Ein Fragezeichen bedeutet, hier kann ein beliebiges Zeichen stehen, ein Stern bedeutet, ab hier bis zum Ende des Filenamens können beliebige Zeichen stehen. Angenommen, Sie wollten lediglich die BASIC-Files ausgelistet bekommen, so können Sie dies mit dem Kommando

```
a$="*.BAS"
IDIR,@a$
```

erreichen. Sie könnten aber auch beispielsweise alle Files ausgeben lassen, die mit dem Kürzel "Math" beginnen. Dazu müßten Sie eingeben:

```
a$="Math*.*"
IDIR,@a$
```

Die Verwendung von Wild Cards und Jokern kann hier sehr nützlich und zeitsparend sein. Gerade beim IDIR-Befehl dient diese Selektion der Übersichtlichkeit.

Der Vorteil des CAT-Befehls ist, daß er die Filenamen vor der Ausgabe sortiert und die Filelängen angibt. Der IDIR-Befehl hat den Vorteil, daß man die Ausgabe selektieren kann.

IDISC Dieser Befehl enthält die beiden Befehle IDISC.IN und IDISC.OUT. Da Sie für das Kassettenlaufwerk und die Floppystation dieselben Basic-Befehle verwenden müssen, könnten Sie nur eines der beiden Geräte erfolgreich ansprechen. Damit Sie aber auch das Kassettenlaufwerk benutzen können (sofern vorhanden), wenn Sie mit der Floppy arbeiten, wurde Ihnen mit den Befehlen IDISC und ITAPE die Möglichkeit an die Hand gegeben, frei das Eingabe- bzw. das Ausgabegerät zu wählen.

IDISC.IN Nachdem dieser Befehl gegeben wurde, werden die folgenden Eingabebefehle automatisch auf die Floppy bezogen und nicht auf das Kassettenlaufwerk:

```
LOAD "Dateiname"
RUN "Dateiname"
CHAIN "Dateiname"
CHAIN MERGE "Dateiname"
MERGE "Dateiname"
OPENIN "Dateiname"
CLOSEIN
EOF
CAT
INPUT §9
LINE INPUT §9
```

Dieses Kommando schaltet den ITAPE- oder den ITAPE.IN-Befehl wieder aus. Sollten Sie beispielsweise folgende Befehlssequenz haben

```
ITAPE
IDISC.IN
OPENIN "Datei"
OPENOUT "Backup"
```

so würde ein Eingabekanal auf der Floppy zum Lesen eröffnet, der Ausgabekanal aber schreibt auf Kassette. Auf diese Art und Weise können Sie beispielsweise Dateien von Diskette auf Kassette kopieren (zur Datensicherheit).

`IDISC.OUT` Dieser Befehl ist dem Befehl IDISC.IN ähnlich, jedoch werden hier die Ausgaben auf die Floppy gelegt. Folgende Befehle sind davon betroffen:

```
SAVE "Dateiname"
OPENOUT "Dateiname"
CLOSEOUT
WRITE #9
PRINT #9
```

```
ITAPE

IDISC.OUT
IOPENIN "Backup"
IOPENOUT "Datei"
```

Dies wäre die Umkehrung des Beispiels unter IDISC.IN. Nun würde also von der Kassette die Backup-Datei mit den Lesebefehlen gelesen,

die Schreibbefehle würden sich aber auf die Floppy beziehen.

IDRIVE Dieser Befehl IDRIVE,<Stringausdruck> ist identisch mit den Kommandos IA und IB, jedoch ist dieser Befehl flexibel, da man in einer Stringvariablen den Namen des zu wählenden Standardlaufwerkes übergeben kann.

```
a$="b"
IDRIVE,@a$
```

würde zur Konsequenz haben, daß das Laufwerk B vom Zeitpunkt der Ausführung des Befehls an zum Standardlaufwerk definiert wird. Sie können dieses Kommando nicht verwenden, wenn Sie nur ein Laufwerk haben.

IERA ERA steht für das englische Wort Erase(=Löschen). Mit diesem Kommando können Sie Files löschen. Sie können auch bei diesem Kommando Wild Cards benutzen, beispielsweise würde der Befehl

```
a$="*.seq"
IERA,@a$
```

alle Dateien mit dem Dateityp "seq" löschen. Die Benutzung dieses Befehls sollte immer gut durchdacht sein, insbesondere dann, wenn man mit wild cards arbeitet, da unter Umständen auch nicht vorgesehene Files gelöscht werden könnten. Bei dem Befehl IERA wird von der Benutzung von Jokern abgeraten, wenn man sich nicht schon ein wenig mit der Floppy und der Benutzung der Joker auskennt.

Das Kommando

```
a$="*.*"
IERA,@a$
```

würde alle Files löschen. In diesem Fall erwartet AMSDOS eine spezielle Bestätigung des Kommandos.

IREN REName(=Umbenennen) eines Files. Mit diesem Kommando können Sie also Files einen anderen Namen geben. Der erste Stringausdruck muß den neuen Filenamen enthalten, der zweite Stringausdruck enthält den alten Filenamen. Wollen Sie beispielsweise das File mit dem Namen "Opa" umbennen, weil Sie Ihre Oma mehr mögen, so können Sie folgende Befehle eingeben:

```
alt$="Opa.bas"
neu$="Oma.bas"
IREN,@neu$,@alt$
```

Wenn Sie sich nun mit dem Kommando IDIR das Inhaltsverzeichnis ansehen, so werden Sie sehen, daß das File "Opa.BAS" nicht mehr existiert, dafür das File "Oma.Bas". Die Angabe beider Stringausdrücke ist Pflicht.

ITAPE Dieses Kommando hebt die Kommandos IDISC, IDISC.IN und IDISC.OUT auf und legt sowohl die Eingabe als auch die Ausgabe auf das Kassettenlaufwerk. Das Kommando faßt die Befehle ITAPE.IN und ITAPE.OUT zusammen.

ITAPE.IN Diesen Befehl verwendet das Kassettenlaufwerk als Eingabedatei. Der Befehl hebt die Kommandos IDISC und IDISC.IN auf.

ITAPE.OUT Es wird das Kassettenlaufwerk als Ausgabegerät verwendet. Die Befehle IDISC und IDISC.OUT werden aufgehoben.

IUSER Mit diesem Kommando können Sie einen bestimmten User = Benutzer definieren. Sie haben sicherlich auch schon gesehen, daß in den Inhaltsverzeichnissen der Diskette Ihnen auch die Meldung

```
USER:0
```

ausgegeben wird. Dies ist ein spezielles CP/M-Kommando und sehr nützlich. Nähere Informationen hierzu sind unter Kapitel 1.4.2 enthalten.

Mit dem Kommando IUSER können Sie beispielsweise Files vor der Sicht anderer sichern. Speichern Sie einmal ein beliebiges BASIC-Programm wie folgt ab:

```
IUSER,3
SAVE "Programm"
```

```
IUSER,0
CAT
```

oder viel einfacher mit dem Kommando

```
SAVE "3:Programm"
```

So werden Sie Ihr frisch abgespeichertes Programm nicht in der Liste vorfinden, da sich das soeben abgespeicherte Programm nicht im aktuellen Benutzerbereich befindet. Wenn Sie aber

```
IUSER,3
CAT
IUSER,0
```

eingeben, so wird Ihnen ein weitaus kürzeres Inhaltsverzeichnis mit nur einem Eintrag angezeigt. Sie können also mehrere verschiedene Inhaltsverzeichnisse einrichten oder bestimmte Programm vor anderen Benutzern verbergen.

Sie haben sicherlich gemerkt, daß die Befehle mit einem Stringausdruck besonders umständlich in der Anwendung sind. So müssen Sie beispielsweise

```
a$="a"
IDRIVE,@a$
```

eingeben, anstatt des einfacheren

```
IDRIVE,"a"
```

Diese erwähnte *einfachere* Methode ist bei den beiden Rechnern CPC 664 und CPC 6128 möglich, da hier Korrekturen im BASIC-Betriebssystem vorgenommen worden sind. Wollen Sie Programme schreiben, die möglichst auf allen drei Rechnern laufen, so ist es erforderlich, daß Sie sich der hier erwähnten Methode bedienen, da beispielsweise ein IDRIVE,"A" beim CPC 464 zu einer Fehlermeldung führen würde - es lohnt sich also, dieses etwas umständlichere Verfahren anzuwenden - der Kompatibilität zu Liebe.

Jedoch sieht es das BASIC-Betriebssystem des CPC 464 so vor, daß eine Adresse eines Stringausdruckes an das Betriebssystem AMSDOS übergeben wird. AMSDOS holt sich diesen String dann selbst aus dem Speicher. Strings werden irgendwo im Speicher abgelegt. Mit der Funktion *@Variablenname* können Sie sich die Adresse anzeigen lassen, wo die Variable, in unserem Fall der String, abgespeichert ist. Sicherlich ist die Form der Datenübermittlung nicht sehr komfortabel, jedoch werden Sie sich schnell an diese Gebrauchsform gewöhnen.

Das @-Kommando ist ein nützliches BASIC-Kommando, das nicht im Bedienungshandbuch erwähnt wird. Geben Sie einmal ein:

```
a=12.2
PRINT @a
```

Je nachdem, wie viele Variablen Sie bereits benutzt haben, wird der ausgegebene Wert höher oder niedriger sein. Je später eine Variable deklariert wird, desto größer ist der Wert der Funktion @Variablenname. Weiterhin ist es bedeutend, wie lang das aktuelle BASIC-Programm ist, da die Variablentabelle unmittelbar nach dem BASIC-Programm im Speicher beginnt.

Wie Sie vielleicht wissen, werden numerische Variablen von unten nach oben im Speicher abgelegt, die Strings liegen an der oberen Grenze des Speichers und bewegen sich nach unten. Sie können dies anhand des folgenden Beispiels ersehen:

```
NEW
a=10.1 : b=20 : a$="Beispiel 1" :
b$="Beispiel 2"


PRINT @a,@jb,@ja$,@b$
```

Wenn Sie sich nun die Adressen der Variablen ansehen, so belegen die numerischen Variablen 9 Bytes. Hier sind die Werte der Variablen a und b gespeichert, sowie deren Namen. Die Stringvariablen werden aber auf eine andere Weise abgespeichert. Geben Sie einmal

```
PRINT PEEK(@a$)
```

ein. Sie erhalten dann den Wert 10, dies ist exakt die Länge der Variablen a$. Dasselbe gilt für die Stringvariablen b$. Die Werte der Speicherstellen

```
@a$+1 sowie
@a$+2
```

beinhalten einen Zeiger. Dieser Zeiger teilt dem Betriebssystem mit, wo es die eigentliche Zeichenkette finden kann. Diese Methode der Stringspeicherung erscheint Ihnen vielleicht als kompliziert, sie hat jedoch einen entscheidenden Vorteil: Man braucht nicht die gesamte Variablentabelle zu verschieben, wenn ein String seine Länge ändert; man muß lediglich die drei beschriebenen Bytes verändern.

Doch lassen wir uns erst einmal den String a$ auf eine unkonventionelle Art und Weise ausgeben:

```
ad=PEEK(@a$+1) + 256*PEEK(@a$+2)
```

Wir haben nun die Adresse errechnet, an der die Zeichenkette a$ gespeichert ist - die Adresse haben wir in der Variablen ad zwischengespeichert.

```
FOR I=0 TO PEEK(@a$)-1
  PRINT CHR$(PEEK(ad+I));
NEXT I
```

Und Sie erhalten exakt den Inhalt der Variablen a$ wieder, Sie können sehr einfach die Zeiger der Stringvariablen verbiegen und somit erreichen, daß Sie den Inhalt der Variablen beliebig verändern. Diese Eigenschaft wird sich auch in den Programmen in Kapitel 5 (Fehlerabfangroutine und Relative Dateiverwaltung) zu Nutze gemacht. Verbiegen Sie beispielsweise den String a$ einmal auf Ihren Bildschirm:

```
POKE @a$+1,&00POKE @a$+2,&C1
```

Der Zeiger für die Zeichenkette zeigt jetzt mitten in den Bildschirmspeicher. Lassen Sie sich jetzt einmal die Stringvariable a$ ausgeben:

```
PRINT a$
```

Der Effekt läßt sich jetzt nicht genau beschreiben, da es bei jedem anders aussehen wird - jeder hat etwas anderes auf seinem Bildschirm stehen - aber vielleicht blinkt der Rahmen, oder der Schriftmodus wechselt. Auf jeden Fall erhalten Sie eine Flut wirrer Zeichen. Dies sollte Ihnen als Einführung in den @-Befehl dienen.

1.5 Sequentielle Datenspeicherung

1.5.1 Was ist sequentielle Datenspeicherung?

Ein Diskettenlaufwerk soll natürlich nicht nur dem Abspeichern und Laden von Programmen dienen - es eignet sich darüber hinaus ausgezeichnet zur Abspeicherung großer Datenmengen, die bislang mit dem Kassettenlaufwerk kaum oder gar nicht zu bewältigen waren.

Hier gilt es zunächst einmal den Begriff *Daten* zu klären. Wir müssen unterscheiden zwischen Dateien, die *Programme* beinhalten und Dateien, die lediglich *Daten* beinhalten. Beide Arten von Dateien werden zwar sequentiell (s.u.) abgespeichert, haben aber eine wesentliches Unterscheidungsmerkmal:

Programmdateien (seien es nun BASIC-Programme oder Maschinenprogramme) können Sie mittels des LOAD-Kommandos in den Speicher des Rechners *direkt* einlesen. Anders bei den Daten-Dateien; diese können Sie nur über Programme in den Speicher einlesen und verwalten. Eine Finanzbuchhaltung, die in BASIC geschrieben ist, ist beispielsweise auf der Diskette eine Programmdatei. Die in diesem Programm anfallenden Daten würden dann als sequentielle Dateien abgespeichert.

Mit der DDI-1 können Sie standardmäßig nur sequentielle Files verarbeiten, so wie Sie es schon vom Kassetten-BASIC her kennen; das Prinzip ist vollkommen identisch. Sequentielle Datenspeicherung ist nicht die schnellste, aber die einfachste Methode, Daten zu speichern und für kleine bis mittelgroße Probleme (damit ist der Programmumfang und der anfallende Datenumfang gemeint) auch vollkommen ausreichend. Da sie außerdem einfach zu verstehen ist, bietet sie sich zum Erlernen direkt an.

Sequentiell bedeutet nichts anderes als Zeichen für Zeichen. Ein sequentielles File ist eine Sequenz von Zeichen (Buchstaben, Ziffern, Sonderzeichen u.ä.), also eine Aneinanderreihung von Zeichen. Um das letzte Zeichen eines Files zu lesen, muß man alle vorangestellten Zeichen überlesen, vom ersten bis zum vor-

letzten. Stellen Sie sich ein Buch mit 100 Seiten vor. Sie wollen auf der 100. Seite den ersten Buchstaben lesen und müßten dazu vorher alle Zeichen der Seiten 1 - 99 überlesen. Auf diese Weise wird ein sequentielles File gelesen - eine sicherlich nicht sehr komfortable Methode. Zum Glück ist die DDI-1 sehr schnell, so daß der Zeitverlist kaum auffällt.

Abb. 5 Sequentielle Datei

Eine weitaus komfortablere Möglichkeit des Lesens von Daten ist die Möglichkeit, *frei* auf ein File zugreifen zu können man nennt dies *freier Zugriff* oder *Random Access*, auch der Begriff *relative Dateien* hat sich eingeprägt.

Freier Zugriff bedeutet, daß Sie - um bei unserem Vergleich mit dem Buch zu bleiben - direkt den beispielsweise 55. Buchstaben auf der 29. Seite lesen können, ohne alle vorangehenden Seiten überlesen zu müssen. Ein weiterer Vorzug: Wenn Sie in einem sequentiellen File am 100. Datensatz angekommen sind, können Sie nicht mehr ein Zeichen aus dem 5. Datensatz lesen - man kann also nicht rückwärts gehen. Dies ist eine Einschränkung der Leistungsfähigkeit sequentieller Dateien, die man bei den relativen Dateien nicht vorfindet.

Dies aber nur, um Ihnen grob die Unterschiede anzuzeigen. In Kapitel 5 finden Sie mehr über relative Dateien, und wie man Sie auf der DDI-1 programmiert.

Wir wollen nun anhand unterschiedlicher Beispiele erläutern, wie man die sequentielle Datenspeicherung effektiv nutzen kann, da sie (zunächst) die einzige uns gegebene Möglichkeit ist, Daten auf Diskette zu sichern.

Jetzt soll unsere erste kleine Datei erstellt werden. Als Beispiel wollen wir ein Telefonregister einrichten, um die wichtigsten Telefonnummern von Verwandten oder Bekannten zu speichern.

Wenn man eine Datei anlegt, so steht am Anfang immer die Überlegung, was will ich überhaupt speichern? Im folgenden Beispiel haben wir uns für diese Angaben entschieden:

1) Name (=Feld 1)
2) Vorname (=Feld 2)
3) Telefonnummer (=Feld 3)

Ein Datensatz ist eine abgeschlossene Einheit; jeder Datensatz beinhaltet bestimmte Informationen, die man in jedem anderen Datensatz auch vorfindet. Unser Datensatz besteht aus 3 *Feldern*, auch *Datenfelder* genannt. Um das Verständnis für einen Datensatz zu verdeutlichen, diese Grafik:

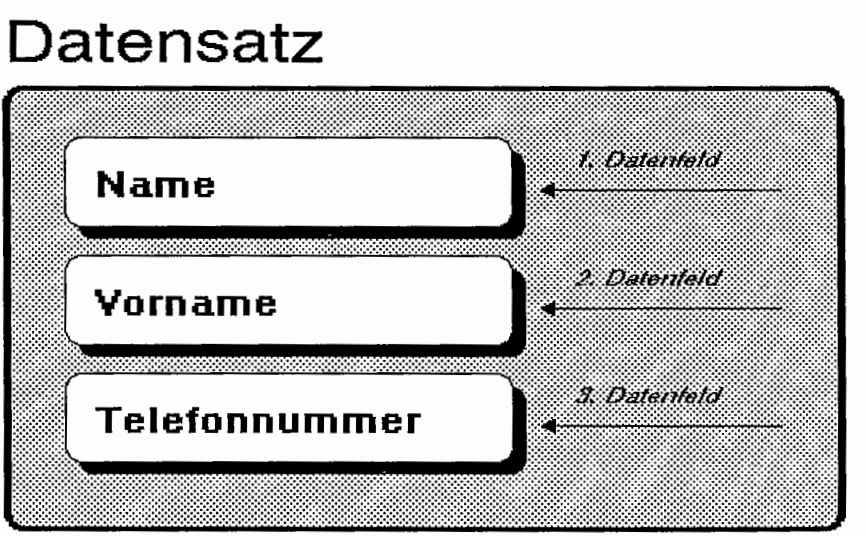


Abb. 6 Datensatz

So könnten wir einen Datensatz "Müller" haben, der als Information Namen, Vornamen und Telefonnummer des Kollegen Müller beinhaltet.

Feld 1: *Name*
Feld 2: *Vorname*
Feld 3: *Telefonnummer*

Sie können die Unterteilungen eines Datensatzes in verschiedene Datenfelder in dieser Grafik gut erkennen.

Wenn die Daten in ein File geschrieben werden, so kann man nicht so deutlich wie hier auf dem Papier die Unterteilungen von Datensätzen erkennen, es werden lediglich die Datenfelder physisch voneinander getrennt. Die Trennung der Datensätze unterscheidet sich nicht von der Trennung von Datenfeldern, es wird dasselbe Trennsymbol verwandt. Die Trennung der Datensätze - also die korrekte Zuordnung der Datenfelder - obliegt dem Programmierer. Er weiß eigentlich als einziger, wie viele Felder jeder Datensatz hat. Deswegen sollte man solche Informationen deutlich im Programmlisting vermerken.

Haben Sie eine Eingabe auf dem Bildschirm beendet, so betätigen Sie die ENTER-Taste, auch *Wagenrücklauf* oder *Carriage-Return* genannt. Genauso ist es bei der sequentiellen Datenspeicherung, hier trennt man einzelne Datenfelder durch einen Wagenrücklauf auf der Diskette, mit der "Taste" ENTER beenden Sie also die Eingabe in einem Datenfeld.

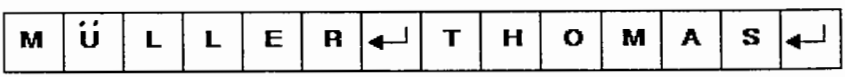


Abb. 7 Trennung von Datenfeldern

Um Ihnen die Vorstellung zu erleichtern, wie nun die einzelnen Datenfelder getrennt werden, sehen wir uns einmal die Abbildung 7 an. Das Trennsymbol Wagenrücklauf wird durch den abgeknickten Pfeil repräsentiert.

An unserem Beispiel können Sie erkennen, daß Datenfelder nicht die gleiche Länge haben müssen, sondern in sequentiellen Files eine beliebige Länge annehmen können, dasselbe gilt für die Länge von Datensätzen. Trotzdem bleiben die Datenfelder eindeutig bestimmmbar, da sie durch das Symbol für Wagenrücklauf voneinander getrennt sind.

Um eine solche Datei einzulesen, verwendet man den INPUT #9-Befehl.

```
OPENIN "Dateiname"
INPUT #9,Name$,Vorname$,Telefon$
CLOSEIN
```

Nach diesen Kommandos würde nur ein bestimmter Datensatz eingelesen; die Variablen hätten beispielsweise folgende Werte:

```
Name$     "Seeler"
Vorname$  "Uwe"
Telefon$  "142251"
```

Dabei zeigen die Anführungszeichen lediglich den Beginn und das Ende der Zeichenkette an und gehören nicht zum eigentlichen String. Trotzdem können Strings Anführungszeichen enthalten.

1.5.2 Die Programmierung sequentieller Files

Doch wollen wir nun konkret ein sequenielles File auf Diskette erstellen. Dazu geben Sie folgendes kleines Programm ein:

```
 10 REM ****************************
 20 REM ** Erstes Datei-Programm **
 30 REM ****************************
 40 :
 50 OPENOUT "Test.dat"
 60 PRINT #9,"Seeler"
 70 PRINT #9,"Uwe"
 80 PRINT #9,"142251"
 90 REM --- Erster Datensatz ---
100 PRINT #9,"Berghoff"
110 PRINT #9,"Dagmar"
120 PRINT #9,"239901"
130 REM --- Zweiter Datensatz ---
140 PRINT #9,"Toll"
150 PRINT #9,"Karl"
160 PRINT #9,"181551"
170 REM --- Dritter Datensatz ---
180 CLOSEOUT
190 END
```

Wir haben damit ein File unter dem Namen "Test.dat" erstellt, in dem drei Datensätze enthalten sind.

Das BASIC-Kommando PRINT #n wird genauso gehandhabt, wie das ganz normale PRINT. Sie kennen das Kommando PRINT # sicher schon im Zusammenhang mit den Windows (Fenstern), die Sie auf Ihrem Schneider definieren können. Mit PRINT # können Sie eines der definierten Fenster ansprechen, wenn n im Bereich 0 bis 7 liegt. Bei n=8 wird der Drucker angesprochen und bei n=9 das Kassettenlaufwerk oder die Floppy, je nach Einstellung. Dasselbe gilt für die Kommandos INPUT #n und LINE INPUT #n.

Das logische Trennzeichen Carriage Return wird in unserem Beispiel zwischen die Datensätze gesetzt, da dem PRINT #9-

einen Datensatz in einer INPUT-Anweisung einlesen, beispielsweise mit

```
INPUT #9,Name$,Vorname$,Telefon$
```

oder ob Sie den Datensatz in mehreren Input-Anweisungen stückchenweise einlesen, beispielsweise mit

```
INPUT #9,Name$
INPUT #9,Vorname$
INPUT #9,Telefon$
```

Wenn Sie einen Datensatz eingelesen haben, so merkt sich ein sogenannter Pointer die Stelle, an der weitergelesen werden muß.

Sie können ein File, das Sie zum Lesen geöffnet haben, auch vorzeitig schließen, d.h. Sie brauchen nicht alle Elemente aus diesem File auszulesen.

Sollten Sie aber das letzte Element des Files gelesen haben, und Sie tätigen einen weiteren Leseversuch auf genau dieses File, so erhalten Sie die Fehlermeldung:

```
EOF met
```

Man kann sagen, daß hierdran ein sogenannter *interner Pointer (Zeiger)* Schuld ist. Dieser interne Pointer wird direkt nach dem Öffnen einer Datei angelegt. Am Anfang (also nach dem Öffnen) zeigt dieser interne Pointer auf die erste Stelle des Eingabebuffers, der im CPC-RAM liegt, und somit auf das erste Zeichen in der Datei. Wird dieses Zeichen ausgelesen, so bewegt sich der Zeiger um eine Stelle weiter nach rechts und zeigt auf das nächste zu lesende Zeichen, das beim nächsten Lesebefehl dann auch ausgelesen wird.

Ist das letzte Zeichen gelesen, so zeigt dieser interne Pointer auf die END-OF-FILE-Marke und Sie erhalten die entsprechende Fehlermeldung auf dem Bildschirm.

Dieses Verfahren des internen Pointers soll die folgende Grafik noch verdeutlichen:

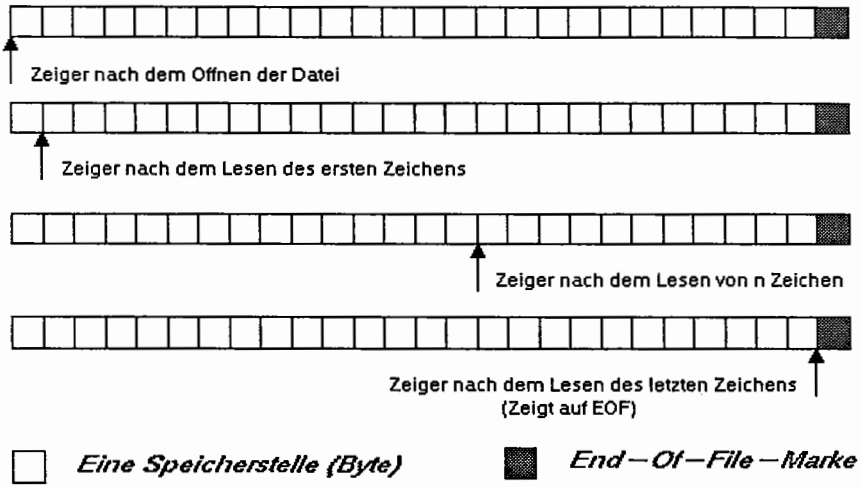


Abb. 8 Interner Pointer

Liest man nun nicht zeichenweise sondern Datenfeld- oder gar Datensatzweise, so kann man dies auch mit dem Internen-Pointer-Modell verdeutlichen. Dieser Zeiger zeigt dann halt am Anfang auf den ersten Datensatz und bewegt sich nach dessen Auslesen auf den folgenden Datensatz.

Die BASIC-Funktion EOF sagt uns also, ob wir das letzte Element eines Files bereits gelesen haben oder noch nicht.

Um zu vermeiden, über das Dateiende hinaus zu lesen, können Sie die Standard-BASIC-Funktion EOF verwenden. Wenn Sie nicht genau wissen, wieviel Datensätze Sie einlesen müssen, so

ist es sogar umungänglich, die Funktion EOF zu verwenden. Unser Beispielprogramm sähe dann wie folgt aus:

```
 10 REM ****************************
 20 REM ** Daten einlesen mit eof **
 30 REM ****************************
 40 :
 50 OPENIN "Test.dat"
 60 WHILE NOT EOF
 70   INPUT #9,Name$,Vorname$,Telefon$
 80   PRINT Name$,Vorname$,Telefon$
 90 WEND
100 CLOSEIN
110 END
```

Dabei ist dieses Beispiel schon gefährlich, denn wir gehen davon aus, daß in unserer Datei auf jeden Fall immer drei weitere Datenfelder zur Verfügung stehen, wenn wir nicht auf EOF gestoßen sind. Bei Files unbekannter Struktur ist dies allerdings nicht möglich. Dann müßte unser Programm wie folgt aussehen:

```
 10 REM ********************************
 20 REM ** Datei einlesen - flexibel **
 30 REM ********************************
 40 :
 50 OPENIN "Test.dat"
 60 WHILE NOT EOF
 70   INPUT #9,Feld$
 80   PRINT Feld$
 90 WEND
100 CLOSEIN
110 END
```

Bei diesem Programm ist ein EOF-MET-Error ausgeschlossen, da jedes Feld einzeln eingelesen und dann wiederum auf EOF getestet wird.

Wenn Datenfelder durch das INPUT #9-Kommando eingelesen werden, so gelten folgende Zeichen als Trennsymbol der Datenfelder:

Carriage Return (Wagenrücklauf)
Komma (,)
EOF-Marke

Wenn Sie numerische Variablen durch INPUT #9 einlesen, so gilt weiterhin die *<Leertaste>* als Trennzeichen.

Auch diese Trennzeichen kann man sich leicht merken, da das Komma und der Wagenrücklauf ebenfalls bei der normalen INPUT-Routine als Trennsymbol gelten.

Natürlich könnte man sich auch ein "privates" Endezeichen definieren, etwa könnte man in das letzte Datenfeld immer ein "*ende" schreiben oder ähnliches und im Programm dann dieses Zeichen abfragen. Sie werden mir aber sicherlich zustimmen, daß die Handhabung der EOF-Funktion sicherer und einfacher ist.

Meistens will man aber nicht nur die Daten lesen und direkt ausgeben, sondern die Daten sollen zum Auswerten und Ändern im Speicher des Computers bleiben. In diesem Falle legt man am besten ein ARRAY an, eine indizierte Variable. Sollte Ihnen dies kein Begriff sein, so lesen Sie bitte das entsprechende Kapitel im BASIC-Handbuch durch.

Unser Beispielprogramm sähe dann wie folgt aus:

```
10 DIM Name$(3),Vorname$(3),Telefon$(3)
20 REM ******************************
```

```
 30 REM ** Datei einlesen-speichern **
 40 REM ******************************
 50 :
 60 x=0
 70 OPENIN "Test.dat"
 80 WHILE NOT EOF
 90   x=x+1
100   INPUT #9,Name$(x),Vorname$(x),Telefon$(x)
100 WEND
110 PRINT "Es sind";x-1;"Datensätze eingelesen worden."
120 CLOSEIN
130 END
```

In der DIM-Anweisung in Zeile 10 müssen Sie entsprechend der oberen Grenze der Datensatzanzahl die Indexgrenze definieren.

Mit einer solchen Schleife kann man Dateien in eine Feldvariable einlesen, dann mit dem Programm bearbeiten und schließlich wieder sequentiell abspeichern. Es gilt hier also die Regel der Datenverarbeitungskaufleute, die als EVA bekannt ist, Einlesen, Verarbeiten und Ausgeben, in dieser Reihenfolge. Wenn der Speicher es zuläßt, sollten Sie Ihre Dateien auch auf diese Weise einlesen und im Programm verwalten; Sie haben hier enorme Geschwindigkeitsvorteile.

1.5.3 Sequentielle Dateien und Feldvariablen

Bei Programmen größeren Umfangs, die auch große Datenmengen verwalten müssen, ist es angebracht, die vorgesehene Datenmenge vorher genau zu ermitteln und dann entsprechend im Programm zu definieren. Man nennt diesen Vorgang auch *Programmkonzept*; man ermittels das sogenannte *Mengengerüst*.

Eine sequentielle Datei ist also eine Aneinanderreihung von Datensätzen, die wiederum aus Datenfeldern bestehen.

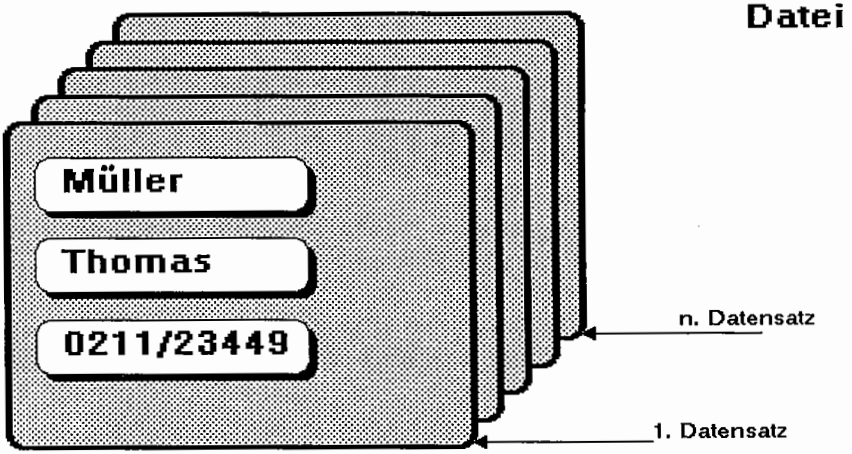


Abb. 9 Datei

Man kann den Begriff "Datei" aber noch sehr viel anschaulicher verdeutlichen, wenn man sich der sogenannten *Syntax Diagramme* bedient, die man beispielsweise zur Syntax-Definition (praktisch die Grammatik) höherer Programmiersprachen wie PASCAL oder C benutzt. Erstmals wurden solche Syntax-Diagramme wohl bei der Definition der strukturierten Programmiersprache PASCAL von Nikolaus Wirth eingesetzt.

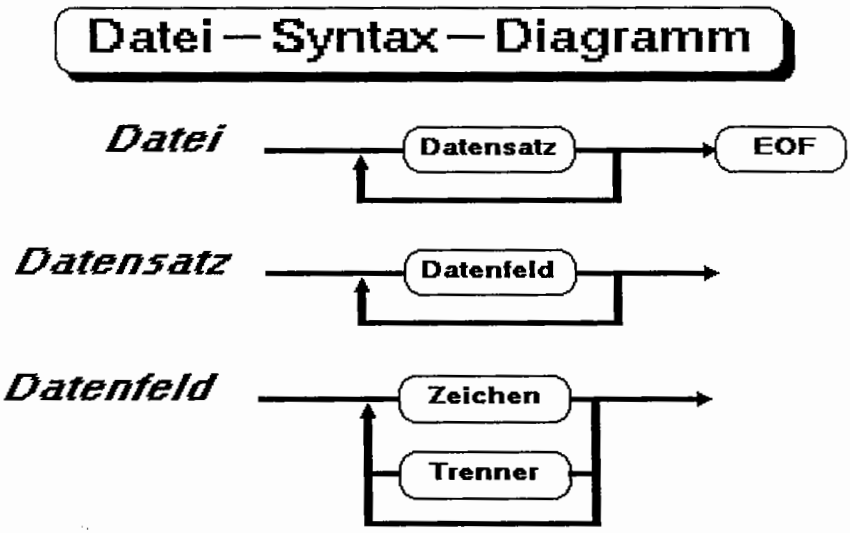


Abb. 10 Syntaxdiagramm Datei

Wenn Sie das von uns erstellte sequentielle File einmal auslesen und in der Feldvariablen alle Datensätze ablegen, sparen Sie eine Menge Zeit, da Sie von da an nicht mehr auf das Diskettenlaufwerk zurückgreifen brauchen. (Ein Zugriff auf Variablen im Speicher ist immer noch schneller als ein Zugriff auf die Diskette).

Unsere Feldvariablen sehen so aus :

Index	*NAME$*	*VORNAME$*	*TELEFON$*
1	Seeler	Uwe	142251
2	Berghoff	Dagmar	239901
3	Toll	Karl	181551

Abb. 11 Datensatz

Wir haben diese Daten jetzt also einmal in Feldvariablen eingelesen, allerdings in drei verschiedene eindimensionale Feldvariablen. Es ist praktischer für den Programmierer, ein zwei-dimensionales Feld anzulegen, um dem Kinde einen Namen zu geben; es dient also der Übersichtlichkeit des Programmes.

2-Dimensionale-Feldvariable

Index	*NAME*	*VORNAME*	*TELEFON*
1	D$(1,1)	D$(1,2)	D$(1,3)
2	D$(2,1)	D$(2,2)	D$(2,3)
3	D$(3,1)	D$(2,3)	D$(3,3)
4	D$(4,1)	D$(2,4)	D$(4,3)
5	D$(5,1)	D$(2,5)	D$(5,3)

Abb. 12 2-dimensionales Feld

Unsere Feldvariable D$(x,y) hat somit fünf Datensätze mit je drei Datenfeldern. Diese Feldvariable müßte mit der Anweisung

```
DIM D$(5,3)
```

im Programm "dimensioniert", d.h. eingerichtet werden. Die Feldvariable D$(x,y) hätte dann beispielsweise folgende Eintragungen:

D$(x,1)=Name *vorher Name$(x)*
D$(x,2)=Vorname *vorher Vorname$(x)*
D$(x,3)=Telefon *vorher Telefon$(x)*

Unser Beispielprogramm sieht dann so aus:

```
 10 REM *************************
 20 REM ** Einlesen in D$(x,y) **
 30 REM *************************
 40 DIM D$(3,3)
 50 REM 3 Datensaetze & 3 Datenfelder
 60 OPENIN "Test.dat"
 80 FOR I=1 TO 3
 90   FOR J=1 TO 3
100     INPUT #9,D$(I,J)
110   NEXT J
120 NEXT I
130 :
140 REM ==========================
150 REM          AUSGABE
160 REM ==========================
170 FOR I=1 TO 3
180   FOR J=1 to 3
190     PRINT D$(I,J),
200   NEXT J
210   PRINT
220 NEXT I
230 CLOSEIN
240 END
```

Dieses Beispielprogramm können Sie als Musterprogramm nehmen, wenn Sie selbst einmal ein Dateiprogramm schreiben. Durch Ineinanderschachteln von zwei FOR-NEXT-Schleifen erreichen Sie schrittweises logisches Einlesen der Datenfelder und -sätze. Wenn Sie in einem Dateiprogramm Daten Ändern wollen, so lesen Sie am besten alle Daten in eine solche Feldvariable ein. Dann können Sie die Daten bearbeiten, korrigieren etc. Am Ende des Programmes müssen die Daten dann wieder unter denselben Namen abgespeichert werden.

Meistens erstellt man noch ein Hilfsfile, in dem sich Informationen über das Mengengerüst befinden. Unter Mengengerüst kön-

nen Sie verstehen: Wie viele Daten sind im Feld eingetragen? Wie viele Datenfelder hat jeder Datensatz? Wenn Sie ein flexibles Dateiprogramm erstellen, so ist dies aus praktischen Gründen unumgänglich. Außerdem können Sie auf diese Weise wichtigen Speicherplatz sparen, da jede unnötige Koordinate Speicherplatz und auch Zeit kostet. In der DIM-Anweisung kann man auch dynamisch dimensionieren, d.h. Sie können Variablen als Indexgrenzen eintragen. Wir erstellen einmal ein solches Informationsfile. Dazu tragen wir die Anzahl der DATENSÄTZE, sowie die Anzahl der DATENFELDER pro Datensatz ein.

```
OPENOUT "Test.inf"
WRITE #9,3,3
CLOSEOUT
```

Wir nennen den Dateityp "INF" für INFofile. Der Dateiname muß in unserem Beispiel gleich dem Dateinamen der Datei (.dat) sein. Hier ist die flexible Einleseroutine:

```
 10 REM ******************************
 20 REM ** Flexible Einleseroutine **
 30 REM ******************************
 40 :
 50 INPUT "Dateiname : ";file$
 60 OPENIN file$+".inf"
 70   INPUT #9,datsaetze,datfelder
 80 CLOSEIN
 90 :
100 DIM d$(datsaetze,datfelder)
110 :
120 OPENIN file$+".dat"
130 FOR I=1 TO datsaetze
140   FOR J=1 TO datfelder
150     INPUT #9,d$(I,J)
160   NEXT J
```

```
170 NEXT I
180 CLOSEIN
190 END
```

Starten Sie das Programm und geben Sie "Test" ein.

In Zeile 100 wird dann immer dem Bedarf entsprechend dimensioniert. In Fällen, wo sich die Anzahl der Datensätze während des Programmablaufes erhöhen kann, muß man dies schon bei der Dimensionierung bedenken. Eine weitere Lösung ist es, die Maximalzahl der Datensätze vorher festzulegen; die Anzahl der Datenfelder aber kann trotzdem dynamisch bleiben, da sich die Anzahl der Datenfelder wohl kaum ändern wird. Das Einlesen der Anzahl der Datensätze und der Datenfelder ist aber trotzdem eine wertvolle Hilfe. Wenn Sie die Zeile 100 gegen die Zeile

```
100 DIM d$(200,datfelder)
```

ersetzen, so haben Sie den Effekt erzielt, daß die Datenfelderanzahl dateiabhängig ist.

Bis jetzt haben wir beim Öffnen von Files immer feste Dateinamen angegeben. Fest bedeutet, daß der Filename nicht variabel ist, in Anführungsstrichen steht und schon bei der Programmierung feststand. Manchmal öffnet man allerdings auch mit Stringvariablen, so wie wir es in unserem Beispiel getan haben (Zeile 60 und 120). Leider ist das Betriebssystem an dieser Stelle nicht ganz ausgereift. Manchmal öffnet AMSDOS hier die Files nicht unter dem richtigen Namen bzw. reagiert mit einer Fehlermeldung. Wenn Sie sich nach dieser Fehlermeldung die entsprechende Stringvariable ausgeben lassen, so können Sie nur ganz unverständlich erkennen, daß der Inhalt der Variablen stimmt.

Beim Öffnen eines Files wird ein 4096 Bytes großer Buffer im RAM des Schneider CPC angelegt. Dieser Buffer ist frei verschieblich und wird dazu benutzt, die einzulesenden Daten *von* Diskette bzw. die zu schreibenden Daten *auf* Diskette zwischenzuspeichern. Dieser Speicher liegt zumeist in den oberen

Regionen des Speichers. Dabei wird der Zeiger für die größte unter BASIC verfügbare Speicherstelle (HIMEM) um genau diese 4096 Bytes erniedrigt, und zwar solange, bis die eröffnete Datei wieder geschlossen wird. Der Floppy-Buffer liegt dann über diesen Speicherendezeiger HIMEM (Siehe auch dazu die Grafik).

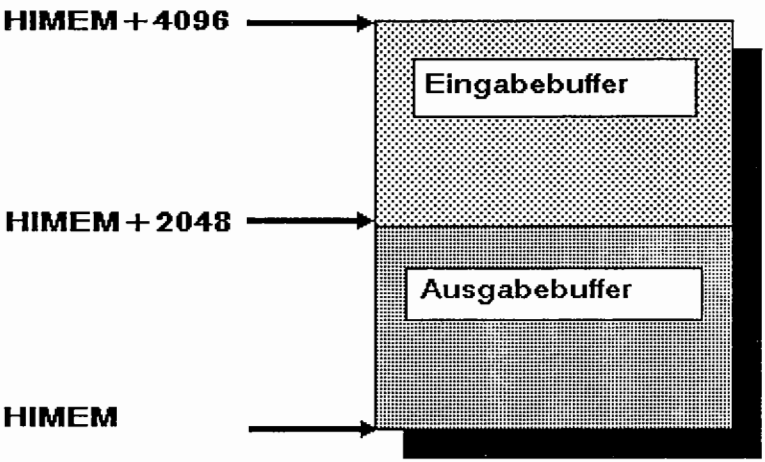


Abb. 13 Floppy-Buffer

In diesen oberen Regionen des RAM tummeln sich aber auch die Stringvariablen. Hier kann es leicht zu "Kollisionen" zwischen Strings und dem Floppy-Buffer kommen, besonders dann, wenn Sie mit Feldvariablen arbeiten, da dann die Zahl der Strings besonders hoch ist.

Von Zeit zu Zeit ist es notwendig, den für die Stringvariablen zugeordneten Speicher *aufzuräumen*. Anders als bei den numerischen Variablen bleiben bei den Stringvariablen bei Stringmanipulationen *Stringfetzen* im Speicher zurück. Dieses "Aufräumen" nennt man *Garbage Collection*, es ist eine Art Speicherreorganisation, die den Stringspeicher neu organisiert. Hierbei kann es vorkommen, daß dieser Disketten-Buffer der Reorganisation zum Opfer fällt. Wenn der Buffer durch die Garbage Collection verschoben wird, so kann der Filename nicht mehr ordnungsgemäß gelesen werden. Mit folgender kleinen Routine kann man das ganze umgehen:

```
OPENOUT "Dummy"
  MEMORY HIMEM-1
CLOSEOUT
```

Diese Anweisungsfolge sollte man an den Anfang jedes Programmes setzen, das mit der Diskette arbeitet. Es wird der 4096-Bytes-Buffer vorsorglich angelegt und durch das MEMORY-Kommando geschützt. (Der HIMEM-Pointer wird entsprechend verschoben.) Sie müssen dieses Kommando verwenden, bevor auch nur eine Stringvariable belegt wird. Ab sofort kann der Diskettenbuffer nicht mehr durch die Garbage Collection zerstört werden, da der Buffer durch das MEMORY-Kommando gesichert ist.

Der Name "Dummy" in der OPENOUT-Anweisung bezeichnet in diesem Falle kein File (es wird ja nicht hineingeschrieben), sondern steht in der Computerfachsprache für eine Art "elektronischer Mülleimer". Durch das Kommando OPENOUT "Dummy" erreicht man, daß der o.g. 4096-Bytes-Buffer angelegt wird. Die Systemvariable HIMEM wird dann entsprechend dezimiert, damit der Buffer vor Variablen geschützt ist. Nach einem CLOSEOUT würde HIMEM wieder heraufgesetzt, um dies aber zu verhindern, wird das MEMORY-Kommando verwendet.

Sollten Sie also in unserem Beispielprogramm, das diese Routine ja noch nicht enthielt, eine Fehlermeldung erhalten haben, so fügen Sie noch die Zeile 40 wie folgt ein:

```
40 OPENOUT "Dummy" : MEMORY HIMEM-1 : CLOSEOUT
```

Es besteht jetzt noch die Gefahr, daß der Benutzer einen falschen Dateinamen eingibt, was zum Abbruch des Programmes führen würde. Aber auch das kann verhindert werden, indem man beispielsweise ein File erstellt, in dem alle Filenamen registriert sind - also praktisch ein "privates Inhaltsverzeichnis". Wir nennen dieses File "Filename.dir".

```
OPENOUT "Filename.dir"
PRINT #9,"Test"
CLOSEOUT
```

Unser privates Inhaltsverzeichnis verfügt nun nur über den Eintrag "Test". Nun müssen wir nur noch diese Testroutine einbauen, dann ist eine Fehlermeldung ausgeschlossen.

```
10 REM ******************************
20 REM ** Flexible Einleseroutine **
30 REM ******************************
40 OPENOUT "Dummy" : MEMORY HIMEM-1 : CLOSEOUT
50 INPUT "Dateiname : ";file$
60 GOSUB 1000
70 IF gefunden=0 THEN 50
80 OPENIN file$+".inf"
90   INPUT #9,datsaetze,datfelder
100 CLOSEIN
110 :
120 DIM d$(200,datfelder)
130 :
140 OPENIN file$+".dat"
150 FOR I=1 TO datsaetze
160   FOR J=1 TO datfelder
170     INPUT #9,d$(I,J)
180   NEXT J
190 NEXT I
200 CLOSEIN
210 END
1000 REM ============================
1010 REM  Teste ob Filename erlaubt
1020 REM ============================
1030 OPENIN "Filename.dir"
1040 gefunden=0
1050 WHILE NOT EOF AND gefunden=0
1060   INPUT #9,Name$
1070   IF Name$=file$ THEN gefunden=1
```

```
1880 WEND
1090 CLOSEIN
1100 RETURN
```

Eine solche oder ähnliche Abfrage ist unerläßlich, um ein Programm benutzerfreundlich zu gestalten. Fehleingaben sollten, wenn möglich, stets erkannt und abgeblockt werden.

Es ist immer nur ein geöffneter Eingabekanal und ein geöffneter Ausgabekanal erlaubt. Sie können also maximal zwei Kanäle zur Floppy öffnen. *Kanäle* sind der Vermittler zwischen dem Betriebssystem des CPC und dem AMSDOS, das für die Speicherung und das Lesen der Daten auf Diskette zuständig ist. Beispielsweise gibt es im Betriebssystem des CPC auch einen Kanal zum Bildschirm und einen Kanal zur Tastatur. Jeder dieser Kanäle hat eine logische Nummer, über die man dann die verschiedenen Geräte ansprechen kann.

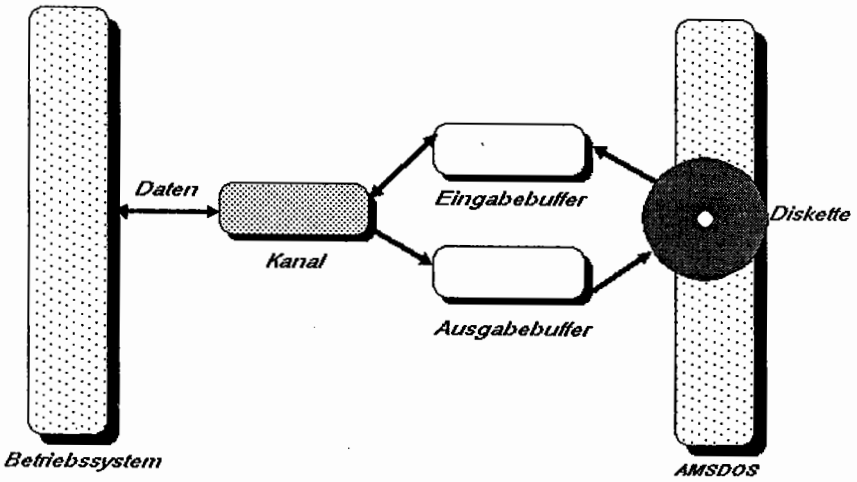


Abb. 14 Kanäle & Buffer

In der Unterroutine, die in Zeile 1030 beginnt, wird getestet, ob es ein File mit Namen file$ gibt. Dazu werden alle vorhandenen Einträge in der Datei "Filename.dir" mit der Variablen file$ verglichen. Sollte die Routine auf das Ende des Files stoßen, ohne den Namen gefunden zu haben, so ist die Variable "gefunden" gleich null(0). Bei einem Erfolg wird die Variable auf 1 gesetzt. Diesen Wert kann man dann im Hauptprogramm abfragen und entsprechend darauf reagieren.

Wenn Sie beispielsweise ein komfortables Adreßverwaltungsprogramm selbst schreiben wollten, so wäre es schon zu empfehlen, ähnliche Schutzroutinen einzubauen.

Natürlich muß man schon voraussetzen, daß das File "Filename.dir" existiert, sollte dies nicht der Fall sein, so wird eine Fehlermeldung File not found ausgegeben. Um so etwas zu vermeiden, könnte man einen Menüpunkt vorsehen, der sich "Zurücksetzen des Systems", "Initialisierung" oder ähnlich nennt, der dieses Files anlegt.

All diese "Weisheiten" sind in dem kleinen Dateiverwaltungsprogramm enthalten, das Sie in diesem Buch in Kapitel 5 vorfinden. Es empfiehlt sich, die entsprechenden Passagen im Programm durchzuarbeiten, damit Sie die Handhabung der Routinen verstehen.

1.5.4 Unterschiede zwischen PRINT # und WRITE

Bisher haben wir lediglich das Kommando PRINT # benutzt. Sie haben sicherlich aber auch schon sowohl im Programmiererhandbuch als auch in diesem Buch gesehen, daß es auch ein Kommando WRITE # gibt.

In unseren Beispielprogrammen wurde *Carriage Return* als Trennzeichen verwandt. Sie können sich aber erinnern, daß das Carriage Return nicht das einzige Trennsymbol ist: Das Komma (,) gilt ebenso als Trennsymbol, sowohl bei den Strings als auch bei den numerischen Daten. Bei den numerischen Daten kann es gleich, ob man nun mit Komma, Carriage Return oder gar Leerzeichen trennt. Bei Stringvariablen allerdings kann so einiges Unangenehme passieren. Probieren Sie einmal folgendes Beispiel aus:

```
10 REM ************************
20 REM ** Beispiel für PRINT # **
30 REM ************************
40 :
50 OPENOUT "Demo"
60   PRINT #9,"Berghoff, Dagmar"
70   PRINT #9,"Ratlos, Susi"
80 CLOSEOUT
90 :
100 REM ========================
110 REM    Und wieder einlesen
120 REM ========================
130 :
140 OPENIN "Demo"
150   INPUT #9,Name1$
160   INPUT #9,Name2$
170 CLOSEIN
180 PRINT "Name1 = "Name1$
190 PRINT "Name2 = "Name2$
200 END
```

Sie erwarten sicherlich auch, daß die Ausgabe auf den Bildschirm wie folgt aussieht:

```
Name1 = Berghoff, Dagmar
Name2 = Ratlos, Susi
```

Einige unter Ihnen wissen es sicherich schon: So sieht das Ergebnis leider nicht aus. Wenn Sie das Programm einmal starten, können Sie folgendes "blaue Wunder" erleben:

```
Name1 = Berghoff
Name2 = Dagmar
```

Sie sehen an diesem Beispiel sehr deutlich, daß das Komma als Trennsymbol gilt und bei der INPUT #9- Anweisung dazu geführt hat, daß die zwei Strings getrennt wurden. Es fehlen außerdem sowohl das Komma in der Zeichenkette als auch das Leerzeichen vor "Dagmar". Die Lösung des Problems ist sehr einfach: Wenn man das WRITE #-Kommando benutzt, so wird ein auszudruckender String in Anführungszeichen gesetzt. Beim Einlesen dieses Strings werden jetzt auch alle führenden Leerzeichen sowie alle Kommata in den String übergeben. Lediglich die Anführungszeichen werden und sollen nicht übernommen werden.

Sie können sich mit dem WRITE-Kommando auch Werte auf dem Bildschirm ausgeben lassen und so die Funktionsweise des Kommandos verdeutlichen. Geben Sie einmal ein:

```
WRITE 1,2,"Ja, ja, so ist das.",3
```

Auf dem Bildschirm erscheint:

```
1,2,"Ja, ja, so ist das.",3
```

Wenn Sie dasselbe mit der PRINT-Anweisung ausgeben, so erscheint folgendes Bild auf dem Monitor:

```
1      2      Ja, ja, so ist das.  3
```

Ein Komma in der PRINT-Anweisung läßt den Tabulator bekanntlich an die nächste Tabulatorstelle springen, so daß große Leerräume entstehen. Genauso wie auf dem Bildschirm wird auch auf Diskette geschrieben, d.h. daß auch entsprechend viele Leerräume auf Diskette übernommen werden (nebenbei auch nicht gerade speicherplatzsparend).

Deutlich zu erkennen ist aber, daß der String aufgrund der Kommata nicht mehr als ein String erkannt werden kann. Bei dem WRITE-Kommando wird durch die Anführungsstriche unmißverständlich Anfang und Ende des Strings markiert.

Als angenehmer Nebeneffekt ist der erwähnte gesparte Platz auf der Diskette zu nennen, wenn man numerische Daten speichern will.

Man könnte das WRITE-Kommando auch durch das PRINT-Kommando simulieren:

```
PRINT 1",";2;",",CHR$(34);"Ja, ja, so
ist das.";CHR$(34);",";3
```

Jetzt sähe das Ergebnis sowohl auf dem Bildschirm als auch auf der Diskette gleich aus, man würde beim Einlesen auch den gewünschten Effekt erzielen. Jedoch ist das WRITE-Kommando einfacher und bequemer, darum sollte man es auch meistens verwenden. Besonders praktisch ist es dann, wenn man mehrere Datenfelder in einer BASIC-Zeile auf Diskette schreiben will. Wie Sie dem Beispielprogramm entnehmen können, haben wir pro PRINT #-Zeile nur ein Datenfeld auf Diskette gespeichert und somit, da wir am Ende der PRINT #Anweisung kein Semikolon stehen haben, das Carriage Return als Trennsymbol

senden lassen. Mit der WRITE-Anweisung wird unser Programm somit kürzer und übersichtlicher und sieht nun so aus:

```
10 REM ************************
20 REM ** Beispiel für WRITE # **
30 REM ************************
40 :
50 OPENOUT "Demo"
60   WRITE #9,"Berghoff, Dagmar"
70   WRITE #9,"Ratlos, Susi"
80 CLOSEOUT
90 :
100 REM ========================
110 REM    Und wieder einlesen
120 REM ========================
130 :
140 OPENIN "Demo"
150   INPUT #9,Name1$
160   INPUT #9,Name2$
170 CLOSEIN
180 PRINT "Name1 = "Name1$
190 PRINT "Name2 = "Name2$
200 END
```

Nach dem Starten des Programmes sehen Sie, daß wir nun zum gewünschten Ergebnis gekommen sind.

Nach einer PRINT #- und einer WRITE #-Anweisung also automatisch ein Trennzeichen, das Carriage Return. Es gibt aber Fälle, in denen man darauf verzichten möchte, beispielsweise wenn Sie Zeichenketten berechnen müssen und diese nach und nach ausgeben. Wir wollen nun einmal das gesamte Alphabet ausgeben und lösen dieses Problem aber nicht mit einer simplen PRINT #-Anweisung, sondern programmieren eine Schleife.

```
10 REM ****************************
20 REM ** Beispiel für Anhängen **
30 REM ****************************
40 :
50 OPENOUT "Demo"
60 FOR i=1 TO 26
70   PRINT #9,CHR$(64+i);
80 NEXT i
90 PRINT #9
100 CLOSEOUT
110 OPENIN "Demo"
120 INPUT #9,a$
130 PRINT a$
140 CLOSEIN
150 END
```

In der Zeile 60 definieren wir eine Schleife i, die von 1 bis 26 hochzählt. Bekanntlich hat das Alphabet 26 Buchstaben. Der ASCII-Code von "A" ist 65, so daß wir in der Zeile 70 den ASCII-Code des zu druckenden Zeichens aus aktuellem Schleifenwert und dem Offset 64 zusammensetzen können. (ASCII-Code steht für *A*merican *S*tandard *C*ode for *I*nternational *I*nterchange und wird von den meisten HOME- und PERSONAL-Computern zur Kodierung des Alphabets und der Sonderzeichen benutzt.) Wichtig ist aber, daß das letzte Zeichen in der Zeile 70 ein Semikolon ist. Genauso wie im normalen BASIC, wenn man auf dem Bildschirm ausgibt, hat das Semikolon bei der Diskette die Wirkung, daß kein "Datenfeld-Ende" gesendet wird, vergleichbar also mit einem fehlenden "Zeilen-Ende" auf dem Bildschirm.

Nach Starten des Programmes sehen wir, daß die Variable a$ das vollständige Alphabet beinhaltet. Sicher ist nun die Bedeutung des Semikolons geklärt. In diesem Beispiel hätten Sie keinesfalls das WRITE-Kommando verwenden dürfen. Die Ausgabe hätte es dann so ausgesehen:

```
"A""B""C""D""E""F""G""H"...."Y""Z"*
```

während wir folgendes Resultat in unserem Beispiel erhielten:

```
ABCDEFGH...YZ*

(* steht wieder für Carriage Return)
```

Sie sehen, daß man sich bei jeder Anwendung überlegen muß, welches Kommando sich optimal eignet. Oft ist es gleich, ob man nun PRINT #9 oder WRITE #9 verwendet. Manchmal ist WRITE #9 günstiger, manchmal aber auch, wie in diesem Beispiel, vollkommen unbrauchbar.

Überlegen Sie einmal, was die Variable a$ für einen Wert erhielte, wenn man die Zeile 70 ersetzen würde durch:

```
70  WRITE #9,CHR$(i+64);
```

Wenn Sie zu einem Ergebnis gekommen sind, tauschen Sie einmal die Zeile aus und starten Sie das Programm. Hatten Sie den richtigen Schluß gezogen? Dann können Sie zum nächsten Kapitel übergehen.

1.5.5 Unterschiede zwischen INPUT # und LINE INPUT

Die Unterschiede zwischen PRINT #9 und WRITE #9 sind Ihnen jetzt deutlich geworden. Auch zum Einlesen von Daten existieren zwei verschiedene Befehle: der normale INPUT #9-Befehl, den wir bislang immer benutzt haben, sowie der Befehl LINE INPUT #9. Auch hier gibt es wesentliche Unterschiede.

Wir beschränken uns bei unserem Vergleich von INPUT und LINE INPUT auf Stringvariablen, da nur bei Stringvariablen der Sinn der LINE-INPUT-Anweisung deutlich wird.

Wir wissen, daß Datenfelder durch Trennsymbole voneinander abgegrenzt sind. Trennsymbole können sein: Der Wagenrücklauf, das Komma und das EOF-Zeichen. Wir können also keinen String mittels INPUT #9 einlesen, der ein Komma beinhaltet (es sei denn, wir setzen diesen String in Anführungszeichen), da dieses Komma automatisch als Trennsymbol interpretiert wird. Daraus resultiert ein weiteres Problem: Wenn Anführungszeichen einen String markieren, wie kann ich dann Anführungszeichen lesen? Anführungszeichen können mittels INPUT überhaupt nicht in einen String eingelesen werden, so viel ist klar.

An dieser Stelle setzt das Kommando LINE-INPUT ein. LINE-INPUT liest alle Zeichen ein; es gilt nur der Wagenrücklauf als Trennsymbol, also werden auch Kommas und Anführungszeichen von LINE INPUT akzeptiert.

Wenn Sie das Kommando

```
WRITE #9,1,2,"Ja, ja, so ist das.",3
```

verwenden, um auf Diskette zu schreiben, und dies dann wieder mit

```
LINE INPUT #9,Alles$
```

einlesen, so wird die Variable Alles$ den Inhalt

```
1,2,"Ja, ja, so ist das.",3
```

haben. Es wurden also alle Trennsymbole ohne Ausnahme in den String übernommen, die Anführungszeichen eingeschlossen.

Nicht immer ist LINE INPUT dem Kommando INPUT überlegen; meistens würde es wohl sogar katastrophale Folgen haben, wenn Sie in einem bestehenden Programm aus INPUT #9 einfach LINE INPUT #9 machten. Sie können sich sicherlich schon denken, daß dies daran liegt, daß die Trennsymbole überlesen

werden. Wenn man beim PRINT- bzw. WRITE-Kommando nicht bedenkt, daß mit LINE INPUT eingelesen wird und dementsprechend nur mit Wagenrücklauf trennt, kann man so manches Wunder erleben. Wenn Sie sich selbst ein Programm schreiben, das häufig auf Diskette zugreift, so überlegen Sie sich gut, welche Befehle Sie zum Schreiben und Lesen verwenden.

Doch hier einmal ein Beispiel, wo der LINE-INPUT-Befehl unerläßlich ist, weil eine von Inhalt und Aufbau her unbekannte Datei eingelesen werden soll.

Laden Sie einmal eines der in diesem Buch erstellten Programme und speichern Sie es direkt wieder mit

```
OPENOUT "ASCII.DAT"
LIST #9
CLOSEOUT
```

Sie haben nun das Listing des Programmes auf Diskette unter dem Namen "ASCII.DAT" gespeichert. Ein normales Programmfile kann man nicht mittels OPENIN-Befehl zum Lesen öffnen, da dieses File einen Header (Vorspann) hat, der durch die OPENIN-Anweisung nicht akzeptiert wird. Nachdem Sie also ein Programmfile auf Diskette geLISTet haben, geben Sie folgendes kleine Programm ein:

```
NEW

10 REM ***************************************
20 REM ** Auslesen einer beliebigen Datei **
30 REM ***************************************
40 :
50 OPENIN "ASCII.DAT"
60 WHILE NOT EOF
70   LINE INPUT #9,zeile$
80   PRINT zeile$
90 WEND
```

```
100 CLOSEIN
110 END
```

Diese Routine zeigt Ihnen Ihr Programmlisting auf dem Bildschirm an, mit allen Anführungszeichen und Kommata wie im Originalprogramm. Für solche und ähnliche Aufgaben ist das LINE INPUT-Kommando sehr gut geeignet, da es problemlos alle Zeichen von Diskette holt.

Leider ist es im Amstrad-BASIC nicht vorgesehen, einzelne Zeichen aus dem zum Lesen geöffneten File zu lesen, so wie man es in anderen BASIC-Versionen auf anderen Rechnern kennt (GET-Funktion). Diese Möglichkeit hätte natürlich den Vorteil, selbst das Zeichen Wagenrücklauf lesbar zu machen. Jedoch existiert im AMSDOS eine Routine *DISC IN CHAR*, die Sie dazu nutzen können (siehe auch DOS-Listing) - allerdings nur in Maschinensprache. Sie können sich dann individuell für Ihr Problem eine Routine schreiben, die das gelesene Zeichen an Ihr BASIC-Programm übergibt.

Hier ist eine kleine Routine, die den GET-Befehl in BASIC simuliert. Wenn Sie die Routine anspringen, muß das File zum Lesen geöffnet sein.

```
10000 REM ===================================
10010 REM    GET-Routine
10020 REM ===================================
10030 :
10040 IF LEN(in$)=0 THEN 10080
10050 ch$ = LEFT$(in$,1)
10060 in$ = MID$(in$,2)
10070 RETURN
10080 IF EOF THEN CLOSEIN : ch$="" : RETURN
10090 LINE INPUT #9,in$
10100 IF NOT EOF THEN in$ = in$ + CHR$(13)
10110 GOTO 10050
```

Es wird Ihnen in der Variablen ch$ das gelesene Zeichen übergeben, auch der Wagenrücklauf ist jetzt ein legales Zeichen. Wenn das letzte Zeichen gelesen wurde, ist ch$ leer, LEN(ch$) ist also null.

Lesen Sie auf Ihrem Bildschirm numerische Werte (Zahlen) ein, so bedienen Sie sich meistens des Kommandos

```
10 INPUT wert
```

Wenn Sie nun ein Zeichen eingeben, das keine Zahl ist, so meldet sich das System mit

```
?Redo from start
```

AMSDOS kann natürlich nicht eine solche Fehlermeldung ausgeben. Manche Systeme geben dann einen FILE-TYPE-ERROR aus, was soviel bedeutet wie: Es wurde ein String anstatt eines numerischen Wertes übergeben. AMSDOS gibt keine Fehlermeldung aus, sondern führt intern folgende Berechnung durch:

```
10 INPUT #9,a$ : wert=val(a$)
```

Hier wird deutlich, warum das Leerzeichen als Trennsymbol bei Zahlen gilt. Darüber hinaus fungiert jedes nicht numerische Zeichen als Trennsymbol, mit Ausnahme des "E" bzw. des "e", das für die Exponentialschreibweise benötigt wird.

1.6 Relative Datenspeicherung

1.6.1 Was ist relative Datenspeicherung?

Wie bereits schon einmal erwähnt, stützen sich alle existierenden Datenspeicherungstechniken (für Daten) entweder auf die sequentielle oder die relative Datenspeicherung. Die sequentielle Datenspeicherung haben wir nun genau betrachtet, jetzt wollen wir auch wissen, was man unter der relativen Datenspeicherung zu verstehen hat.

Dabei stellt sich allerdings direkt ein Problem für Sie als CPC-Besitzer. Leider wurde im Betriebssystem des Schneiders die relative Datenspeicherung nicht vorgesehen; es existieren also keine Befehle, so daß das Abhandeln dieses Themas anhand von Beispielen Schwierigkeiten bereiten mag. Aber auch diesem Problem wollen wir gemeinsam entgegentreten, wenn wir den theoretischen Teil der relativen Datenspeicherung abgehandelt haben.

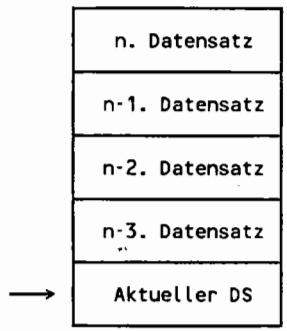
Bei sequentiellen Dateien können wir eine Datei entweder zum Schreiben *oder* zum Lesen eröffnen. Wird die Datei zum Schreiben eröffnet, so wird der alte Inhalt der Datei vollkommen überschrieben; die Datei wird praktisch unter demselben Namen neu eingerichtet, dabei spielt der alte Inhalt keine Rolle. Das Betriebssystem des Schneider CPC erstellt zwar eine Sicherheitskopie, die BAK-Datei, doch dies stellt nur einen kleinen Trost dar.

Bei etwas komfortableren Betriebssystemen kann man an eine bestehende sequentielle Datei auch noch weitere Daten anhängen. Man eröffnet eine Datei also nicht zum Schreiben, dies würde den alten Inhalt ja zerstören, sondern zum Anhängen. Der *interne Pointer* zeigt nach dem Öffnen direkt auf das letzte Element. Folgende Schreibaktionen werden dann an die bestehenden Daten angehangen. In der englischen Sprache nennt man diese Aktion *Append*. Der alte Inhalt einer Datei bleibt also erhalten und man hängt lediglich *neue* Daten an das Ende dieser beste-

henden Datei an. Dies kann beispielsweise bei einem Journal in der Buchführung interessant sein.

Vollkommen anders sieht dies in der relativen Datenspeicherung aus. Hier wird eine Datei immer zum Lesen und Schreiben gleichzeitig eröffnet.Wir müssen uns zunächst von der Vorstellung freimachen, daß ein Zeichen (Byte) dem anderen gezwungenermaßen folgt. Anders als bei der sequentiellen Datenspeicherung kann man bei der relativen Datenspeicherung frei auf jedes beliebige Zeichen der Datei zugreifen.

Wir haben uns die sequentielle Datei als Karteikasten vorgestellt, in dem eine Karteikarte hinter der anderen steht. Dabei stellt jede Karteikarte einen Datensatz dar.



Wenn Sie in unserem Beispiel an den nten Datensatz kommen wollen, so müssen alle davorstehenden Datensätze *überlesen* werden. Ein zeitaufwendiges Unterfangen. Wenn wir uns wieder an unser Karteikastenmodell anlehnen, so müßten Sie jede Karteikarte, die vor der nten steht, durchlesen.

Bei der relativen Datenspeicherung nun könnten Sie direkt mit dem internen Zeiger auf den n. Datensatz zeigen und den Inhalt auslesen, *ohne* daß Sie sich um den Inhalt der weiteren drei eigentlich vorher zu lesenden Datensätze kümmern müßten; der

interne Zeiger ist also *frei beweglich.* Diese freie Beweglichkeit soll auch folgende Grafik verdeutlichen:

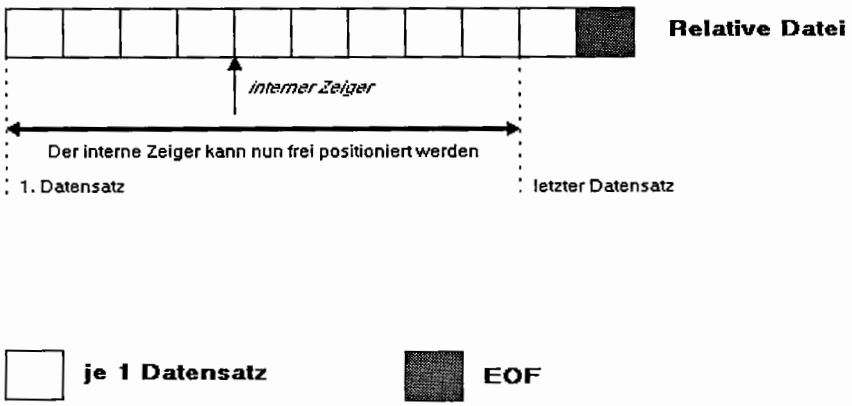


Abb. 15 Interner Zeiger

Wenn wir eine relative Datei zur Verfügung stehen haben, wäre es bei unserem Karteikastenmodell vergleichbar mit dem gezielten herausnehmen einer beliebigen Karteikarte

Die relative Datenspeicherung wird in der englischen Sprache *Random Acces* genannt, was soviel bedeutet wie freier Zugriff. Auch den Begriff "relativ" kann man "relativ" leicht erklären: Man greift immer relativ zum Dateianfang einen Datensatz heraus.

Stellen Sie sich vor, Sie ziehen eine Schublade auf, in der 100 Aktenmappen hängen. Jede Aktenmappe stellt einen Datensatz dar, mit zahlreichen Informationen über einen Kunden

beispielsweise. Sie können nun jede beliebige Aktenmappe herausgreifen; Sie erkennen die gewünschte beispielsweise an einer Beschriftung.

Nun hat die relative Datenspeicherung nicht nur Vorteile. Sind Zugriff und Handhabung zwar bequemer als bei der sequentiellen Datenspeicherung, so ist es notwendig, daß man sich vor der Einrichtung einer relativen Datei unbedingt Gedanken über das Mengengerüst macht. Hierzu zählt die Größe des Datensatzes sowie die Anzahl der voraussichtlichen Datensätze, die diese Datei umfassen soll. Anders als bei der sequentiellen Datei muß man bei der relativen Datei nämlich zunächst eine Datei *einrichten*, bevor man sie beschreiben kann. Hier gibt es allerdings auch Unterschiede: bei manchen Betriebssystemen kann man später noch weitere Datensätze anhängen - bei anderen Betriebssystemen ist das nicht möglich. Wir wollen aus zwei Gründen davon ausgehen, daß wir keine Datensätze mehr anhängen können:

a) mit "unserer" relativen Dateiverwaltung geht dies auch nicht, da dies nicht vorgesehen wurde,

b) man sollte sich ohnehin zunächst immer Gedankenüber das Mengengerüst einer Datei machen, da dann die Fehleranfälligkeit geringer ist und man keine bösen Überraschungen erlebt.

1.6.2 Das Einrichten einer relativen Datei

Warum aber - so werden Sie sich nun vielleicht fragen - muß ich mir bei der relativen Datenspeicherung Gedanken über die Größe eines jeden Datensatzes machen?

Wir haben bereits erwähnt, daß die relative Datenspeicherung von Seiten des Betriebssystemes nur dann ermöglicht werden kann, wenn genau feststeht, welche Länge jeder Datensatz hat.

Die Einschränkungen sind nicht ganz so arg, wie Sie jetzt vielleicht vermuten.

Diese definierte maximale Datensatzlänge dürfen Sie selbstverständlich *unterschreiten*, so daß Sie ausschließlich eine maximale Datensatzgröße festlegen müssen. Allerdings müssen Sie im Programm dann darauf achten, daß diese Datensatzlänge auch nicht überschritten wird. Sollte dies vorkommen, so wird mit an Sicherheit grenzender Wahrscheinlichkeit der nachfolgende Datensatz überschrieben und somit ganz oder teilweise gelöscht.

Es ist auch von der Maschine, auf der Sie programmieren, abhängig, ob Sie lediglich die Daten*satz*länge definieren müssen oder ob die Länge jedes Daten*feldes* explizit definiert wird. Beim IBM-PC ist es beispielsweise so, daß die Länge jedes Datenfeldes definiert werden muß. Eine sicherlich sehr sinnvolle Eigenschaft des sowieso sehr leistungsstarken MS-BASICs, da man auf diese Weise festgesteckte Grenzen der Datenfelder nicht überschreiten darf, was einen disziplinierteren Programmierstil zur Folge hat.

In "unserer" relativen Dateiverwaltung (die in Kapitel 5 realisiert wird) ist es allerdings so, daß die Länge der einzelnen Datenfelder durchaus variieren darf, nur die Summe der Längen der Datenfelder darf nicht die maximale Datensatzlänge überschreiten.

Wir wollen nun aber etwas genauer betrachten, wie man diese vielzitierte maximale Datensatzlänge berechnet.

Ein Datensatz - das haben wir bereits in Kapitel 1.5 kennengelernt - setzt sich aus einem oder mehreren Datenfeldern zusammen. Welche Datenfelder man benötigt, ist natürlich stark situationsabhängig. Wir wollen in unserem Beispiel einen Bezug zur konventionellen Datenverarbeitung schaffen und nehmen deswegen als Beispiel einen einfachen Adreßdatensatz. Dieser Datensatz soll folgende Datenfelder beinhalten:

Name (inkl Vorname)	35 Zeichen
Straße (+Hausnr.)	20 Zeichen
PLZ + Ort	22 Zeichen
Telefon	15 Zeichen
Gesamt:	92 Zeichen

Jeder Datensatz umfaßt also 92 Zeichen. Programmieren wir auf einem System, das flexible Datenfeldlängen zuläßt, so müssen wir noch 4 Zeichen zu diesen errechneten 92 hinzuaddieren. Diese 4 Zeichen benötigen wir als Trennsymbol der einzelnen Datenfelder. Auch die relative Dateiverwaltung, die in diesem Buch abgedruckt ist, benötigt diese 4 Trennsymbole - beziehen Sie also die Trennsymbole bitte in Ihre Rechnung mit ein. Wir kommen dann auf eine totale Datensatzlänge von:

96 Zeichen

Erreicht ein Datensatz nicht diese Länge, so wird der restliche Platz für diesen Datensatz auf der Diskette "aufgefüllt".

Nachdem wir nun die Länge unseres Datensatzes errechnet haben, sollten wir uns auch Gedanken darüber machen, wie viele dieser Datensätze wir benötigen. Wie bereits erwähnt, gibt es Systeme, bei denen man später noch weitere Datensätze anhängen kann - doch dies ist nicht die Regel und auch bei unserer relativen Dateiverwaltung nicht möglich. Deswegen sollte man genauestens abwägen, welche Menge an Datensätzen man in Zukunft erwartet, da diese Anzahl dann entsprechend eingerichtet werden muß.

Nehmen wir einmal an, Sie hätten bereits 60 solcher Karteikarten, die Sie nun per Computer verwalten wollen. Sie

erwarten demnächst einen Zuwachs von 50%. Dann sollten Sie nicht sparen und ruhig 120 bis vielleicht sogar 150 Datensätze vorsehen und einrichten.

Wie Sie diese Datensätze einrichten müssen, das verraten wir Ihnen im Kapitel 5, wenn Ihnen auch die anderen neuen Kommandos erklärt werden.

1.6.3 Wie arbeite ich mit einer relativen Datei?

Ist erst einmal genügend Platz auf der Diskette geschaffen worden, um die vorgesehene Anzahl an Datensätzen aufzunehmen, so ist eigentlich auch schon das wichtigste getan, um sich über diese relative Datei herzumachen. Anders als bei der sequentiellen Datenspeicherung können wir nun darauf verzichten, die gesamte Datei in den Speicher - beispielsweise in eine Feldvariable - einzulesen. Da die Handhabung der relativen Datenspeicherung freien Zugriff schnell und problemlos ermöglicht, kann man eine relative Datei fast schon als *externe Feldvariable* bezeichnen.

Dieser Vergleich ist wirklich nicht an den Haaren herbeigezogen: Wir haben auf der Diskette die relative Datei eingerichtet. Sie können dies mit der Dimensionierung einer Feldvariablen durchaus vergleichen. Beim DIM-Kommando wird Platz für die Variable im Speicher geschaffen.

Genauso einfach wie bei einer Feldvariablen können Sie auch bei der relativen Datei auf jede *vorhandene* Komponente zugreifen. Man nennt diese "Komponente", welche ja einen Datensatz darstellt, in der Fachsprache RECORD. So hat jeder Datensatz (Record) auch eine fest zugeordnete Recordnummer, die ihn identifiziert. Stellt sich nur noch die Frage, ob wir unsere 150 Datensätze mit 0 bis 149 oder mit 1 bis 150 durchnumerieren. Auf den allermeisten Anlagen beginnt man mit der

Recordnummer 0, so daß Sie eigentlich nie etwas falsch machen, wenn Sie mit der Recordnummer 1 beginnen, Sie vergeuden höchstens einen Record.

Kleiner Tip: Irgendwo müssen Sie sich ja auch auf einem externen Datenspeicher merken, wie viele Records bereits in Ihrer Datei belegt sind. Sie können diese und ähnliche Daten selbstverständlich in einer sequentiellen Datei ablegen, es hat sich jedoch in der Praxis als nützlich erwiesen, den Record 0 ausschließlich für solche Zwecke als Speicher zu "mißbrauchen". So fehlt einem auch nicht der Bezug zwischen irgendeiner sequentiellen und einer relativen Datei. Vergessen Sie dann aber nicht, sicherheitshalber einen Datensatz mehr in Ihre Berechnung aufzunehmen.

Wie bereits erwähnt, wird eine relative Datei immer zum Lesen und Schreiben *gleichermaßen* eröffnet. Da auf eine relative Datei zumeist auch sehr häufig zugegriffen wird, eröffnet man sie normalerweise ganz zu Anfang und schließt sie erst wieder kurz vor Verlassen des Programmes. Man erspart sich auf diese Weise viel Zeit, die ja beim Öffnen und Schließen ohne Zweifel verloren geht. Leider ist es beim Schneider CPC nicht möglich, eine relative und eine sequentielle Datei gleichzeitig offen zu haben. Das liegt daran, daß der Platz für Dateibuffer mit für nur zwei offene Dateien beim CPC recht mager ausgefallen ist - doch dies können wir leider nicht ändern. Da die relative Datei zum Schreiben und Lesen gleichzeitig geöffnet wird, werden beim CPC auch beide Buffer belegt.

Ist eine relative Datei erst einmal eröffnet worden, so ist der Zugriff auf einen beliebigen Record kein Problem mehr. Die Recordnummer reicht zur Identifikation vollkommen aus. Man gibt die Recordnummer in der Regel bei Lese- oder Schreibkommando mit an, beispielsweise:

```
GET #1,11,A$
```

In PASCAL wird man die Recordnummer als ersten Parameter des WRITE-Kommandos übergeben. Es kommt auch vor, daß man den gewünschten Record durch ein explizites Kommando wie beispielsweise SEEK oder RECORD auswählt. Bei unserer relativen Dateiverwaltung ist dies der Fall: Bevor man einen Datensatz lesen oder schreiben kann, wird dieser mittels des RECORD-Kommandos ausgewählt. Dies ermöglich einen schönen Übergang zu der Frage:

1.6.4 Wie funktioniert die relative Datenspeicherung?

Bisher haben wir Ihnen noch verschwiegen, wozu das Betriebssystem die Recordlänge benötigt. Aber ohne Recordlänge ist überhaupt nichts möglich.

Wir haben bereits erwähnt, daß das Betriebssystem bei Datensätzen, die kürzer als die maximale Datensatzlänge sind, den verbleibenden Platz auffüllt. Das kann das Betriebssystem aber nur dann machen, wenn es die maximale Datensatzlänge kennt. Dies ist der erste, nicht aber der wichtigste Punkt, warum das Betriebssystem diese Recordlänge kennen muß.

Viel wichtiger ist diese Information noch bei der Bestimmung der relativen Position zum Dateianfang anhand der Recordnummer. Eine Datei mit 150 Datensätzen zu je 96 Zeichen ist ja nichts anderes als eine Aneinanderreihung von

*150 * 96 = 14400 Zeichen.*

Wenn Sie nun beispielsweise den Datensatz mit der Recordnummer 77 lesen wollen, so befindet der sich relativ zum Dateianfang an der Position:

*77 * 96 = 7392*

Anhand dieser errechneten Zahl 7392 weiß das Betriebssystem, daß es 7392 Zeichen überlesen kann. Bei einer Sektorgröße von 512 Bytes befindet sich der 77. Datensatz somit im

7392 / 512 = 14.4375

14. Sektor der relativen Datei. Beachten Sie, daß es bei dieser Rechenform auch den nullten Sektor gibt. Im Inhaltsverzeichnis sind die durch eine Datei belegten Sektoren verschlüsselt verzeichnet. Man kann dann die Track-/Sektornummer dieses Sektors bestimmen und ihn gegebenenfalls in den Dateibuffer lesen. "Gegebenenfalls" sage ich deshalb, weil ein optimiertes Betriebssystem (wie dies auch bei unsere raltiven Dateiverwaltung der Fall ist) einen Block nur dann in den Buffer liest, wenn es auch wirklich notwendig ist. Wurde derselbe Block bereits vorher gelesen, so ist ein erneutes Lesen natürlich überflüssig.

Nun müssen wir noch bestimmen, an welcher Stelle in unserem errechneten Sektor der Datensatz beginnt. Auch hierfür ist die Recordlänge von großer Bedeutung. Mathematisch gesehen ist diese Berechnung denkbar einfach: Es handelt sich hierbei um die ganz normale Restfunktion, auch bekannt unter dem Namen MODULO-Funktion. Die Position im Block berechnen wir nun wie folgt:

*7392 - INT(7392/512)*512 = 224*

Das 224. Zeichen im Buffer ist also folglich das erste Zeichen im 77. Datensatz. Auf diese Weise wurde auch die relative Datenspeicherung auf dem CPC in diesem Buch realisiert. Allerdings mit einer kleinen Einschränkung bezüglich der

Recordlänge; doch hierzu finden Sie weitere Informationen unter Kapitel 5.3.

1.6.5 Wann ist die relative Datenspeicherung sinnvoll?

Dies ist sicherlich eine interessante Frage, wenn wir Sie nun "heiß" auf die relative Datenspeicherungstechnik gemacht haben sollten. Zugegeben: Die relative Datenspeicherung ist sehr bequem. Doch sollten Sie sich davor hüten, jetzt für jedes kleinere Problem die relative Datenspeicherung zu benützen.

Sie sollten niemals vergessen, daß die Programmierung relativer Dateien in der Regel etwas mehr an Programmierarbeit in Anspruch nimmt. Zur Lösung kleinerer Probleme wäre eine solche Programmierarbeit höchst uneffizient; dies betrifft übrigens auch die Ausführungsgeschwindigkeit.

Wollen Sie also kleinere Tabellen o.ä. abspeichern, so bietet sich die sequentielle Datenspeicherung hierzu geradezu an. Die relative Datenspeicherung beginnt sich ab ca. 50 Datensätze an zu lohnen. Ferner lohnt sich die relative Datenspeicherung dann, wenn man alle Datensätze nur schwer oder gar nicht auf einmal im Speicher halten kann.

Bei großen Adreß- oder Personendateien ist die relative Datenspeicherung sicherlich unverzichtbar. Jedes größere Dateiverwaltungsprogramm wie etwas DBASE oder DATAMAT arbeitet ausschließlich mit relativen Dateien. Beispielsweise beim Sortieren einer solchen Datei nach einem bestimmten Kriterium ist wegen des großen Speicherplatzbedarfes die relative Datenspeicherung die einzig möglich Lösung.

Bevor Sie ein größeres Problem mit der relativen Datenspeicherung lösen wollen, sollten Sie zunächst ein wenig "üben". Hierzu bietet sich ein kleines Programm an, das Adressen speichert, lädt und korrigiert. Wenn Sie dann alles im Griff

haben, dann können Sie sich an größere Datensätze mit größeren Datensatzzahlen heranmachen. Es wäre ja ein Jammer, wenn irgendwann ein Fehler auftritt und Sie feststellen müßten, daß unwillkürlich Datensätze überschrieben werden.,

1.6.6 Die ISAM-Dateien

Man kann wohl ohne zu zögern behaupten, daß *ISAM*-Dateien die in der Praxis am häufigsten vorkommenden Dateiarten sind. ISAM steht für:

*I*ndexed *S*equential *A*ccess *M*ethod

Sie haben schon ganz richtig vermutet: Die "indiziert sequentielle Zugriffsmethode" ist eine Mischung aus relativer und sequentieller Datenspeicherungstechnik. Um den Begriff "Index" zu erläutern, müssen wir noch einen kleinen Schritt in die Welt der Dateiverwaltungstechnik wagen:

Sie erinnern sich bestimmt noch an unseren Datensatz, der vier Datenfelder beinhaltet: Name, Straße, Ort und Telefon. Nehmen wir nun einmal an, wir hätten 100 Datensätze in dieser Datei gespeichert. Was müßten Sie tun, wenn Sie den Datensatz eines Kunden heraussuchen lassen wollten, wenn Ihnen nur der Name bekannt wäre? Richtig: Das Programm müßte - vorausgesetzt, die Datei wäre alphabetisch nach dem Datenfeld Name sortiert worden - alle Datensätze vom ersten bis zum betreffenden Datensatz durchgehen und das Datenfeld Name mit dem zu suchenden Namen vergleichen. Der Vorteil gegenüber einer sequentiellen Datei wäre nicht mehr klar erkennbar.

Es gibt zwar Suchverfahren, die diesen Suchvorgang ein wenig optimieren, beispielsweise den sogenannten binären Suchbaum. Doch sind immer noch einige (unnötige) Zugriffe auf Diskette notwenig. Man hat sich dann überlegt, ein Datenfeld aus dem Datensatz als "Schlüsselfeld" zu bezeichnen, in der englischen

Sprache nennt man dies "Key" oder auch einfach "Index". Dieses Indexfeld soll sich dann resident (also immer) im Speicher befinden. Man wählt hierzu am besten das Feld aus, das am häufigsten als Such- oder Sortierkriterium benutzt wird.

Neben dem Inhalt dieses Datenfeldes wird natürlich noch die zugehörige Recordnummer im Speicher gehalten. Auf diese Weise kann man dann gezielt direkt auf einen Datensatz auf Diskette zugreifen, ohne überflüssige Datensätze lesen zu müssen.

Der Vorteil dieses Verfahrens liegt klar auf der Hand: Einerseits spart man Speicherplatz, indem man nicht den gesamten Datensatz im Speicher behält. Anstatt *96*100=9600* Bytes brauchen wir nun nur noch

*(35 + 1) * 100 = 3600*

Bytes. Diese Summe setzt sich zusammen aus 35 Bytes für das Schlüsselfeld plus 1 Byte für die Recordnummer mal 100 Datensätze.

Zweiter wesentlicher Vorteil ist die enorme Zeitersparnis. Der Computer kann natürlich viel schneller eine Zeichenkette im Speicher vergleichen als jedesmal zunächst auf Diskette zugreifen zu müssen.

Diese Schlüsselwortliste wird übrigens nach jeder Ergänzung, Löschung oder Korrektur aufs neue sortiert; sonst würde ein Datensatz im schlimmsten Falle ja nicht gefunden. Wenn man das Programm beendet, so wird diese Indexliste einfach als sequentielle Datei abgespeichert. Die relative Datei braucht so bei Sortiervorgängen u.ä. nicht umdisponiert zu werden.

Man kann natürlich mehrere Indexdateien anlegen, so daß man für jeden Fall dieses "optimale" Zugriffsverfahren zur Verfügung hat. In der Praxis kommt man aber in der Regel mit einem Indexfeld vollkommen aus. Sollte einmal nach einem anderen Datenfeld gesucht werden müssen, so muß man halt die etwas längere Wartezeit in Kauf nehmen.

1.7 Die verschiedenen Schneider-Rechner

Der CPC 464, der CPC 664 und der CPC 6128, alle drei stammen aus Hause Schneider und sind mehr oder weniger kompatibel. BASIC-Programme, die auf dem CPC 464 laufen, laufen auch auf den anderen beiden Rechnern CPC 664 und CPC 6128, jedoch leider nicht unbedingt umgekehrt.

Der CPC 664 und der CPC 6128 haben ein identisches BASIC, nämlich die Version 1.1. Diese beiden Rechner können BASIC-Programme untereinander austauschen. Vorsicht ist bei den Maschinenprogrammierer geboten: Hier liefern die drei Rechner einen absoluten RAM- und ROM-Chaos. Erfreulicherweise sind das Floppy-ROM und die Floppy-RAM-Adressen von den Versionsänderungen nicht betroffen, so daß alles in diesem Buch gesagte für alle drei Rechner gleichermaßen gilt.

Beim CPC 464 und dem CPC 664 wird das CP/M 2.2 mitgeliefert. Da der CPC 6128 ein 128 KByte-Rechner ist, wird bei diesem Rechner sowohl CP/M 2.2 als auch CP/M 3.0 mitgeliefert. Genauere Beschreibung der beiden CP/M-Versionen sowie Aufklärung der Unterschiede und der Bedienung finden Sie im CP/M-Trainingsbuch zum CPC.

Hier eine kleine Tabelle, die Aufschluß über die Unterschiede der drei Schneider-CPC-Rechner geben soll:

	CPC 464	CPC 664	CPC 6128
Eingebaute Floppy	Nein	Ja	Ja
Speicher (BASIC)	64 (48)	64 (48)	128 (48)
CP/M-Version	2.2	2.2	2.2 & 3.0
BASIC-Version	1.0	1.1	1.1

1.7.1 Der CPC 664 und der CPC 6128 und Diskettenfehler

Tritt beim CPC 464 ein Fehler auf - das haben wir gelernt -, so wird unweigerlich das eventuell laufende Programm unterbrochen und die Fehlermeldung auf dem Bildschirm angezeigt. Eine sehr unangenehme und unkomfortable Sache. Das haben sogar die Leute von Locomotive Software gemerkt und als Trost (mehr ist dies wirklich nicht) eine kleine Änderung im BASIC vorgenommen.

Tritt bei der BASIC-Version 1.1 ein Fehler im Disketten-Handling auf, so kann man den Abbruch des Programmes durch das bekannte ON-ERROR-GOTO-Kommando verhindern. Die Errorvariable ERR enthält dann den Fehlercode 32. Sie wissen dann, daß es sich um einen Diskettenfehler handelt.

Ferner hat man dem BASIC-Programmierer die Fehlervariable DERR spendiert, die genauere Auskunft über den Diskettenfehler geben soll. Leider ist diese übermittelte Fehlernummer nur sehr unzureichend. Ferner wird der Fehlertext nach wie vor auf dem Bildschirm angezeigt. Handelt es sich dabei auch noch um die Frage

Retry, Ignore or Cancel?

so müssen Sie zunächst antworten, bevor es weitergehen kann. Sicherlich - ein kleiner Schritt vorwärts, aber immer noch sehr unkomfortabel, denn: was soll der Benutzer denken, wenn Meldungen auf dem Bildschirm erscheinen, mit denen er nichts anzufangen vermag?

Wenn man dann über die Kompatitbilität nachdenkt: Wenn Sie ein Programm schreiben, daß vielleicht nicht nur Ihrem privaten Vergnügen dienen soll, sondern auf möglichst vielen Rechnern laufen soll, so können Sie von der ON-ERROR-GOTO-Variante und der Fehlervariablen *DERR* keinen Gebrauch machen, denn CPC-464-Rechner würden Ihnen dies sehr übel nehmen. So bietet es sich an, für solche Anwendungen die in diesem Buch unter Kapitel 5.2 abgedruckten BASIC-Ergänzungen zu benutzen, die auf allen drei Rechnern laufen. Auf einem CPC 664 oder einem CPC 6128 würde dann bei einem Diskettenfehler die Fehlernummer 32 gemeldet, beim CPC 464 die Fehlernummer 18. Damit kann man fertig werden. Ferner erhält man im Klartext die Fehlermeldung und sie erscheint nicht auf dem Bildschirm.

1.7.2 Weitere Unterschiede des CPC 664 und des CPC 6128

Eine sehr sinnvolle Ergänzung des Schneider-BASIC ist die folgende, die eigentlich nicht eine Ergänzung der Diskettenkommandos im eigentlichen Sinne darstellt, sondern lediglich eine Änderung der CALL- und der RSX-Kommandos und deren Parameterlisten.

Sie erinnern sich, daß wir in diesem Buch immer gepredigt haben, daß Sie, beispielsweise um eine Datei zu löschen, den Dateinamen zunächst in einer Stringvariablen ablegen sollen, um dann den Pointer der Variablen übergeben zu können, etwa:

```
A$ = "PROBE.BAS"
IERA,@A$
```

Beim Schneider CPC 664 und dem Schneider CPC 6128 kann man dieses Kommando auch wie folgt eingeben:

```
IERA,"PROBE.BAS"
```

oder

```
A$ = "PROBE.BAS"
IERA,A$
```

Wie gesagt, sicherlich eine sehr sinnvolle Ergänzung des Schneider-BASIC, da die Bedienung nun sehr viel einfacher und einleuchtender wird. Diejenigen unter Ihnen, die Programme schreiben, die auf allen drei Rechnern laufen sollen, können von diesem Luxus jedoch keinen Gebrauch machen. Der CPC 464 würde bei den beiden letzten Varianten ausschlagen.

Diese Verkürzung des RSX-Kommandos IDEL gilt natürlich ebenfalls für alle anderen Disketten-RSX-Kommandos:

```
IERA, "Dateiname"
IREN, "Dateiname1", "Dateiname2"
IDIR, "Stringausdruck"
IDRIVE, "Laufwerk"
```

Damit hätten wir eigentlich die Unterschiede der drei CPC-Rechner erläutert, die die Diskettenprogrammierung betreffen.

Kapitel 2:
Disketten-Programmierung für Fortgeschrittene

2.1 Die DOS-Vektoren

Nachdem Ihnen in den vorigen Kapiteln ausführlich die Möglichkeiten des AMSDOS so wie auch des CP/M für den 'normalen' Gebrauch erläutert wurden, werden Sie in den folgenden Kapiteln etwas über die Arbeitsweise des DOS und die Nutzung des DOS in Maschinensprache erfahren. Die folgenden Kapitel richten sich darum in der Hauptsache an die Leser, die bereits über eine gewisse Grundkenntnis der Maschinenprogrammierung des Z80 verfügen. Sollte Ihnen dieses Wissen fehlen, so werden Ihnen auf den folgenden Seiten einige Dinge unverständlich bleiben. Legen Sie aber jetzt nicht gleich das Buch aus der Hand. Auch wenn einige Dinge nicht sofort klar werden, viele Informationen auf den nächsten Seiten werden Sie trotzdem verwenden können.

Grundsätzlich kann man bei der Programmierung der Floppystation des CPC drei Ebenen der Programmierung unterscheiden. Die oberste Ebene ist die einfachste. Auf diesem Level werden alle wesentlichen Dinge vom DOS erledigt. Im Grunde handelt es sich um die maschinensprachlichen Äquivalente zu den BASIC-Befehlen, die Sie ja bereits aus dem vorigen Kapitel kennen.

Die zweite Ebene der Diskettenprogrammierung wäre ohne genaue Kenntnis des AMSDOS-ROMs undenkbar. Auf diesem Level stehen teilweise recht einfache, aber auch sehr komplexe Routinen zur Verfügung, die vom einfachen Einschalten des Floppy-Motors bis zum Lesen und Schreiben der Daten auf die Diskette gehen. Wir werden die wesentlichen Routinen des DOS beschreiben, da sich hiermit einige sehr interessante Dinge realisieren lassen, die vom BASIC aus nicht möglich sind.

Die dritte mögliche Art der Programmierung des Floppy-Laufwerkes stellt quasi die Programmierung 'zu Fuß' dar. Auf diesem Level werden sich sicher nur wenige Spezialisten tummeln, da die Funktionen des Disk-Controllers direkt programmiert wer-

den müssen. Da aber die wesentlichen Funktionen bereits in irgendeiner Form im DOS enthalten sind, beschränkt sich diese Programmierung auf ganz wenige Spezialfälle. Auch auf diese Programmierung werden wir eingehen und die Möglichkeiten des Floppy-Controllers ausgiebig erläutern.

Doch beginnen wir zunächst mit den einfacheren Dingen. Die Betriebssysteme aller drei CPC-Rechner (464, 664 und 6128) sind ursprünglich nicht für den Betrieb einer Floppy ausgelegt. Als einziges externes Speichermedium kennen sie den Cassettenrecorder. Erst durch das AMSDOS-ROM (oder ein anderes externes Floppy-ROM) wird der Betrieb einer Floppystation möglich. Steht dieses Erweiterungs-ROM beim Einschalten des Gerätes nicht zur Verfügung, so wirken die Befehle LOAD, SAVE, OPENIN, OPENOUT, CAT und die Befehle zum Filehandling auf das Cassetten-Laufwerk. Steht aber das AMSDOS-ROM dagegen bei der Initialisierung des CPC zur Verfügung, dann ändert sich dies gründlich. Fast alle zuvor für den Recorder zuständigen Befehle wirken jetzt auf die Floppy. Einzige Ausnahme ist der Befehl SPEED WRITE, der ja bekanntlich bei der Floppy keine Wirkung hat.

Wesentlich für diese Flexibilität ist das von den Entwicklern des CPC benutzte Prinzip der Vektoren. Anstatt die gewünschte Aktion sprich Routine des Betriebssystems direkt aufzurufen wird die Routine über eine Adresse im RAM angesprochen, an der dann ein Sprung zur jeweiligen Betriebsystemroutine steht. Daß die Routine über den im RAM liegenden Vektor angesprochen wird, mag auf den ersten Blick als unnütz erscheinen. Tatsächlich aber wird dies Konzept die Kompatibilität mit verschiedenen BetriebssystemVersionen gewahrt. Nur durch diesen Umweg ist es möglich, daß Maschinen-Programme sowohl auf dem 464 wie auch auf dem 664 oder 6128 laufen, obwohl die Rechner teilweise stark unterschiedliche Betriebssysteme besitzen. Solange die Vektoren an den selben Stellen im Ram stehen, und die zugehörigen Routinen von der Funktion her identisch sind, kann man fast beliebige Änderungen im Betriebssystem vornehmen. Die Kompatibilität bleibt gewahrt.

Ein weiterer, sehr wesentlicher Punkt ist die Tatsache, daß man auch als Programmierer alle Fäden in der Hand hat. Zwar sind die ROM-Routinen nicht ohne weiteres zu modifizieren, da aber die Vektoren im Ram liegen, können diese ohne Probleme auf eigene Routinen 'umgebogen' werden. Somit können alle über die Vektoren angesprochenen Funktionen bei Bedarf geändert werden.

Genau dies geschieht im Fall, wenn die Floppy angeschlossen ist. Im CPC sind 22 Vektoren für den Betrieb des Recorders vorgesehen. Von diesen werden 13 Vektoren beim Betrieb der Floppy 'gepatched', also die Adressen der eigentlich aufzurufenden Routinen geändert. Über diese 13 Vektoren werden alle Funktionen der Floppy von BASIC aus angesprochen. Aber auch in Maschinensprache lassen sich diese Vektoren ausgezeichnet benutzen. Nachfolgend finden Sie eine Aufstellung der geänderten Vektoren. Die verwendeten Namen entsprechen denen des CPC-464-Firmware-Manuals.

&BC77 CAS IN OPEN	eröffnet ein File zum Lesen
&BC7A CAS IN CLOSE	schließt zum Lesen geöffnetes File ordnungsgemäß
&BC7D CAS IN ABANDON	schließt zum Lesen geöffnetes File sofort
&BC80 CAS IN CHAR	holt ein Zeichen vom Input-File
&BC83 CAS IN DIREKT	liest ein Input-File komplett in den Speicher
&BC86 CAS RETURN	das letzte Zeichen wird zurückgeschrieben
&BC89 CAS TEST EOF	prüft, ob das Ende des Files erreicht ist
&BC8C CAS OUT OPEN	eröffnet ein File zum Schreiben
&BC8F CAS OUT CLOSE	schließt ein Output-File ordnungsgemäß
&BC92 CAS OUT ABANDON	schließt ein Output-File sofort
&BC95 CAS OUT CHAR	schreibt ein Zeichen ins Output-File

&BC98	CAS OUT DIREKT	schreibt Speicherbereich komplett in ein File
&BC9B	CAS CATALOG	Funktion wie der BASIC-Befehl CAT

So, nachdem Sie diese Tabelle gelesen haben, sollten Sie die verwendeten Namen ganz schnell wieder vergessen. Es erscheint uns nämlich recht unlogisch, in Verbindung mit der Floppy die zugehörigen Routinen mit CAS zu bezeichnen. Wir werden im folgenden die Zeichen CAS durch DISK ersetzen. Damit wird der Name der Funktion gerecht.

2.1.1 DISK IN OPEN &BC77

Bevor Daten aus einer Datei gelesen werden können, muß die Datei eröffnet werden. Das Eröffnen einer Datei wird von der Routine DISK IN OPEN vorgenommen. Dabei ist es unerheblich, ob es sich bei der zu lesenden Datei um eine ASCII-Datei oder um ein Programm handelt. Diese Routine muß in jedem Fall einmal aufgerufen werden.

Um die Datei zu lesen, werden natürlich einige Parameter erwartet. Der wichtigste Parameter ist der Filename der Datei. Bei allen DISK-Routinen werden Parameter in den Registern übergeben. Da ein Filename aber bekanntlich aus 8 Zeichen plus drei Zeichen für die sogenannte Extension besteht, kann der ganze Name nicht in Registern übergeben werden. Statt dessen wird das HL-Registerpaar mit der Adresse versorgt, an der der Filename im Speicher steht. Dabei kann der Filename überall im RAM stehen, auch unter dem ROM, also in den ersten 16 KByte des Speichers des CPC.

Ein weiterer Parameter, der im B-Register an die Routine übergeben werden muß, ist die Länge des Filenamens. Dies ist eine sehr angenehme Eigenschaft des DISK IN OPEN, da der Filename nicht schon von Ihnen auf eine feste Länge gebracht werden muß. Sie können dadurch auch Filenamen mit z.B. 4 Zeichen Länge und einem 2 Zeichen langen Extension eingeben. Die Routine wird selbsttätig den Namen auf die geforderte

Länge erweitern. Bedenken Sie aber immer, auch die Zeichen des Extension mit anzugeben.

Ein dritter Parameter wird von der DISK-IN-OPEN-Routine im DE-Registerpaar erwartet. Hier müssen Sie die Adresse eines 2048 Byte großen Buffers angeben. In diesen Buffer werden die zu lesenden Daten geschrieben. Auch hier ist man bezüglich der Lage des Buffers im Speicher keinen Einschränkungen unterworfen.

Nachdem man die drei Parameter den Registern übergeben hat, kann die Routine aufgerufen werden. Betrachten wir einen solchen Aufruf einmal an einem Beispiel. Wir wollen die Datei mit dem Namen 'TEST.DAT' zum Lesen eröffnen. In Assembler programmiert sieht die Folge der Anweisungen folgendermaßen aus:

```
100  LD       HL,FILNAM          ;Adresse des Namens
110  LD       B,BUFF-FILNAM      ;Länge des Namens
120  LD       DE,BUFF            ;Adresse des Buffers
130  CALL     DISK-IN-OPEN
140  RET
150  FILNAM:  DEFM  'test.dat'   ;Name der Datei
160  BUFF:    DEFS  &0800        ;Platz für 2K
```

Was aber macht die Routine genau, und welche Ergebnisse liefert sie?

Zunächst einmal wird nachgesehen, ob der angegebene Filename gewissen formalen Regeln entspricht. Dazu gehört, daß einige Zeichen im Filenamen verboten sind. Wird eins dieser Zeichen gefunden, so wird das Kommando abgebrochen. Auch darf die Anzahl der Zeichen im Filenamen nicht größer als 15 Zeichen sein. Diese Anzahl setzt sich zusammen aus Usernummer, Drivenummer mit Doppelpunkt, acht Zeichen Filename, Trennpunkt und drei Zeichen Extension. Ein solcher Filename könnte z.B. "0A:FILENAME.DAT" lauten.

Gleichzeitig mit der Prüfung des Filenamens werden alle Klein-Buchstaben in Großbuchstaben gewandelt. Filenamen, die kürzer sind, werden auf die benötigte Länge mit Leerzeichen aufgefüllt.

Nach dieser Prüfung wird nun auf der Diskette nachgesehen, ob eine Datei mit diesem Namen überhaupt existiert. Ist eine Datei mit diesem Namen auf der Diskette nicht vorhanden, z.B. weil der Name nicht richtig oder unvollständig eingegeben wurde, dann meldet das AMSDOS 'Filename' not found. Diese Fehlermeldung erhalten Sie auch, wenn Sie ein Programm laden wollen, das sich nicht auf der eingelegten Diskette befindet.

Unabhängig davon, ob nun die Datei auf der Diskette verfügbar ist oder auch nicht, in jedem Fall erhalten Sie nach dem Aufruf einige wichtige Parameter in Registern zurück.

Zunächst einmal kann grundsätzlich am Zustand des Carry- und Zero-Flags abgelesen werden, ob der DISK IN OPEN erfolgreich war. Dazu wird das Carry-Flag gesetzt und das Zero-Flag gelöscht, wenn der OPEN erfolgreich war. Sind dagegen sowohl Carry als auch Zero gelöscht, dann haben Sie versucht, eine zweite Datei zum Lesen zu eröffnen. Sie haben also bereits eine Datei geöffnet gehabt. Die dritte Möglichkeit besteht darin, daß das Zero-Flag gesetzt, das Carry jedoch gelöscht ist. In diesem Fall wurde das angegebene File nicht gefunden.

War der OPEN erfolgreich, dann kann aus dem Wert des Akkus der Dateityp bestimmt werden. Handelt es sich bei der Datei z.B. um ein BASIC-Programm, so enthält der Akku nach der OPEN-Routine den Wert 0. Ein Wert von &16 bedeutet, daß es sich beim eröffneten File um eine ASCII-Datei handelt. Die weiteren möglichen Werte finden Sie unter DISK OUT OPEN in tabellarischer Form wiedergegeben.

Das HL-Registerpaar enthält nach dem Eröffnen der Datei die Adresse des File Headers. Der Begriff File Header ist Ihnen zu diesem Zeitpunkt sicher noch nicht geläufig, darum soll er hier kurz erläutert werden. Eine detaillierte Beschreibung des Aufbaus des File Header finden Sie aber unter DISK OUT OPEN. Der File Header enthält eine Menge Informationen über die zu-

gehörige Datei. So sind in ihm der Name der Datei, der Dateityp, die Länge der Datei sowie die Adresse, von wo die Datei geschrieben wurde, enthalten. Fast alle Dateien, die auf der Diskette gespeichert werden, also auch und gerade BASIC- und Maschinenprogramme, werden zusammen mit dieser Information abgespeichert. Einzige Ausnahme sind die reinen ASCII-Dateien, bei denen der File Header nicht mit abgespeichert, sondern vom AMSDOS erzeugt wird.

Eine weitere Information erhalten Sie im DE-Registerpaar. Hier wird Ihnen die Startadresse der Datei mitgeteilt. Diese Information ist natürlich nur dann sinnvoll, wenn es sich bei der Datei um ein Programm handelt. Wenn es sich bei der eröffneten Datei um ein BASIC-Programm handelt, dann ist der Wert in DE normalerweise &0170. Bei Maschinenprogrammen enthält DE die Adresse, ab der das Programm geschrieben wurde.

Als letzte Information wird im BC-Registerpaar mitgeteilt, wie lang die Datei ist. Auch hier ist wieder zu bemerken, daß diese Information bei reinen ASCII-Dateien nicht zutrifft. ASCII-Dateien können wesentlich größer als die mit einem Registerpaar darstellbaren 64 K sein. Diese Information ist auch nur wieder für das Laden von Programmen sinnvoll.

2.1.2 DISK IN CLOSE &BC7A

Nach dem Aufruf dieser Routine ist ein zum Lesen eröffnetes File geschlossen. Danach kann nicht weiter aus dem File gelesen werden. In der Hauptsache wird DISK IN CLOSE benötigt, wenn Daten aus einer zweiten Datei gelesen werden sollen. Dann wird zunächst die bereits eröffnete Datei mit DISK IN CLOSE geschlossen, bevor die neue Datei mit DISK IN OPEN zum Lesen geöffnet wird.

Wer es genau wissen will, der kann nach dem Aufruf der Routine an Hand des Carry-Flag feststellen, ob eine Datei zum Lesen eröffnet war. Ist das Carry gesetzt, so war ein File offen.

Diese Routine benötigt keine weiteren Parameter, da ja z.Z. immer nur ein Input-File offen sein kann.

2.1.3 DISK IN ABANDON &BC7D

Diese Routine erfüllt fast dieselbe Aufgabe wie DISK IN CLOSE. Auch nach DISK IN ABANDON ist ein offenes Input-File geschlossen. In der Hauptsache ist diese Routine gedacht, um bei einem eventuellen Fehler die Datei zu schließen. Auch wird DISK IN ABANDON vor jedem LOAD, SAVE und CAT vom BASIC-Interpreter angesprungen. Das bedeutet, daß offene Dateien nach diesen Befehlen grundsätzlich immer sind.

2.1.4 DISK IN CHAR &BC80

Grundsätzlich kennt das AMSDOS zwei verschiedene Methoden, um Daten aus einer Datei zu lesen. Die erste Methode besteht im Aufruf der Routine DISK IN CHAR, die ein Zeichen (Character) aus der Datei holt. Auch diese Routine benötigt keinerlei Parameter, da nur ein Input-File offen sein kann.

Das aus der Datei gelesene Zeichen wird im Akku übergeben. Zusätzlich wird mit Carry- und Zero-Flag signalisiert, ob die Routine ordnungsgemäß ausgeführt wurde. Wurde ein Zeichen aus der Datei geholt, dann ist das Carry gesetzt, das Zero-Flag ist dagegen gelöscht. Sollten Sie jedoch versuchen, ein Zeichen aus einer nicht eröffneten Datei zu lesen, so erhalten Sie als Fehlerkennzeichen die Zustände Carry gelöscht, Zero gelöscht. Zusätzlich steht im Akku die Fehlermeldung &0e. Dieses Zeichen bedeutet, daß das Input-File nicht, wie erwartet, geöffnet ist.

Auch gibt es den Fall, daß Sie am Ende der Datei ankommen. In diesem Fall sind die Flags wie zuvor beide gelöscht, im Akku bekommen Sie jedoch den Wert &1a zurück, das Kennzeichen für END OF FILE.

Die möglichen Fehlermeldungen werden Ihnen am Ende dieses Kapitels vorgestellt.

Das Lesen von Daten mit DISK IN CHAR kann nur in sequentieller Form geschehen. Haben Sie z.B. in einer Datei das 100ste Zeichen gelesen, wollen aber noch einmal auf das zehnte Zeichen zugreifen, so müssen Sie die Datei schließen und danach neu eröffnen.

2.1.5 DISK IN DIREKT &BC83

Wie bereits erwähnt, bietet das AMSDOS zwei Möglichkeiten zum Lesen von Dateien. Die erste Möglichkeit mit der Routine DISK IN CHAR haben Sie bereits kennengelernt. DISK IN DIREKT stellt die zweite mögliche Form des Zugriffs auf die Daten eines Files dar. Im Gegensatz zu DISK IN CHAR kann mit dieser Routine die gesamte Datei in den Speicher gelesen werden. Hauptanwendung dieser Routine ist natürlich das Laden von zuvor gespeicherten Speicherbereichen. Über diese Routine werden BASIC- und Maschinenprogramme geladen.

Im Gegensatz zu DISK IN CHAR muß DISK IN DIREKT mit einem Parameter versorgt werden. Es ist dies die Ladeadresse des Datenblocks, den Sie im HL-Registerpaar übergeben müssen.

```
100         LD    HL,BUFFER        ;Startadresse für Daten
110         CALL  DISK-IN-DIREKT
120         RET
130 BUFFER: EQU   $                ;Hierhin werden die Daten gelesen
```

Nach dem Aufruf von DISK IN DIREKT geben die Flags in schon gewohnter Weise Aufschluß über den Erfolg oder Nichterfolg der Routine. Wie auch bisher bedeutet Carry gesetzt/Zero nicht gesetzt, daß die Daten korrekt gelesen wurden. Die möglichen Fehlermeldungen entsprechen denen von DISK IN CHAR.

Zusätzlich enthält nach dem Aufruf dieser Routine das HL-Registerpaar die sogenannte ENTRY-Adresse aus dem File Header. Die Bedeutung dieser Adresse entnehmen Sie bitte der Beschreibung unter DISK OUT OPEN.

Zu den beiden Möglichkeiten des Zugriffs auf die Daten eines Files gibt es noch einige Besonderheiten anzumerken. Zunächst einmal ist einem in der Benutzung der beiden Routinen freie Hand gegeben. Anders als in BASIC kann also auch ein Programm mit DISK IN CHAR gelesen werden. Wenn jedoch ein Zeichen aus dem Input-File mit DISK IN CHAR gelesen wurde, so kann der Rest nicht mehr über die DIREKT-Routine in den Speicher geholt werden. Umgekehrt ist es nicht möglich, ein mit der DIREKT-Routine gelesenes File anschließend mit der CHAR-Routine weiter zu lesen, abgesehen davon, daß man nach DIREKT sowieso das Ende der Datei erreicht hat. Auch sollte ein einmal mit DISK IN DIREKT gelesenes File nicht ein zweites Mal gelesen werden. Andernfalls können die Daten im Speicher zerstört werden.

2.1.6 DISK RETURN &BC86

Haben wir bei DISK IN CHAR gesagt, daß eine Datei nur sequentiell bearbeitet werden kann, so ist diese Aussage zwar grundsätzlich richtig. Mit DISK IN CHAR jedoch besteht die Möglichkeit, Zeichen öfter als nur einmal zu lesen. Mit DISK RETURN wird das zuletzt gelesene Zeichen wieder in den Buffer zurückgeschrieben und steht für ein weiteres Mal zum Lesen zur Verfügung. Werden die ersten 20 Zeichen aus der Datei gelesen, so wird nach zwanzigmaligem Aufruf von DISK RETURN beim nächsten DISK IN CHAR wieder das erste Zeichen gelesen. Allerdings hat diese Möglichkeit ihre Grenzen. Ursprünglich ist das Zurückschreiben nämlich nur für ein Zeichen gedacht.

Ein Beispiel macht deutlich, wo die Grenzen dieser Routine liegen. In diesem Beispiel denken wir uns eine Datei mit 4000 Zeichen. Haben wir mit DISK IN CHAR 2048 Zeichen gelesen, so ist der Buffer, den wir beim DISK IN OPEN angegeben

haben, 'leer', wir haben alle Zeichen gelesen. Beim Zugriff auf das nächste, das 2049ste Zeichen wird der Buffer mit neuen Daten von der Diskette gefüllt. Haben wir jetzt dieses 2049ste Zeichen gelesen und rufen die DISK IN RETURN-Routine öfter als einmal auf, dann müßten eigentlich die ersten 2048 Datenbytes erneut in den Buffer geladen werden. Dies geschieht aber nicht. Statt dessen verläßt der Zeiger auf das nächste zu lesende Zeichen den Buffer-Bereich, so daß die danach gelesenen Zeichen nichts mehr mit dem Inhalt der Datei zu tun haben.

Aus diesen Fakten wird klar, daß DISK RETURN nur sehr begrenzt eingesetzt werden darf. Der Haupteinsatz dieser Routine liegt im ganz gezielten Prüfen eines einzelnen Zeichens. So wird DISK RETURN auch zum Prüfen des EOF-Kriteriums &1a verwendet.

2.1.7 DISK TEST EOF &BC89

Diese Routine ermöglicht es Ihnen, festzustellen, ob Sie das Ende der Datei erreicht haben. Dazu wird das nächste Zeichen im File mit DISK IN CHAR gelesen und auf den Wert &1a entsprechend 26 dezimal geprüft. Ist der Wert des gelesenen Zeichens nicht 26, so wird das Zeichen anschließend mit DISK RETURN in den Buffer zurückgeschrieben und die Routine kommt mit gesetztem Carry-Flag und gelöschtem Zero-Flag zurück. Wurde dagegen der Wert &1a gefunden, so sind Carry und Zero-Flag gelöscht.

Diese Routine lässt sich ausschließlich beim zeichenweisen Lesen einer Datei verwenden, da die Routine selbst die Routine DISK IN CHAR verwendet. Die Benutzung dieser Routine jedoch ist ja bei DISK IN DIREKT untersagt und führt zu Fehlermeldungen.

2.1.8 DISK OUT OPEN &BC8C

Alle zuvor beschriebenen Routinen gehen davon aus, daß bereits Daten auf der Diskette sind. Die jetzt folgenden Beschreibungen von Routinen zeigen Ihnen, wie Sie in Maschinensprache Daten und Programme auf Diskette speichern können. Analog zum Lesen von Daten muß auch vor dem Schreiben die gewünschte Datei eröffnet werden. Das geschieht durch den Aufruf der Routine DISK OUT OPEN. Ähnlich wie beim Lesen müssen verschiedene Parameter an die Routine übergeben werden.

Da ist zunächst einmal der Filename, unter dem die Daten später auf der Diskette wiederzufinden sind. Die Adresse des gewünschten Namens wird in bekannter Weise im HL-Registerpaar übergeben. Auch die Länge des Namens und die Adresse eines 2048 Byte großen Buffers muß, wie beim DISK IN OPEN, im B-Register und im DE-Registerpaar übermittelt werden.

```
100             LD    HL,FILENAME          ;Adresse des Namens
110             LD    B,BUFFER-FILENAME    ;Länge des Namens
120             LD    DE,BUFFER            ;Datenbuffer
130             CALL  DISK-OUT-OPEN
140             RET
150 FILENAME:   DEFM  'test.dat'           ;Filenname
160 BUFFER:     DEFS  &0800                ;Platz für 2048 Zeichen
```

Zunächst wird der Filename auf seine Gültigkeit hin untersucht und evtl. auf die korrekte Anzahl von Zeichen erweitert. Danach wird die Extension mit drei Dollar-Zeichen überschrieben und dieser Name wird dann auf der Diskette gesucht. Unter diesem Namen wird nämlich zunächst ein temporäres File angelegt, bei dem erst später die originale Extension eingetragen wird. Des weiteren wird ein File Header angelegt, dessen Adresse im HL-Registerpaar nach Abschluß der Routine dem Anwender zur Verfügung steht. Das setzt aber voraus, daß keine Fehler im DISK OUT OPEN auftreten. Sie können dies wieder an Hand des Zustands des Carry-Flags prüfen. Ist im Abschluß an diese Routine das Carry gesetzt, dann ist alles OK, das File ist zum Schreiben eröffnet. Ist das Carry dagegen gelöscht, so ist irgend ein Fehler aufgetreten. Mögliche Fehler sind z.B Fehleingaben

beim Filenamen. Auch Lesefehler beim Prüfen des Directory werden als Fehler gemeldet. In diesem Fall ist dann der Inhalt von HL unbestimmt.

Doch betrachten wir jetzt den Fall, wenn das Output-File korrekt geöffnet wurde. Dann erhalten wir ja die Adresse des File Header im HL-Registerpaar zurück. Diesen File Header wollen wir etwas genauer untersuchen.

Zunächst einmal ist der File Header nichts anderes als ein 64 Byte großer Speicherbereich. In diesen 64 Bytes werden die wichtigsten Daten über die Datei gespeichert. Die Bedeutung der einzelnen Bytes im Header sind beim Lesen und Schreiben von Daten identisch. Die Struktur des Header ist zudem mit dem Aufbau des Header beim Cassettenbetrieb identisch.

Die ersten 16 Bytes enthalten den Filenamen. Nun kann ein Filename bei Cassettenbetrieb schon max. 16 Zeichen lang sein, beim Betrieb der Diskette besteht der Name jedoch aus nur acht Zeichen. Rechnet man noch die Extension mit max. drei Zeichen dazu, so haben wir immer noch erst 11 Zeichen. Zusätzlich wird als erstes Zeichen die User-Nummer eingetragen. Das ergibt eine Anzahl von 12 Bytes, die letzten vier Zeichen des Filenamens im File Header sind immer 0.

Die Bytes 16 und 17 sind in Verbindung mit der Floppy ohne Bedeutung. Sie enthalten beim Betrieb der Cassetten die momentane Blocknummer und ein Kennzeichen, ob der momentane Block der letzte Block der Datei ist. Durch den gänzlich anderen Aufbau jedoch entfallen diese Angaben unter AMSDOS.

Das folgende Byte Nummer 18 jedoch ist auch unter AMSDOS recht wichtig. Es enthält den Filetyp der Datei. Der Filetyp ist in binärer Form codiert, d.h. die einzelnen Bits in diesem Byte haben bestimmte Bedeutungen.

Das Bit 0 sagt aus, ob die Datei geschützt ist. Bei Programmen, die mit SAVE "filename",P abgespeichert worden sind, ist das Bit 0 im Filetyp gesetzt.

Die Bits 1, 2 und 3 bestimmen den eigentlichen Filetyp. Sind alle drei Bits gelöscht, so handelt es sich um ein BASIC-Programm. Ist das Bit 1 gesetzt, ist die Datei eine Binärdatei, also ein Speicherbereich des CPC. Der Filetyp ASCII-Datei wird durch Setzen der Bits 1 und 2 erzeugt.

Die Bits 4 bis 7 sind üblicherweise nicht gesetzt und stellen, so das Firmware Manual zum CPC, die Versionsnummer dar, was immer auch damit gemeint sein mag. Nur ASCII-Dateien haben die Versionsnummer 1, das Bit 4 im Filetyp ist gesetzt.

Nach dem erfolgreichen DISK OUT OPEN ist der Filetyp auf ASCII-Datei eingestellt. Es ist Ihre Aufgabe, hier den richtigen und von Ihnen benötigten Wert einzutragen.

Die Bytes 19 und 20 sind wieder ein Überbleibsel aus dem Cassettenbetrieb. Sie enthalten die Anzahl der Bytes im derzeitigen Datenblock, sind jedoch im Disk-Betrieb ohne Bedeutung.

Wichtiger sind die beiden folgenden Bytes Nummer 21 und 22. Sie enthalten die Adresse, ab der die Daten geschrieben wurden. Diese Angabe ist natürlich beim DISK OUT OPEN noch nicht versorgt. Erst beim Schreiben einer Datei mit DISK OUT DIREKT (siehe unten) wird dieses Feld im File Header versorgt. Bei ASCII-Dateien finden Sie hier nur den Wert 0, da der Angabe in diesem Fall keine Bedeutung zukommt.

Das Byte 23 im File Header wird immer &FF sein. Im Cassettenbetrieb sagt dieser Wert, daß der derzeit gelesene Datenblock der erste der Datei ist. Auch diese Angabe hat beim Diskettenbetrieb keine Aussage.

Die Bytes 24 und 25 enthalten die Länge der Datei. Dieser Wert wird nach dem DISK IN OPEN im DE-Registerpaar übergeben. Bei ASCII-Dateien ist dieser Wert immer 0, da ASCII-Dateien in ihrer Größe nur durch die Diskettenkapazität beschränkt sind.

Die letzten benutzten Bytes 26 und 27 enthalten bei Maschinenprogrammen die Startadresse des Programms. Wenn Sie einen Speicherbereich mit dem Befehl

```
SAVE "speicher.bin",b,1000,2000,1200
```

abspeichern, dann wird der Wert 1200 in hexadezimaler Form in diesen beiden Speicherzellen abgelegt. Arbeiten Sie jedoch mit den entsprechenden Routinen in Maschinensprache, so müssen Sie diesen Wert selbst eintragen.

Alle übrigen Bytes des Fileheader werden vom AMSDOS nach dem DISK OUT OPEN auf 0 gesetzt, jedoch nicht weiter genutzt. Sie stehen Ihnen danach zur freien Verfügung.

2.1.9 DISK OUT CLOSE &BC8F

Auch ein Output-File muß geschlossen werden. Dabei ist die Notwendigkeit des Schließens einer Datei beim Schreiben noch wesentlich größer als beim Lesen einer Datei. Das liegt daran, das nicht jedes Zeichen direkt auf die Diskette sondern erst in den 2048 Byte großen Buffer geschrieben wird, der beim DISK OUT OPEN angelegt wurde. Läuft dieser Buffer über, wird also die Anzahl der zu speichernden Bytes größer als 2048, so wird dieser Buffer auf die Diskette geschrieben. Beim Schließen einer Datei können sich somit maximal 2047 Zeichen im Buffer befinden. Beim Aufruf der CLOSE-Routine wird dieser im Buffer vorhandene Rest ebenfalls auf die Diskette geschrieben. Erst jetzt ist die Datei auf der Diskette vollständig. Sie verlieren also im ungünstigsten Fall 2047 Bytes, wenn Sie die Diskette bei geöffnetem Output-File wechseln.

Weiter wird der beim DISK OUT OPEN angelegte temporäre Filename (der mit der Extension .$$$) ersetzt gegen den origi-

nalen Filenamen. Ein eventuell unter diesem Namen bereits vorhandenes File wird dann umbenannt, es erhält die Extension .BAK. Wenn Sie einmal eine Datei mit dieser Extension im Inhaltsverzeichnis sehen, dann handelt es sich in der Regel um ein nicht ordnungsgemäß geschlossenes File.

Die CLOSE-Routine benötigt keine besonderen Übergabeparameter, da wie auch beim Lesen nur eine Datei zur Zeit geöffnet sein kann. Im Abschluß an diese Routine können Sie wieder an Hand des Carry-Flag und des Zero-Flag feststellen, ob der CLOSE ordnungsgemäß ausgeführt wurde. In diesem Fall ist dann wieder das Carry gesetzt, das Zero-Flag dagegen ist gelöscht.

2.1.10 DISK OUT ABANDON &BC92

Sollte beim Schreiben einer Datei ein schwerwiegender Fehler auftreten, dann kann die Datei mit dieser Routine geschlossen werden. Ein Beispiel für einen solchen schwerwiegenden Fehler wäre etwa die Meldung, daß die Diskette voll ist. In diesem Fall sollten Sie DISK OUT ABANDON aufrufen, da dann die bisher belegten Blöcke auf der Diskette wieder freigegeben werden. Auch erscheint der Name des Files nicht im Directory. In einem solchen Fall müssen natürlich alle bisher geschriebenen Daten erneut auf eine Diskette mit mehr Speicherplatz geschrieben werden.

2.1.11 DISK OUT CHAR &BC95

Wie auch beim Lesen von Daten kennt das AMSDOS zwei verschiedene Wege, Daten auf die Diskette zu schreiben. Die erste der beiden Methoden führt über DISK OUT CHAR. Mit dieser Routine wird das im Akku befindliche Zeichen in den OPENOUT-Buffer geschrieben. Sollte sich hierbei ein Überlauf ergeben, so werden die bisher in den Buffer geschriebenen Zeichen auf die Diskette gerettet und damit wieder Platz im Buffer geschaffen.

Grundsätzlich kann mit dieser Routine jedes gewünschte Zeichen in eine Datei geschrieben werden. Wird die Datei jedoch später mit DISK IN CHAR gelesen, dann müssen Sie bei der Verwendung des Zeichens &1a entsprechend 26 dezimal besonders vorsichtig sein. Dieses Zeichen könnte sonst eventuell beim späteren Lesen als EOF-Kriterium angesehen werden, auch wenn die Datei an dieser Stelle noch überhaupt nicht zu Ende ist.

Außer dem zu schreibenden Zeichen im Akku müssen dieser Routine keine weiteren Parameter übergeben werden. Wie auch bei den anderen Routinen ist im Abschluß an DISK OUT CHAR das Carry gesetzt, wenn das Zeichen ordnungsgemäß geschrieben wurde. Sind dagegen Carry- und Zero-Flag gelöscht, dann haben Sie es versäumt, die benötigte Output-Datei ordnungsgemäß zu eröffnen.

2.1.12 DISK OUT DIRECT &BC98

Diese Routine stellt die zweite Möglichkeit des AMSDOS dar, Daten in einer Datei auf Diskette zu speichern. Im Gegensatz zum Schreiben einzelner Zeichen wird diese Routine zum Speichern kompletter Speicherbereiche des CPC verwendet. Dazu müssen DISK OUT DIRECT verschiedene Parameter übergeben werden.

Zunächst muß das HL-Registerpaar mit der Startadresse des zu schreibenden Speicherbereichs versehen werden. Ab dieser Adresse werden die Daten in die Datei geschrieben.

Der nächste wichtige Wert ist die gewünschte Länge der Datei. Hierzu laden Sie das DE-Registerpaar mit der Anzahl der zu speichernden Bytes. Damit ist auch schon geklärt, daß die maximale Größe einer solchen Datei 64 K entsprechend 65536 Bytes beträgt. Größere Dateien sind aber auch gar nicht erforderlich. Diese Dateigröße reicht aus, um den kompletten Ram-

Speicher des CPC auf Diskette zu bannen, was im übrigen nicht besonders sinnvoll ist, da man diese Datei nicht mehr vollständig laden kann.

Wenn Sie mögen, dann können Sie im BC-Registerpaar die Entry-Adresse (z.B. die Startadresse von Maschinenprogrammen) übergeben. Der Inhalt dieses Registerpaars wird in die Bytes 26 und 27 des File Header geschrieben.

Einen letzten Parameter können Sie im Akku übergeben. Es ist dies der gewünschte Filetyp. Diese Angabe wird ebenfalls in den Fileheader geschrieben, und zwar in das Byte 18 des Header.
Wie auch beim Lesen von Dateien ist es beim Schreiben wichtig, daß die einmal benutzte Methode zum Schreiben nicht gewechselt wird. Man kann also nicht in einer Datei zwischen den beiden Methoden hin- und herspringen. Haben Sie erst einmal ein Zeichen mit DISK OUT CHAR geschrieben, so führt der Versuch, anschließend DISK OUT DIRECT zu benutzen, zu einer Fehlermeldung.

Auch ist es leider nicht möglich, in einer Datei mehr als einmal DISK OUT DIRECT zu verwenden. Auch wenn nach einem weiteren DISK OUT DIREKT die Meldung des AMSDOS ein OK signalisieren sollte, so gibt es spätestens beim DISK OUT CLOSE die Meldung 'File not Open as Expexted'. Auch wird die Datei nicht ordnungsgemäß geschrieben werden. Nach dem einmaligen Abschluß dieser Routine muß also das Output-File geschlossen werden. Jeder weitere Speicherbereich des CPC muß in einer neuen Datei auf die Diskette geschrieben werden.

2.1.13 DISK CATALOG &BC98

Allein wegen dieser sinnvollen Einrichtung ist die Diskette dem Cassettenbetrieb vorzuziehen. In kürzester Zeit erhalten Sie einen Überblick über die Dateien auf dem derzeitigen Laufwerk. Zusätzlich wird die Anzahl freier Blocks auf der Diskette angezeigt.

Vor dem Aufruf von DISK CATALOG müssen Sie im DE-Registerpaar die Startadresse eines 2048 Bytes großen Bufferbereichs angeben. Obwohl dieser Buffer grundsätzlich nicht nötig wäre und wohl mehr aus Kompatibilitätsgründen zum Cassettenbetrieb eingesetzt wurde, haben sich die Programmierer des AMSDOS mit diesem Buffer doch eine nette Kleinigkeit ausgedacht. Beim Aufruf des DISK CATALOG werden nämlich die Namen der Dateien von der Diskette gelesen und zusammen mit der Information über die Anzahl der belegten Blocks in diesem Buffer abgelegt. Allerdings werden die Filenamen nicht einfach der Reihe nach in den Buffer geschrieben, sondern bei dieser Gelegenheit auch schön in alphabetischer Reihenfolge sortiert. Das ist der Grund, weshalb Sie immer ein so schön sortiertes Directory beim CAT-Befehl erhalten. Beim DIR-Kommando dagegen werden die Files in der Reihenfolge angezeigt, in der sie tatsächlich auf der Diskette stehen, also unsortiert.

2.1.14 Das Patchen der Vektoren

Zu Anfang dieses Kapitels haben wir ausführlich die Vorzüge der im Ram befindlichen Vektoren erläutert. Wenn Sie jedoch auf die Idee kommen sollten, von einem wesentlichen Vorteil, der leichten Veränderbarkeit der entsprechenden Routinen, Gebrauch zu machen, so sollten Sie nicht überstürzt an die Arbeit gehen. Die Sache hat einen gefährlichen Haken.
Ein Blick in das ROM-Listing im Kapitel 3 zeigt, wo die Schwierigkeit liegt. Alle 13 Vektoren zeigen nach dem Initialisieren des AMSDOS den selben Inhalt. In allen Fällen wird der drei Bytes jedes Vektors lauten:

```
RST    &18
DEFW   &A88B
```

Eine einfache PEEK-Schleife bestätigt diese Aussage. Mit dem RST &18, auch RST 3 genannt, besteht die Möglichkeit, jede beliebige Routine in jedem beliebigen ROM oder im RAM als Unterprogramm, also quasi als CALL anzuspringen. Dazu reicht jedoch die 2-Byte-Adresse hinter dem Restart-Befehl nicht aus, da in zwei Bytes nur eine Adresse, nicht aber ein zusätzlicher

ROM-Auswahlwert untergebracht werden kann. Die Lösung des Problems ist nun folgende: Die Adresse hinter dem RST zeigt auf die Adresse im Speicher, an der die endgültige Drei-Byte-Adresse steht.

In unserem Fall müssen wir an der Adresse &A88B nachsehen, um die Adresse und das ROM-Auswahlbyte der tatsächlichen DOS-Routine zu erhalten. An der Adresse &A88B finden sich nun die Werte &30, &CD und &07. Das ergibt in 'normaler' Lesart die Adresse &CD30 im Erweiterungs-Rom 7.

Wenn man jetzt einmal an der angegebenen Adresse im AMSDOS-ROM nachsieht, so findet man eine etwas eigenartige Routine, die über die Manipulation des Stack die Adresse des aufgerufenen Vektors ermittelt (tatsächlich liegt auf dem Stack die Adresse des Vectors +3). Zu dieser Adresse wird dann ein Wert von &10D2 addiert und oben auf den Stack gelegt. Ein anschließender RETURN führt dann zu der eigentlich geforderten Routine. Was passiert da?

Warum man diesen indirekten Weg des Ansprungs der einzelnen Routinen gewählt hat, wird klar, wenn man sich vor Augen hält, daß der RST 3 wie ein CALL wirkt. Nun wird bei einem CALL eine Rücksprungadresse auf den Stack gelegt. Diese Adresse aber zeigt hinter den aufgerufenen Vektor auf den nächsten Eintrag im Jump-Block. Unser Programm jedoch soll ja an der Stelle fortgeführt werden, die dem Aufruf des Vektors folgt. Darum muß unbedingt die Rücksprungadresse des RST vom Stack verschwinden.

Genau diese Aufgabe übernimmt die Routine bei &CD30. Die zu entfernende Rücksprungadresse wird aber nicht einfach weggeworfen, sondern mit dem schon erwähnten Wert &10D2 addiert und ergibt die tatsächliche Adresse der aufzurufenden Routine. Darin sind aber auch schon die Schwierigkeiten beim Patchen der Routinen erklärt. Kommt ein RST auf die Adresse &CD30 von einer anderen Adresse als einem Jump-Block-Eintrag, so ergibt die Addition einen völlig unsinnigen Wert, Absturz oder Reset sind die warscheinlichen Folgen.

Doch genug der Erklärung, schauen wir uns einmal an, wie man die Routinen trotzdem patchen kann.

Das gewählte Beispiel ist recht simpel. Die einzige Funktion des Patches ist es, zu zeigen, wie ein solcher Patch arbeitet. Für dies Beispiel werden wir den CAT-Vektor verbiegen.

```
100  INIT:   LD   HL,&BC9C      ;Adresse des CAT-Vektors
110          LD   TEMP,HL       ;speichern
120          LD   HL,PATCH      ;neue Adresse des RST
130          LD   &BC9C,HL      ;patchen
140          RET                ;alles klar
150  PATCH:  EQU  $             ;hier könnte Ihre Routine stehen
160          LD   HL,TEMP       ;ursprüngliche Adresse
170          LD   &BC9C,HL      ;wieder eintragen
180          CALL &BC9B         ;Cat-Vektor mit korrekter Adresse
190  PATCH1: EQU  $             ;auch hier könnte Ihre Routine sein
200          CALL INIT          ;für nächsten Aufruf wieder setzen
210          RET                ;alles vorbei
```

Wie Sie sehen, tun wir tatsächlich nichts, außer dem Verbiegen des Vektors. Allerdings haben Sie auf diese Weise sowohl vor wie auch nach der Routine die Möglichkeit des Eingriffs.
Ein praktisches Beispiel der angewendeten Methode ist die Reparatur des MERGE und CHAIN MERGE. Vielleicht disassemblieren Sie einmal das kleine Programm, um zu ergründen, wie der Eingriff in die DISK IN CHAR programmiert ist.

2.2 Die Befehlserweiterungen des AMSDOS

Mit den im vorigen Kapitel beschriebenen Routinen sind die Möglichkeiten des Programmierers natürlich noch lange nicht erschöpft. Auch die Befehle der Befehlserweiterung, in BASIC durch den senkrechten Balken | gekennzeichnet, lassen sich ohne Schwierigkeiten in Maschinensprache nutzen. Wie dies im einzelnen geschieht, wollen wir jetzt betrachten.

Die möglichen Kommandos kennen Sie ja bereits aus früheren Kapiteln und auch aus Ihrem Handbuch zur Floppy. Hier noch einmal eine Übersicht über die verfügbaren Kommandos:

CPM	**A**
DISC	**B**
DISC.IN	**DRIVE**
DISC.OUT	**USER**
TAPE	**DIR**
TAPE.IN	**ERA**
TAPE.OUT	**REN**

Im Gegensatz zu den zuvor beschriebenen DISK-Routinen gibt es zu den Kommandos der Befehlserweiterungen keine Vektoren, so daß sie nicht direkt angesprungen werden können. Für die Befehlserweiterungen sind spezielle Maßnahmen erforderlich, um die gewünschte Aktion 'anzuleiern'. Machen wir uns also mit dem erforderlichen Mechanismus vertraut.

2.2.1 Programmierung der Erweiterungen in Assembler

Sollen Kommandos einer beliebigen Befehlserweiterung ausgeführt werden, so muß zunächst die Adresse der Routine bekannt sein. Da diese Routinen sich sowohl im RAM als sogenannte RSX-Erweiterung als auch in ROMs in den oberen 16 K des CPC befinden können, muß weiterhin die eventuelle ROM-Adresse ermittelt werden. Diese Angaben findet eine spezielle Routine im Kernal des CPC, wenn wir den Namen der gewünschten Befehlserweiterung angeben. Dazu muß die Adresse des Namens im HL-Registerpaar eingetragen werden. Auch sollten Sie den Namen nur mit Großbuchstaben angeben. Das letzte Zeichen des Namens wird dadurch gekennzeichet, indem Sie das Bit 7 dieses Zeichens setzen. In genau dieser Form stehen alle Namen der Befehlserweiterungen in ROM und RAM.

Haben wir also den Namen in der gewünschten Form im RAM stehen und ist HL mit der Adresse dieses Namens versehen, so genügt der Aufruf der Routine KL FIND COMMAND.

KL FIND COMMAND ist eine Routine des Kernal des CPC und hat einen Vektor im RAM an der Adresse BCD4. Nachdem diese Routine ausgeführt wurde, erhalten wir folgende Parameter zurück:

Ist das Carry-Flag nicht gesetzt, so wurde das entsprechende Kommando nicht gefunden. Entweder haben Sie den Namen der Befehlserweiterung nicht richtig eingegeben, oder die Erweiterung ist noch nicht initialisiert.

Bei der Floppy kann letzterer Umstand eintreten, wenn Sie zuerst den CPC und danach die Floppy einschalten. Durch gleichzeitiges Drücken der drei Tasten CTRL, SHIFT und ESC wird jedoch ein Reset durchgeführt, bei dem die Befehlserweiterung 'eingeklinkt' wird und danach zur Verfügung steht.

Ist dagegen das Carry-Flag nach der Routine KL FIND COMMAND gesetzt, so erhalten Sie die Adresse der gewünschten Routine im HL-Registerpaar zurück. Die erforderliche ROM-Adresse finden Sie im C-Register.

Die Versorgung gerade dieser Register wurde von den Programmierern des CPC-Betriebssystems mit Bedacht gewählt, denn es existiert eine Kernal-Routine mit dem Namen KL FAR PCHL, die jede gewünschte Adresse in jedem möglichen ROM oder im RAM als Unterprogramm aufrufen kann. Die Adresse der Routine muß dazu in HL, die erforderliche ROM-Adresse im C-Register übergeben werden. Genau da stehen nach KL FIND COMMAND die benötigten Werte, so daß dem Aufruf nichts mehr im Wege ist.

Betrachten wir diesen Mechanismus einmal an Hand des Kommandos IDIR, das ja bekanntlich keine weiteren Parameter benötigt, bevor wir uns mit den Kommandos auseinandersetzen, bei denen Parameter in Form von numerischen oder Stringvariablen nötig sind.

```
100         LD    HL,COMMAND          ;Adresse des gewünschten Kommandos
110         CALL  KL-FIND-COMMAND     ;Adresse des Vectors ist &BCD4
120         RET   NC                  ;Kommando wurde nicht gefunden
```

```
130             XOR   A               ;keine Parameter an DIR übergeben
140             CALL  KL-FAR-PCHL     ;Adresse ist &001B
150             RET
160 COMMAND:    DEFM  'DI','R'+&80    ;Name des Kommandos
```

Die beiden benutzten Routinen des Kernal sind in ihrer Leistungsfähigkeit wirklich fantastisch. Kein Mensch muß sich mehr mit langen Adressen-Tabellen aller benötigten Routinen herumquälen. Es ist vollkommen ausreichend, wenn der Name der Routine bekannt ist.

Nur unwesentlich aufwendiger wird die Programmierung, wenn an die Befehlserweiterung irgendwelche Parameter übergeben werden müssen. Nehmen wir zunächst den Fall, daß nur numerische Integer-Werte übergeben werden müssen. Dies ist z.B. bei dem |USER-Befehl der Fall. Als gültige USER-Nummern sind Werte zwischen 0 und 15 zugelassen. Sehen wir uns ein Beispiel-Programm für den Aufruf des User-Kommandos an, bevor wir die Besonderheiten der Parameterübergabe besprechen:

```
100             LD    HL,COMMAND      ;Adresse des gewünschten Kommandos
110             CALL  KL-FIND-COMMAND ;Adresse des Vectors ist &BCD4
120             RET   NC              ;Kommando wurde nicht gefunden
130             LD    A,1             ;einen Parameter an |USER
                                      ;übergeben
140             LD    IX,NUMBER       ;gewünschte User-Nummer
150             CALL  KL-FAR-PCHL     ;Adresse ist &001B
160             RET
170 COMMAND:    DEFM  'USE','R'+&80   ;Name des Kommandos
180 NUMBER:     DEFW  0004
```

Zunächst sehen Sie, daß wir der |USER-Routine im Akku die Anzahl der übermittelten Parameter mitteilen. Jeder andere Wert als eins führt in diesem Fall zur Ausgabe von 'Bad command'.

Das IX-Register zeigt auf die gewünschte Usernummer, in unserem Beispiel die Usernummer 4. Da nur 16-Bit-Werte übergeben werden können, muß in der Zeile 180 der Wert von

NUMBER mit der Anweisung DEFW als 2-Byte-Wert (entspricht 16 Bit) definiert werden.

Die Programmierung ist also in diesem Fall nicht wesentlich aufwendiger. Etwas mehr Arbeit gibt es, wenn Strings an die Befehlserweiterung übergeben werden müssen. Das ist z.B. bei den Befehlen |ERA, |REN und |DRIVE der Fall.

Wie Sie ja bereits wissen, können Strings nicht direkt an Befehle der Befehlserweiterung übergeben werden. Statt dessen wird der Variablenpointer, den Sie mit der Funktion 'Klammeraffe' erhalten, übergeben. Der Variablenpointer selbst ist ein Zeiger auf den sogenannten Stringdescriptor. Der Aufbau des Stringdescriptors und die Verwaltung der Strings in BASIC sollte Ihnen bekannt sein, da eine detaillierte Erläuterung hier aus Platzgründen nicht erfolgen kann.

Nehmen wir zunächst den einfacheren Fall mit nur einem String als Parameter, das |ERA-Kommando. In unserem Beispiel wollen wir ein File mit dem Namen 'ADRESSEN.DAT' löschen.

```
100             LD    HL,COMMAND          ;Adresse des gewünschten Kommandos
110             CALL  KL-FIND-COMMAND     ;Adresse des Vectors ist &BCD4
120             RET   NC                  ;Kommando wurde nicht gefunden
130             LD    A,1                 ;einen Parameter an |ERA übergeben
140             LD    IX,VARPTR           ;Pointer auf Variablendescriptor
150             CALL  KL-FAR-PCHL         ;Adresse ist &001B
160             RET
170 COMMAND:    DEFM  'ER','A'+&80        ;Name des Kommandos
180 VARPTR:     DEFW  DESCRIP             ;Adresse des Descriptors
190 DESCRIP:    DEFB  12                  ;Länge der Variablen
200             DEFW  FILNAM              ;Adresse des Namens
210 FILNAM:     DEFM  'ADRESSEN.DAT'      ;Name des zu löschenden Files
```

Noch etwas aufwendiger wird die Angelegenheit, wenn zwei Strings, wie im Fall des |REN-Kommandos, benötigt werden. Das folgende Beispiel zeigt Ihnen, wie Sie eine Datei 'ADRESSEN.DAT' in 'ADRESSEN.ALT' umbenennen können.

```
100             LD    HL,COMMAND          ;Adresse des gewünschten Kommandos
110             CALL  KL-FIND-COMMAND     ;Adresse des Vectors ist &BCD4
120             RET   NC                  ;Kommando wurde nicht gefunden
130             LD    A,2                 ;zwei Filenamen an REN übergeben
140             LD    IX,VARPTR           ;Pointer auf Variablendescriptoren
150             CALL  KL-FAR-PCHL         ;Adresse ist &001B
160             RET
170 COMMAND:    DEFM  'RE','N'+&80        ;Name des Kommandos
180 VARPTR:     DEFW  DESCOLD             ;Adresse des Descriptors des alten
                                          ;Namens
190             DEFW  DESCNEW             ;Adresse des Descriptors des neuen
                                          ;Namens
200 DESCOLD:    DEFB  12                  ;Länge der Variablen
210             DEFW  OLDNAME             ;Adresse des neuen Filenamens
220 OLDNAME:    DEFM  'ADRESSEN.DAT'      ;alter Filename
230 DESCNEW:    DEFB  12                  ;Länge der Variable
240             DEFW  NEWNAME             ;Adresse des neuen Filenamens
250 NEWNAME:    DEFM  'ADRESSEN.ALT'      ;neuer Filename
```

Wenn wir auch in diesem Beispiel alle Daten schön beieinander stehen haben, so können sie doch in der Praxis überall im Speicher des CPC stehen. Auch die Reihenfolge spielt keine Rolle. Wichtig ist nur, daß die Pointer auf die Descriptoren in der richtigen Reihenfolge und unmittelbar hintereinander im Speicher stehen.

Mit dem Wissen aus diesem Kapitel sollte es Ihnen jetzt eigentlich möglich sein, alle auch von BASIC aus erreichbaren Möglichkeiten auch in Maschinensprache zu nutzen. Das ist sicherlich schon eine ganze Menge, macht aber nur einen Teil der tatsächlich möglichen Dinge aus. Ziel der Programmierung in Maschinensprache wird es ja gerade sein, die Grenzen der in BASIC vorhandenen Möglichkeiten zu erweitern. Das aber haben wir bisher nicht erreicht.

Welche zusätzlichen Möglichkeiten sich dem Programmierer bieten, wollen wir im nächsten Kapitel erforschen.

2.2.2 Die 'versteckten' Befehle der Befehlserweiterung

Beim Durcharbeiten des ROM-Listings stießen wir auf im Handbuch mit keiner Silbe erwähnte Möglichkeiten. Zusätzlich zu den bekannten 14 Befehlen der Befehlserweiterung existieren weitere 9 zum Teil sehr interessante Befehle, die sich wie die bisher beschriebenen Befehle einsetzen lassen. Allerdings können diese Befehle nicht von BASIC aus wie z.B. das IDIR-Kommando eingesetzt werden.

Alle diese Befehle haben einen ein Zeichen langen Namen. Die Namen lauten &01, &02 ... &09. Da bei allen Namen der Befehlserweiterung das 7. Bit des letzten Zeichens gesetzt sein muß, sind die tatsächlichen Namen &81, &82 ... &89. In BASIC-Programmen können Sie diese Namen nicht eingeben. Dadurch ist die Anwendung dieser Befehle eigentlich dem Maschinensprache-Programmierer vorbehalten. Bei einigen Befehlen wäre die Anwendung in BASIC auch gar nicht sinnvoll, wie Sie den Beschreibungen der einzelnen Befehle entnehmen können.

2.2.2.1 Der Befehl &81 Message ON/OFF

Dieser Befehl erlaubt es, Fehlermeldungen auszuschalten, die im Zusammenhang mit den noch folgenden Befehlen &82 bis &89 auftreten können. Im besonderen sind dies die Fehlermeldungen vom Disk-Controller, die vom Anwender die Eingabe von C, I oder R für Chancel, Ignore und Retry erfordern. Um dieses Ausschalten der Fehlermeldungen zu erreichen, muß der Programmierer vor dem Aufruf im Akku einen Wert ungleich Null übergeben.

```
100             LD    HL,COMMAND        ;Adresse des gewünschten Kommandos
110             CALL  KL-FIND-COMMAND   ;Adresse des Vectors ist &BCD4
120             RET   NC                ;Kommando wurde nicht gefunden
130             LD    A,&FF             ;Wert zum Ausschalten der Meldungen
140             CALL  KL-FAR-PCHL       ;Adresse ist &001B
150             RET
160 COMMAND:    DEFB  &81               ;Name des Kommandos
```

Die ganze Aktion dieses Programmes ist es, den Wert im Akku in die Speicherzelle &BE78 zu übertragen. Das können Sie sicherlich schneller durch explizites Beschreiben dieser Speicherzelle erreichen. Wenn sich jedoch aus irgendwelchen Gründen in späteren Versionen des AMSDOS die Adresse dieser Speicherzelle ändern sollte, so wird die zuvor gezeigte Routine sicher im AMSDOS so angepasst sein, daß die geänderte Ram-Adresse versorgt wird. Sie sollten also unbedingt aus Gründen der Kompatibilität mit diesen möglichen späteren Versionen die oben gezeigte Routine verwenden, wenn Sie nicht ausschließlich für sich selber programmieren.

Im BASIC 1.0 ist die Routine nicht einsetzbar, da Sie auf die Fehlermeldungen am Bildschirm angewiesen sind. Die im Akku übergebene Fehlernummer steht Ihnen ja im BASIC 1.0 nicht zur Verfügung. Statt dessen würde nur das laufende Programm unterbrochen und die lakonische Meldung 'BREAK' auf dem Bildschirm erscheinen. In den BASIC-Versionen des CPC 664 und CPC6128 können Sie diese Routine jedoch einsetzen und Fehler mit ON ERROR GOTO und der reservierten Systemvariablen DERR abfragen. Für BASIC-Programme geeignet sich jedoch das Programm zum Abfangen von Fehlermeldungen besser, das Sie später in der Programmsammlung finden.

2.2.2.2 Der Befehl &82 Drive Parameter

Dieser Befehl erlaubt die Veränderung der Laufwerksdaten. Dazu gehören die Wartezeit, bis das Laufwerk nach dem Einschalten des Drivemotors die Nenndrehzahl erreicht hat, ferner die Zeitkonstante, die nach einem Spurwechsel gewartet werden muß, sowie die Nachlaufzeit des Disk-Motors. Auch die Zeiten

für die HEAD LOAD TIME und HEAD UNLOAD TIME werden in dieser Routine neu gesetzt.

In Gegensatz zu allen bisherigen Routinen der Befehlserweiterung kann &82 nicht mit den bekannten Kernal-Routinen aufgerufen werden. &82 erwartet nämlich im HL-Registerpaar den Anfang der Tabelle für die Laufwerkdaten. Somit entfällt der Ansprung mit KL FAR PCHL, da diese Kernal-Routine in HL die Adresse der anzuspringenden Routine benötigt. Aber es gibt durchaus auch andere Wege, einen Befehl einer Erweiterung aufzurufen.

```
100             LD    HL,COMMAND        ;Adresse des gewünschten Kommandos
110             CALL  KL-FIND-COMMAND   ;Adresse des Vectors ist &BCD4
120             RET   NC                ;Kommando wurde nicht gefunden
130             LD    (FARADR),HL       ;Adresse der gew. Routine merken
140             LD    A,C               ;Romselect in den Akku
150             LD    (FARADR+2),A      ;und ebenfalls abspeichern
160             LD    HL,NEWTAB         ;Tabelle neuer Laufwerksparameter
170             RST   &18               ;wirkt wie CALL zur gew. Routine
180             DEFW  FARADR            ;Vektor auf Drei-Byte Far Adress
190             RET
200 COMMAND:    DEFB  &82               ;Name des Kommandos
210 FARADR:     DEFS  3                 ;Platz für die Drei-Byte-Adresse
220 NEWTAB:     DEFS  7                 ;7 Bytes müssen an &82 übergeben
                                        ;werden
230 HDUNLD:     DEFS  1                 ;hier muß die gew. HEAD UNLOAD-TIME
                                        ;stehen
240 HEADLD:     DEFS  1                 ;HEAD LOAD TIME, nach Datenblatt
                                        ;des FDC
```

Anders als bei den bisherigen Aufrufen von Routinen des AMSDOS haben wir hier einen RESTART-Befehl eingesetzt. Der RST &18 stellt eine besondere Form des CALL-Befehls dar. Über diesen RST kann, ähnlich wie mit der Routine KL-FAR-PCHL, jede gewünschte Adresse in jedem möglichen ROM oder RAM angesprungen werden. Der Vorteil des RST &18 jedoch liegt darin, daß keine Register benötigt werden. Wir können mit dieser Routine also auch im HL-Registerpaar und im C-Register Parameter an aufzurufende Unterprogramme übergeben, was bei

der bisher genutzten KERNAL-Routine nicht möglich war. Dafür müssen die drei Bytes der FAR-Adresse im Speicher des CPC abgelegt werden. Diese drei Bytes, also Zwei-Byte-Adresse und ROM-Select-Byte müssen in jedem Fall in der angegebenen Reihenfolge und zusammenhängend abgelegt werden.
Jetzt müssen wir uns nur noch einmal ansehen, wie denn die geforderte Tabelle auszusehen hat.

Der erste Tabelleneintrag ist ein 16-Bitwert, der die Wartezeit definiert, die nach dem Einschalten des Laufwerkmotors gewartet werden muß. Es ist offensichtlich, daß die Diskette nicht unmittelbar nach dem Einschalten des Laufwerkmotors die Nenndrehzahl erreicht. Dies ist erst nach ca. einer Sekunde der Fall. Entsprechend ist der Standard-Wert nach dem Einschalten 50 dezimal, der mit einem Ticker-Event heruntergezählt wird. Da der Ticker alle 1/50 Sekunde ausgeführt wird, ergibt sich eine Wartezeit von genau einer Sekunde.

Der zweite Tabelleneintrag beschreibt die Nachlaufzeit der Motoren nach dem letzten Diskettenzugriff. Auch hier wird wieder mit dem Ticker gearbeitet. Nach der Initialisierung des AMSDOS ist der verwendete 16-Bitwert 250 dezimal. Das entspricht einer Nachlaufzeit von 2,5 Sekunden.

Beim dritten Tabelleneintrag handelt es sich um einen Ein-Byte Wert. In der Standard-Einstellung ist hier ein Wert von &af enthalten. Dieser Wert wird nur bei der Routine &86, Formatieren einer Spur, benötigt und sollte nicht geändert werden.

Auch der folgende 16-Bit Wert definiert Wartezeiten. Das High-Byte dieses Wertes bestimmt die Zeit, die nach einem Spurwechsel gewartet werden muß. Standardmäßig ist eine Wartezeit von 12 ms eingestellt. Diese Wartezeit ist vom verwendeten Laufwerk abhängig.

Zwei weitere Wartezeiten müssen in der Tabelle definiert werden. Es sind dies die Werte für die HEAD LOAD TIME und HEAD UNLOAD TIME, die vom Controller-IC benötigt werden. Die im ROM vorgegebenen Werte sind 32 ms für die HEAD UNLOAD TIME und 16 ms für die HEAD LOAD

TIME. Die genaue Bedeutung dieser Werte finden Sie weiter unten in der Beschreibung des FDC 765.

Der Befehl &82 wird vom AMSDOS nach jedem Einschalten oder Reset automatisch durchgeführt. Auch unter CP/M wird diese Funktion beim Starten des CP/M einmal durchgeführt. Unter CP/M besteht die Möglichkeit, die verschiedenen Parameter im Programm SETUP.COM nach Wunsch zu definieren. Die nach dem Reset verwendete Tabelle finden Sie an der Adresse &C5D4 im AMSDOS-ROM.

2.2.2.3 Der Befehl &83 Disk Format Parameter

Bekanntlich kann der CPC von Haus aus mit drei unterschiedlichen Diskettenformaten arbeiten. Der Befehl &83 erlaubt es, das Format der im aktiven Laufwerk eingelegten Diskette festzulegen. Zu jedem Format befindet sich im ROM des AMSDOS eine Tabelle. Je nach im Akku übergebenem Wert wird die benötigte Tabelle im RAM des CPC angelegt. Damit kann der Anwender in Maschinenprogrammen mit direktem Sektorzugriff die richtigen Tabellen anlegen. In BASIC ist die Verwendung zwar möglich, aber nicht sinnvoll, da sich das AMSDOS diese Werte selbst bestimmt. Auch wenn Sie also ein bestimmtes Format angeben, so wird doch zusätzlich vor jedem Diskettenzugriff noch einmal das Format ermittelt.

```
100         LD   HL,COMMAND       ;Adresse des gewünschten Kommandos
110         CALL KL-FIND-COMMAND  ;Adresse des Vectors ist &BCD4
120         RET  NC               ;Kommando wurde nicht gefunden
130         LD   (FARADR),HL      ;Adresse der gew. Routine merken
140         LD   A,C              ;Romselect in den Akku
150         LD   (FARADR+2),A     ;und ebenfalls abspeichern
160         LD   A,FORMAT         ;Kennzeichen des gew. Formats (siehe
                                  ;Text)
```

```
170             RST  &18            ;wirkt wie ein CALL zur gew. Routine
180             DEFW FARADR         ;Vektor auf die Drei-Byte Far Adress
190             RET
200  COMMAND:   DEFB &83            ;Name des Kommandos
210  FARADR:    DEFS 3              ;Platz für die Drei-Byte-Adresse
```

Das Programm unterscheidet sich in keiner Weise von den bisher gezeigten. Bleibt nur noch zu klären, wie die Kennzeichen der unterschiedlichen Diskettenformate auszusehen haben.

Diese Unterscheidung wird an Hand der Sektornummern vorgenommen. Beim AMSDOS-CP/M-Format gibt es auf jeder Spur einer Diskette 9 Sektoren mit jeweils 512 Bytes. Diese 9 Sektoren tragen die Sektornummern &41 bis &49. Das bedeutet, daß bei jeder Sektornummer das Bit 6 gesetzt ist. Im AMSDOS-Datenformat dagegen lauten die Sekornummern &C1 bis &C9. In diesem Fall sind die Bits 6 und 7 in jeder Sektornummer gesetzt. Im IBM-CP/M-Format lauten die Sektornummern &01 bis &08. In diesem Format sind ja bekanntlich nur 8 Sektoren auf der Diskette formatiert. Damit steht ein eindeutiges Kriterium zur Unterscheidung der drei Formate zur Verfügung. Übergeben Sie an die Routine &83 den Wert 0 (oder jeden anderen Wert mit gelöschtem Bit 6 und 7), dann wird im RAM die Parametertabelle für das IBM-CP/M-Format eingerichtet. Das AMSDOS-CP/M-Format wird eingestellt, wenn ein Wert mit gesetztem Bit 6 und gelöschtem Bit 7 übergeben wird. Das könnte also &40, aber auch &73 sein. Setzen Sie in dem zu übergebenden Wert die beiden obersten Bits (z.B. &C0 oder &FF), so wird das AMSDOS-Datenformat eingestellt.

2.2.2.4 Der Befehl &84 Read Sector

Die Mächtigkeit dieses Befehls ist nicht zu unterschätzen. Er erlaubt es, jeden beliebigen Sektor direkt von der Diskette zu lesen!

Damit sind Sie nicht mehr an die bekannten Dateistrukturen gebunden, sondern können sich bei Bedarf vollständig eigene Strukturen wie z.B. relative oder ISAM-Dateien aufbauen, zumal

der noch folgende Befehl &85 das direkte Schreiben eines jeden Sektors erlaubt. Mit diesen beiden Befehlen stehen Ihnen alle Bytes auf der Diskette in direktem Zugriff zur Verfügung.
Um einen bestimmten Sektor mit &84 lesen zu können, müssen einige Angaben an die Routine gemacht werden.

Das ist zunächst einmal das gewünschte Laufwerk. Diese Angabe muß im E-Register abgelegt werden. Ein Wert von 0 im E-Register selektiert das Laufwerk A, eine 1 wählt das Laufwerk B.

Weiterhin ist natürlich auch der gewünschte Sektor der Routine mitzuteilen. Als Register für diesen Parameter ist das C-Register gewählt worden. Allerdings muß die tatsächlich auf der Diskette vorhandene Sektornummer angegeben werden. Wollen Sie also den ersten Sektor einer im AMSDOS-CP/M-Format geschriebenen Spur lesen, so lautet die korrekte Sektornummer &41.

Zusätzlich wird noch der Track, also die gewünschte Spur auf der Diskette, benötigt. Die Tracknummer wird im D-Register angegeben. Im AMSDOS sind Tracknummern zwischen &00 und &27 (dezimal 39) zulässig.
Damit sind alle wichtigen Parameter in den richtigen Registern übergeben. Aber Halt! Ganz so schnell geht es nicht. Wir müssen uns jetzt noch entscheiden, an welche Speicheradresse die 512 Bytes eines Sektors gelesen werden. Diese Angabe muß im HL-Registerpaar übergeben werden. Damit haben wir alle Parameter beisammen und die Daten können kommen.

```
100         LD   HL,COMMAND        ;Adresse des gewünschten Kommandos
110         CALL KL-FIND-COMMAND   ;Adresse des Vectors ist &BCD4
120         RET  NC                ;Kommando wurde nicht gefunden
130         LD   (FARADR),HL       ;Adresse der gew. Routine merken
140         LD   A,C               ;Romselect in den Akku
150         LD   (FARADR+2),A      ;und ebenfalls abspeichern
160         LD   E,DRIVE           ;0/1 für Drive A/B
170         LD   D,TRACK           ;0 bis 39 für den gew. Track
180         LD   C,SECTOR          ;Sektornummer mit Format-Offset
190         LD   HL,BUFFER         ;512 Byte für die Daten des Sektors
200         RST  &18               ;wirkt wie ein CALL zur gew. Routine
210         DEFW FARADR            ;Vektor auf die Drei-Byte Far Adress
```

```
220          RET
230 COMMAND: DEFB &84              ;Name des Kommandos
240 FARADR:  DEFS 3                ;Platz für die Drei-Byte-Adresse
```

Nun wollen wir an dieser Stelle nicht gleich einen vollwertigen Diskmonitor schreiben. Das folgende BASIC-Programm stellt aber bereits die Funktion 'Lesen von Sektoren' zur Verfügung. Tippen Sie dieses Programm ruhig ab, erweitern und analysieren Sie es, um mit der Handhabung der Befehle vertraut zu werden. Dieses kurze Programm stellt das Grundgerüst des schon erwähnten Diskmonitors dar.

Bevor Sie sich von der korrekten Arbeitsweise des Programms überzeugt haben, sollten Sie auf jeden Fall bei der verwendeten Diskette den Schreibschutz anbringen. Es genügt ein einziges falsches Byte, nämlich das Kommandobyte, und der ausgewählte Sektor wird nicht gelesen, sondern mit den Daten des Buffers überschrieben. Wenn Sie auf diese Weise z.B. den ersten Sektor des Directory Ihrer CP/M-MasterDiskette mit lauter Nullen überschreiben, so können wir nur hoffen, daß Sie davon bereits eine Kopie angefertigt hatten. Sonst sind 16 Programme auf dieser Diskette mit Sicherheit verloren. Beherzigen Sie bitte in Ihrem eigenen Interesse unseren Tip an dieser Stelle:

BACKUP, BACKUP, BACKUP !!!!

```
100 DEFINT a-z
110 MEMORY &A000-1
120 FOR adress= &A000 TO &A01C
130 READ byte
140 POKE adress,byte
150 check = check + byte
160 NEXT adress
170 IF check <> 2941 THEN PRINT"fehler in datas !": END
180 MODE 2
190 INPUT "Output Stream (0/8)";dev
200 INPUT "Laufwerk      (0/1)";drive
210 INPUT "Track        (0-39)";track
```

```
220 INPUT "Sector        (1-9)";sector
230 POKE &A020,drive
240 POKE &A021,track
250 POKE &A022,sector+64
260 CALL &A000
270 MODE 2
280 lincnt = 0
290 FOR i = &A030 TO &A030+511
300 IF lincnt = 0 THEN PRINT#dev,"   ";HEX$(i-&A030,4);"  ";
310 PRINT#dev,HEX$(PEEK(i),2);" ";:lincnt=lincnt+1
320 IF lincnt = 16 THEN lincnt=0
330 a=PEEK(i):a=a AND 127
340 IF a<32 OR a=127 THEN t$="." ELSE t$=CHR$(a)
350 lin$=lin$+t$
360 IF lincnt = 0 THEN PRINT#dev,lin$:lin$=""
370 NEXT
380 PRINT" <SPACE> fuer weiter "
390 IF INKEY (47) THEN 390
400 GOTO 180
410 DATA &21,&1c,&a0,&cd,&d4,&bc,&22,&1d
420 DATA &a0,&79,&32,&1f,&a0,&21,&20,&a0
430 DATA &5e,&23,&56,&23,&4e,&21,&30,&a0
440 DATA &df,&1d,&a0,&c0,&84
```

Die Handhabung dieses Programms ist eigentlich selbsterläuternd. Die Übergabe der Parameter geschieht nach dem Briefkasten-Prinzip, d.h. wir schreiben die Werte in bestimmte Speicherzellen, aus denen sie dann vom Maschinenprogramm abgeholt werden. Das ist leider nötig, da von BASIC aus keine Möglichkeit besteht, Register des Z80 direkt mit irgendwelchen Werten zu laden.

Nachdem die benötigten Parameter in den Speicher gepoked worden sind, wird die Maschinenroutine mittels CALL angesprungen. Der angegebene Sektor wird auf der Diskette gesucht und die Daten in einen 512 Byte großen Buffer ab &A030 geschrieben. Von hier werden die Bytes vom BASIC-Programm

ausgelesen und in hexadezimaler Form auf dem Bildschirm angezeigt. Wahlweise können Sie die Ausgabe auf einen angeschlossenen Drucker umleiten.

2.2.2.5 Der Befehl &85 Write Sector

Allein das Lesen von Daten ist schon recht interessant. Erlaubt die zuvor dargestellte Routine doch einen recht interessanten Blick z.B. in das Directory oder in die Systemspuren 0 und 1 einer CP/M-Diskette. Wir können aber noch mehr. Das Schreiben von Daten auf jeden beliebigen Sektor der Diskette ist mit dem Befehl &85 genau so einfach wie das Lesen mit dem Befehl &84. Sogar die benötigten Parameter, also Drivenummer, Track, Sektor und Bufferadresse der Daten werden in denselben Registern wie beim Lesen übergeben. Damit reduziert sich die Änderung des zuvor abgedruckten Programms auf ein einziges Byte. Nur die Nummer des Befehls, das letzte DATA-Element im BASIC-Programm, und natürlich die Prüfsumme müssen geändert werden.

Zum Experimentieren können Sie einmal eine frisch formatierte Diskette nehmen und versuchen, einen Sektor zu beschreiben. Dazu können Sie irgendwelche Bytes in den Sektorbuffer POKEn und diesen dann schreiben. Mit der abgedruckten Leseroutine können Sie anschließend prüfen, ob alles nach Wunsch geklappt hat.

2.2.2.6 Der Befehl &86 Format Track

Dieser Befehl ist etwas für die Spezialisten unter Ihnen. Er erlaubt die Formatierung einer einzelnen Spur auf der Diskette. Unter Zuhilfename des Befehls &86 können Sie in Ihre Programme Formatierroutinen einbauen, so daß der Anwender nicht auf das CP/M-Programm 'FORMAT.COM' angewiesen ist.

Bevor wir diesen 'verborgenen'Befehl der Befehlserweiterung weiter erläutern, müssen wir klären, wie der FDC 765 überhaupt eine Spur formatiert. Um jetzt nicht der späteren Beschreibung

des FDC 765 vorzugreifen, sei an dieser Stelle nur so viel verraten:

Der Floppy-Controller ist in Hinsicht auf die Formatierung von Disketten außerordentlich 'benutzerfreundlich'. Ihm genügen einige, wenige Bytes, und er erledigt die gesamte 'Dreckarbeit', wie z.B. das Erzeugen der Checksummen, der verschiedenen ID-Markierungen und der sogenannten GAPs. Weiterhin passt er auf, wann das Indexloch den Spuranfang markiert und erkennt selbsttätig, ob die Diskette schreibgeschützt ist.

Soll dieser pfiffige Baustein einen Track, also eine Spur formatieren, so benötigt er zunächst das entsprechende Kommando. Dieses Kommando sagt ihm (unter anderem), auf welchem Drive er welchen Track formatieren soll, wie groß die zu erzeugenden Sektoren sind, und welchen Wert er als Fillerbyte verwenden soll. Das Fillerbyte ist der Wert, der nach erfolgter Formatierung als Daten auf allen Sektoren steht.

Daß zum Formatieren der Laufwerksmotor eingeschaltet sein muß, dürfte klar sein. Sobald nun nach dem Kommando das Indexloch erkannt wird, beginnt der eigentliche Formatiervorgang. Jetzt erwartet der FDC zu jedem zu formatierenden Sektor 4 Bytes vom Prozessor. Zu jedem Sektor müssen Tracknummer, Kopfnummer, Sektornummer und Größe des Sektors angegeben werden. Dadurch, daß zu jedem Sektor auch die Sektornummer angegeben werden muß, können die Sektoren in einer vom Programmierer festgelegten Reihenfolge auf die Diskette gebracht werden. Dadurch kann der spätere Zugriff teilweise erheblich beschleunigt werden.

Doch dazu später mehr. Wir wollen jetzt sehen, was das alles mit dem Kommando &86 auf sich hat.

Das Kommando &86 erwartet, ähnlich wie die zuvor beschriebenen Befehle zum Lesen und Schreiben von Sektoren, in verschiedenen Registern Parameter. Da wäre zunächst einmal der gewünschte Track. Dieser Wert wird in bekannter Weise im D-Register übergeben. Im E-Register des Z80 muß sich die Nummer für das gewünschte Laufwerk befinden. Das C-Register

bekommt die Nummer des ersten zu formatierenden Sektors mitgeteilt und das HL-Registerpaar wird als Zeiger auf eine Tabelle benutzt.

Die ersten drei Register und ihre Funktionen unterscheiden sich nicht von denen der zuvor beschriebenen Routinen. Aber auch das HL-Registerpaar hat eine dem Schreiben von Daten vergleichbare Funktion. In der durch HL adressierten Tabelle stehen nämlich für jeden zu formatierenden Sektor 4 Bytes, die schon genannten Werte für Track, Head und Drive, Sektornummer und Sektorgröße. Das folgende Beispielprogramm soll einmal zeigen, wie eine einzelne Spur formatiert werden kann.

```
100                     ;FORMATIEREN EINES KOMPLETTEN TRACKS
110         ORG  &5000  ;Startadresse der Formatier-Routine
120 START:  LD   E,DRIVE ;gewünschte Laufwerksnummer ins E-Register
130         LD   D,TRACK ;Tracknummer ins D-Register
140         LD   C,&41  ;erste Sektornummer auf dem Track
150         LD   HL,FTAB ;Tabelle der 32 Bytes fuer den FDC
160         RST  &18    ;Call Format &87
170         DEFW FORMAT ;Adresse der Drei-Byte-Far Adress
180         RET         ;Track ist formatiert
190 FORMAT: DEFW &C652  ;Adresse der Routine &87 im AMSDOS
200         DEFB 7      ;benötigte Rom-Select-Adresse
210 FTAB:   EQU  $
220 SECT1:  DEFB TRACK  ;Tracknummer für die Sektor-ID
230         DEFB HEAD   ;Kopfnummer des Drives
240         DEFB &41    ;Nummer des Sektors
250         DEFB 2      ;Sektorgröße für die ID
260 SECT2:  DEFB TRACK
270         DEFB HEAD
280         DEFB &43
290         DEFB 2
300 SECT3:  DEFB TRACK
310         DEFB HEAD
320         DEFB &45
330         DEFB 2
340 SECT4:  DEFB TRACK
350         DEFB HEAD
360         DEFB &47
```

```
370          DEFB 2
380 SECT5:   DEFB TRACK
390          DEFB HEAD
400          DEFB &49
410          DEFB 2
420 SECT6:   DEFB TRACK
430          DEFB HEAD
440          DEFB &42
450          DEFB 2
460 SECT7:   DEFB TRACK
470          DEFB HEAD
480          DEFB &44
490          DEFB 2
500 SECT8:   DEFB TRACK
510          DEFB HEAD
520          DEFB &46
530          DEFB 2
540 SECT9:   DEFB TRACK
550          DEFB HEAD
560          DEFB &48
570          DEFB 2
```

2.2.2.7 Der Befehl &87 Seek Track

Obwohl alle bisherigen Befehle mit direktem Zugriff auf die Diskette selbsttätig die gewünschte Spur angefahren haben, so kann es doch für manche Aufgaben sinnvoll sein, den Kopf über einer bestimmten Spur zu positionieren. Diese Aufgabe übernimmt der Befehl &87.

Zur Positionierung sind nur zwei Parameter erforderlich. Zunächst wird die Angabe des gewünschten Laufwerks gefordert. Die Tracknummer, die vom Kopf angefahren werden soll, wird als zweiter Parameter gefordert.

Auch bei dieser Routine haben sich die Programmierer an die Register als Schnittstelle gehalten. Die Laufwerksnummer wird

im E-Register benötigt, das D-Register muß die Tracknummer enthalten.

Auch die Benutzung der Flags als Kennzeichen für den Erfolg der Routine ist wie gehabt. Ein gesetztes Carry teilt Ihnen mit, daß die gesuchte Spur auf der Diskette gefunden wurde.

Da sich auch das benötigte Programm nicht wesentlich von den zuvor abgedruckten Routinen unterscheidet, wollen wir an dieser Stelle einmal darauf verzichten.

2.2.2.8 Der Befehl &88 Test Drive

Möchten Sie in einem Maschinenprogramm gern wissen, ob ein bestimmtes Laufwerk verfügbar ist? Nichts einfacher als das. Der Befehl &88 verrät alle wichtigen Informationen über das angewählte Laufwerk. Allerdings stellt diese Routine in gewisser Weise eine Besonderheit dar, da es die Angabe über das zu testende Laufwerk nicht, wie sonst üblich, im E-Register, sondern im Akku erwartet.

Kehrt diese Routine zurück, so kann an Hand der Flags und des Inhalts des Akku ermittelt werden, ob das angewählte Laufwerk verfügbar ist. Im Falle einer fehlerfreien Ausführung der Routine ist das Carry-Flag gesetzt. Der Akku enthält dann den Inhalt des Statusregisters 0 des FDC. Hier ist nur interessant, daß bei Verfügbarkeit der Laufwerke je nach gewähltem Drive eine Null (bei Drive A) oder Eins (bei Drive B) im Akku enthalten ist.

2.2.2.9 Der Befehl &89 Retry Count

Soll auf einer Diskette der Kopf über einer bestimmten Spur positioniert werden, so kann das recht einfach mit dem Befehl &87 geschehen. Bei diesem Befehl wird ja dem Controller die gewünschte Spur mitgeteilt. Nachdem das Laufwerk den Schreib-Lesekopf mit der entsprechenden Anzahl von Schritten in die gewünschte Richtung bewegt hat, liest der FDC selbsttätig

die beim Formatieren auf die Spur gebrachte Tracknummer von der Diskette.

Stimmt die gelesene Nummer mit der gewünschten überein, so ist das Kommando erfolgreich beendet. Anders ist der Fall, wenn keine Information gelesen werden kann, oder die gelesene von der gewünschten Spurnummer verschieden ist. In diesem Fall wird ein neuer Versuch unternommen, den geforderten Track zu finden.

Genau an dieser Stelle greift der Befehl &89 ein. Mit diesem Befehl ist es möglich, die Anzahl der Leseversuche nach dem Positionieren festzulegen. Das AMSDOS stellt einen Default-Wert von 10 Versuchen ein. Das ist normalerweise ausreichend. Es kann allerdings nötig sein, diese Zahl zu erhöhen.

Der gewünschte neue Wert wird der Routine &89 im Akku übergeben. Der Wert 0 wird als Maximalzahl von 256 verstanden. Kleinere Werte als 10 sollten Sie nicht unbedingt einsetzen, Disketten mit Leseproblemen kann man aber schon einmal mit einer höheren Anzahl von Versuchen zu Leibe rücken.

Um die Auswirkungen schnell und ohne Assembler zu prüfen, können Sie den Wert 0 in die Speicherzelle &BE66 POKEn. Das ist gleichbedeutend mit dem Aufruf des Befehls &89 und der Übergabe von 0 im Akku. Wenn Sie dann eine frische, also nicht formatierte Diskette einlegen und sich das Directory anschauen wollen, können Sie sehr schön hören, wie der Kopf auf der Diskette hin- und her'ackert', um die benötigte Spur zu finden. POKEn Sie dagegen eine 1 in die Speicherzelle &BE66, dann gibt sich das Laufwerk deutlich weniger Mühe, auf der leeren Diskette irgendwelche Daten zu finden.

Kapitel 3
Technik der Floppy und der Diskette

3.1 Der Floppycontroller

Während beim CPC 464 die Elektronik des Floppyinterfaces auf das Rechnergehäuse aufgesteckt wird, befindet sie sich beim CPC 664 und CPC 6128 zusammen mit der ersten Floppy im Rechnergehäuse. Welchen CPC Sie besitzen, ist für die Funktion des Interfaces recht unerheblich, da sie sich nur sehr geringfügig voneinander unterscheiden. Die globalen Funktionsblöcke können Sie im Blockschaltbild Abb. 16 sehen.

Den Mittelpunkt des Controller-Boards bildet der integrierte Floppy Disk Controller (FDC) uPD 765. Dieses IC stellt die Schnittstelle zwischen den Laufwerken und dem Prozessor des CPC dar. Zwar kann man durchaus Floppystationen ohne FDC aufbauen, durch die hohe 'Eigenintelligenz' des FDC vereinfacht sich die Konstruktion jedoch wesentlich. Der nötige Hardwareaufwand, aber auch das Ausmaß der zu schreibenden Betriebssoftware wird durch den Einsatz eines FDC drastisch reduziert. Ein Beispiel mag dies verdeutlichen.

Die Floppystation 1541 der Firma Commodore, vielen von Ihnen sicher bekannt als Floppystation zum Commodore C64, ist eine ohne FDC aufgebaute Floppystation. Abgesehen von der konstruktiv bedingten langsamen Geschwindigkeit der Datenübertragung (über die Sie als CPC-Besitzer nur lächeln können) ist der Aufwand der Hardware für dies Laufwerk deutlich höher als bei der CPC-Floppy. Die Digitalelektronik der 1541 enthält einen eigenen Prozessor, zwei 40-polige Peripherie-IC und jede Menge verschiedener TTL-ICs. Ein kompletter CPC enthält eine vergleichbare Menge an Bauteilen!

Die Betriebssoftware für die 1541 ist mit etwa 16 K doppelt so groß wie das AMSDOS. Keine Frage, daß Entwickler (aus Gründen der Bequemlichkeit) und Kaufleute (aus Kostengründen) gern zu den komfortabel einzusetzenden FDCs greifen.

Bevor wir nun auf den folgenden Seiten sozusagen den Deckel des FDCs entfernen und ihm auf die Finger sehen, wollen wir zunächst die einzelnen Stufen des Floppy-Controllers untersuchen.

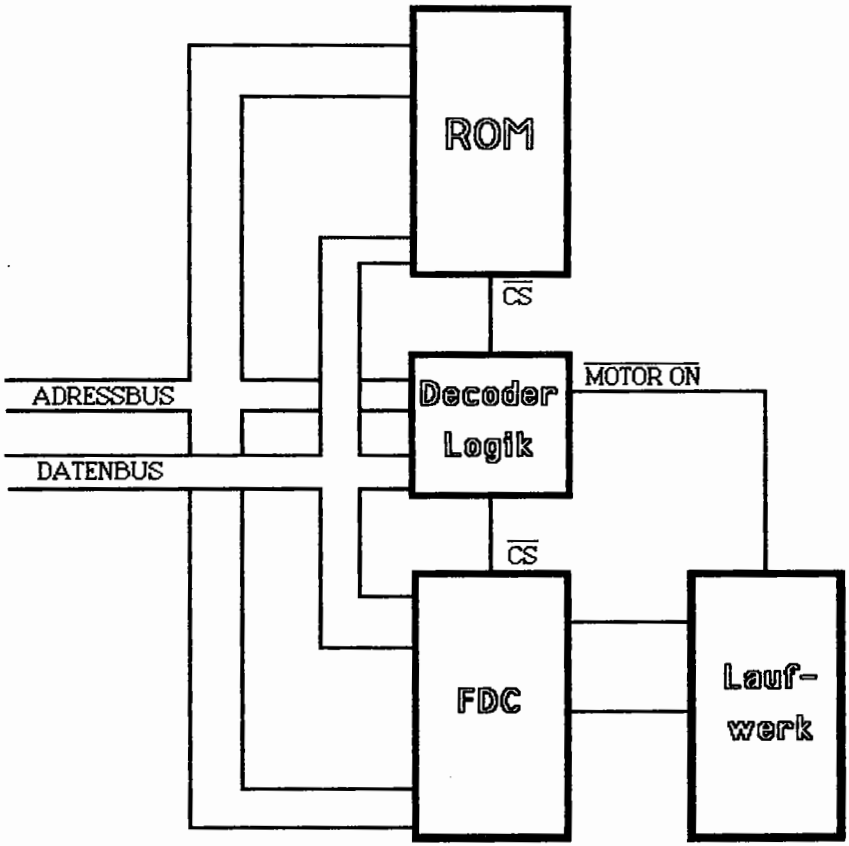


Abb. 16

3.1.1 Beschreibung der Floppy-Schnittstelle

Wie auf dem Blockschaltbild Abb. 16 zu sehen, kann die gesamte Elektronik der Schnittstelle in drei Funktionsgruppen unterteilt werden. Die erste Funktionsgruppe besteht aus dem AMSDOS-ROM und einem Flip-Flop. Das Flip-Flop hat die Aufgabe, das BASIC des CPC auf Kommando hin aus- und das AMSDOS-ROM einzuschalten. Dieses Flip-Flop wird aktiviert, wenn auf die Portadresse &DFxx ein Wert von &07 ausgegeben wird. Jetzt ist bei einem Lesezugriff auf den Adressbereich von &C000 bis &FFFF das AMSDOS-ROM eingeschaltet. Jeder andere ausgegebene Wert schaltet das Flip-Flop und damit das AMSDOS-ROM wieder ab.

Da das ROM in einem 28-poligen IC-Sockel sitzt, besteht die Möglichkeit, das Betriebssystem zu modifizieren oder zu erweitern und ohne Basteleien gegen ein 16K-EPROM auszutauschen. Und für eigene Erweiterungen ist in den genutzten 16 KByte als Platz genug. Das AMSDOS belegt nämlich nur knapp 8K des verfügbaren Speicherplatzes. Die verbleibenden 8K werden von LOGO benötigt, das ohne diesen Bereich im ROM fast keinen Platz für LOGO-Programme mehr ließe. Wenn man jedoch auf die Programmiersprache LOGO verzichten kann, so kann ein erweitertes Floppy-Betriebssystem immerhin 16K groß werden.

Das zweite im Blockschaltbild gezeigte Flip-Flop steuert die Laufwerksmotoren. Der Dateneingang des Flip-Flops liegt auf dem Datenbit 0. Der CLK-Impuls für das Flip-Flop wird über mehrere Gatter aus den Adressbits A7, A8 und A10 sowie den Signalen IORQ und WR gewonnen. Die genannten Adressbits müssen Low sein, um das Flip-Flop zu triggern. Daraus ergibt sich eine Portadresse von &FA7x, im AMSDOS wird das Flip-Flop über den PORT &FA7E angesprochen.

Die Gatter zur Decodierung der Portadresse des Motor-FlipFlops werden auch zur Decodierung der Adressen des FDC verwendet. In diesem Fall aber muß das Adressbit A8 High sein, um das CS-Signal zu generieren. Das ergibt die Portadresse &FB7X. Da das Adressbit A0 des Prozessors zur Auswahl der beiden Register des FDC verwendet werden, ergeben sich die Portadressen

&FB7E für das Haupt-Status-Register und &FB7F für das Datenregister.

Interessant ist noch die Erzeugung der beiden Signale zum aktiviere der Laufwerke (Drive-Select-Signale). Obwohl der FDC über zwei getrennte Anschlüsse (US0, US1) für die Erzeugung des Drive Select verfügt, ist nur ein Anschluß benutzt. Das Ausgangssignal des Pin 29, US0, wird über ein als Inverter geschaltetes NAND-Gatter als Select-Signal für das Drive B und über ein weiteres NAND-Gatter/Inverter als Select-Signal für Drive A benutzt. Je nach Polarität des US0-Signals ist also entweder Drive A oder Drive B aktiv. Durch diesen Aufbau hatten es die Programmierer des AMSDOS recht einfach. Es genügt ein einfaches Decrementieren des Wertes für die Laufwerksauswahl, um festzustellen, welches Laufwerk aktiviert werden soll. Ist der Wert nach dem decrementieren nicht Null, dann war das Laufwerk A aktiv, ist der Wert dagegen Null, so ist Laufwerk B aktiv.

3.1.2 Der FDC 765

Der von der Firmen NEC als uPD 765 von ROCKWELL als R 6765 und von INTEL als 8765 vertriebene FDC kann als ein hoch spezialisierter Microprozessor angesehen werden. Die Möglichkeiten dieses ICs sind so umfangreich und komplex, daß diese Bezeichnung sicher nicht zu hoch gegriffen ist.

Das vom FDC benutzte Datenformat entspricht dem IBM-Format 3740 in Single Density und IBM System 84 in Double Density. Durch diese Festlegung können z.B. Commodore- oder Apple-Disketten leider nicht gelesen oder beschrieben werden.

Mit seinen 40 Pins stellt er alle für den Betrieb handelsüblicher Laufwerke benötigten Signale zur Verfügung. Durch die vorhandenen Steuersignale ist der Entwickler in der Lage, diesen FDC an fast jeden Prozessor anzuschließen. Dabei sind zwei grundsätzliche Möglichkeiten zum Anschluß und Betrieb gegeben. Die erste Methode ist der DMA-Betrieb. Zusammen mit einem DMA-Controller kann der FDC für den Datentransfer beim

Lesen und Schreiben die Kontrolle über den Speicher des Computersystems übernehmen. Er holt sich dann mit Hilfe des DMA-Controllers benötigte neue Daten aus dem Speicher oder schreibt, ebenfalls unter Umgehung des Prozessors, die von der Diskette gelesenen Werte in den Speicher. Diese sehr schnelle Methode des Datentransfers wird aber im CPC nicht eingesetzt und wird hier nur der Vollständigkeit halber erwähnt.

Bei der zweiten, im CPC eingesetzten Methode wird der Datentransfer vom Prozessor übernommen. Bei dieser zweiten Methode muß aber wiederum zwischen zwei Möglichkeiten des Betriebs des FDC unterschieden werden.

Da wäre zunächst die Interrupt-Methode. Hierbei wird zu jedem Datentransfer ein Interrupt erzeugt. In der entsprechenden Interupt-Routine des Prozessors muß dann das nächste Daten- oder Befehlsbyte vom Prozessor geliefert oder gelesen werden. Durch den Hardwareaufbau des CPC kam diese Methode wohl auch nicht in Betracht, so daß die Entwickler zur Polling-Methode gegriffen haben. Hierbei muß der Prozessor regelmäßig in Registern des FDC prüfen, welche Aktion als nächstes vom FDC gefordert wird.

Doch betrachten wir zunächst einmal die Leistungsdaten des 765 im Überblick. Bedenken Sie jedoch, daß die Entwickler des Controllers nicht alle Möglichkeiten des 765 genutzt haben.

* *programmierbare Sektorlänge*
* *alle Laufwerkdaten programmierbar*
* *bis zu vier Laufwerke anschließbar*
* *Datentransfer wahlweise im DMA- oder Nicht-DMA-Modus*
* *anschließbar an fast alle gängigen Prozessortypen*
* *einfache 5-Volt-Stromversorgung*
* *einfacher Ein-Phasen-Takt von 4 oder 8 MHz*
* *40-poliges IC-Gehäuse*

Dem letzten Punkt dieser kurzen Aufstellung wollen wir uns jetzt etwas detaillierter zuwenden.

3.1.2.1 Die Anschluß-Belegung des FDC

Die Anschlüsse des FDC 765 lassen sich in verschiedene Gruppen unterteilen. Die erste Gruppe von Anschlüssen stellen das Interface zum Systemprozessor dar. Über diese Anschlüsse wird also die Steuerung des FDC vom Prozessor aus vorgenommen.

Die zweite Gruppe ist nur in Verbindung mit dem DMA-Betrieb nötig. Über diese Signale kommunizieren DMA-Controller und FDC.

Das Interface zu den Floppy-Laufwerken wird von der dritten, mit 19 Anschlüssen zahlenmäßig stärksten Gruppe der Anschlüsse geliefert.

In der vierten und letzten Gruppe lassen sich die Anschlüsse für die Stromversorgung und den Takt zusammenfassen.
Beginnen wir die Betrachtung der Anschlüsse mit der ersten Gruppe, dem Interface zum Prozessor.

Das Prozessor-Interface

RESET:

Der RESET-Eingang des FDC ist High-Aktiv. Im normalen Betrieb liegt dieser Anschluß auf MassePotential. Durch ein High am RESET-Pin wird der FDC in einen definierten Zustand gebracht.

CS: CHIP SELECT*

Durch ein Low an diesem Pin wird der FDC selektiert. Erst bei CS* = Low werden RD* und WR* für den FDC gültig. Da die Erzeugung des CS dem Entwickler freigestellt ist, kann der FDC wahlweise Memory-Mapped, also als Bestandteil des Speicherbereichs, oder über Portadressen angesprochen werden.

RD: READ**

Dieser Anschluß muß mit dem RD*-Signal des Prozessors verbunden werden. Wann immer der Prozessor Daten aus dem FDC lesen will, wird diese Leitung auf Low gelegt.

WR: WRITE**

Wie die RD*-Leitung Lesezugriffe des Prozessors signalisiert, so zeigt ein Low an WR*, daß der Prozessor Daten oder Befehle in den FDC schreibt.

A0: ADRESS LINE 0

Der FDC verfügt über nur zwei von außen ansprechbare Adressen. Die Unterscheidung zwischen den beiden Adressen wird mit dem Signal A0 vorgenommen. Diese Leitung ist normalerweise mit dem niedersten Adressbit des Prozessors verbunden.

DB0 - DB7: DATABUS 0-7

Diese Anschlüsse des FDC werden mit dem Systemdatenbus verbunden. Alle Kommandos und Daten werden über diese acht bidirektionalen Anschlüsse transportiert. Die jeweilige Datenrichtung wird dabei entweder vom Prozessor oder im DMA-Mode vom DMA-Controller bestimmt.

INT: INTERRUPT

Über diesem Anschluß kann der FDC einen Interrupt des Systemprozessors erzeugen. Interrupts werden zu jedem Byte-Transfer erzeugt (im CPC nicht angeschlossen).

Signale für den DMA-Modus (im CPC nicht verwendet)

DRQ: DMA REQUEST

Über diesen Anschluß signalisiert der FDC dem DMA-Controller, daß ein Speicherzugriff erfolgen soll. Bei der nächsten mög-

lichen Gelegenheit übernimmt daraufhin der DMA-Controller den Systembus. Der Prozessor wird dabei abgeschaltet.

DACK: DMA ACKNOWLEDGE*

Mit diesem Signal wird dem FDC angezeigt, daß der DMA-Controller den Bus übernommen hat und jetzt mit dem Datentransfer begonnen hat.

TC: TERMINAL COUNT

Durch einen High-Pegel an diesem Anschluß wird der Datentransfer von und zum FDC unterbrochen. Obwohl dieser Anschluß in der Hauptsache im DMA-Modus verwendet wird, kann auch in interruptgesteuerten Systemen der Datentransfer über diesen Anschluß unterbrochen werden.

Das Floppy-Interface

US0, US1: UNIT SELECT 0/1

Über diese beiden Anschlüsse können direkt zwei, mit Hilfe eines Zwei-zu-Vier-Decoders jedoch vier Laufwerke angeschlossen werden. Über diese Anschlüsse wird das jeweils gewünschte Laufwerk zum Lesen oder Schreiben von Daten angesprochen.

HD: HEAD SELECT

Da der FDC für den Betrieb von DoppelkopfLaufwerken vorbereitet ist, kann bei Verwendung dieser Drives die Kopfauswahl über diesen Anschluß geschehen.

HDL: HEAD LOAD

Dieses Signal wird fast ausschließlich bei 8"-Laufwerken eingesetzt. Die Motoren dieser Laufwerke werden nicht bei Bedarf eingeschaltet, sondern laufen normalerweise immer. Um nun Diskette und Schreiblesekopf zu schonen, wird der Kopf nur bei Bedarf über einen Hubmagneten 'geladen', also an die Disket-

tenoberfläche gebracht. Die Steuerung des Hubmagneten wird dann mittels HDL vorgenommen.

IDX: INDEX

An diesem Anschluß wird das von der Index-Lichtschranke erzeugte Signal angelegt und signalisiert dem FDC den physikalischen Anfang eines Tracks.

RDY: READY

Das von der Floppy gelieferte Signal READY zeigt an, daß sich im Laufwerk eine Diskette befindet und daß sich diese mit einer gewissen Mindestgeschwindigkeit dreht. Erst nach Erscheinen des READY greift der FDC auf das Laufwerk zu.

WE: WRITE ENABLE

Dieser Ausgang des FDC muß High sein, um Daten auf die Diskette schreiben zu können.

RW/SEEK: READ WRITE/SEEK

Insgesamt liefert ein Floppylaufwerk mehr Signale, als bei einem 40-poligen Gehäuse für das Floppy-Interface zur Verfügung stehen. Allerdings werden nicht zu allen Zeiten alle Signale gleichzeitig benötigt. Acht dieser Laufwerkssignale hat man darum in zwei Gruppen aufgeteilt, die wahlweise an vier Anschlüsse des FDC gelegt werden können. Über den Anschluß RW/SEEK wählt der FDC selbsttätig die jeweils benötigten Signale aus.

FR/STP: FIT RESET/STEP

Dies ist das erste der vier Doppelsignale am FDC. Dieser Ausgang hat je nach ausgeführter Operation verschiedene Bedeutungen. Einmal kann mit diesem Anschluß das bei einigen Laufwerken vorhandene Fehler-Flip-Flip zurückgesetzt werden. Die zweite, weitaus häufigere Verwendung ist die Ansteuerung

des Step-Eingangs des Laufwerks. Zu jedem Kopfwechsel werden die benötigten Impulse an diesem Anschluß geliefert.

FLT/TR0: FAULT/TRACK0

Auch dieser Eingang kann zwei verschiedene Signale auswerten. Wird eine SEEK-Operation (siehe Programmierung des FDC) durchgeführt, dann wird an diesem Anschluß das Track0-Signal des Laufwerks erwartet. Dies Signal wird durch eine Lichtschranke oder einen mechanischen Schalter erzeugt, wenn der Schreib/Lesekopf auf der physikalischen Spur 0 steht. Die zweite Funktion, das Fault-Signal, wird von einigen Laufwerken im Fehlerfall generiert und kann vom FDC mit dem zuvor beschriebenen Signal FR/STP wieder gelöscht werden. Dieses Signal wird bei Read/Write-Operationen des FDC überprüft.

LCT/DIR: LOW CURRENT/DIRECTION

Die Step-Impulse von FR/STP geben ja nur an, daß der Kopf bewegt werden soll. LCT/DIR bestimmt dazu im Seek-Modus die Richtung der Kopfbewegung. Die Funktion LOW CURRENT wird beim Schreiben der Daten benötigt. Durch dieses Signal läßt sich der Schreibstrom auf den inneren Spuren verringern. Einzelheiten zu diesem Signal finden Sie in der Beschreibung der theoretischen Grundlagen der Diskettenspeicherung.

WP/TS: WRITE PROTECT/TWO SIDE

Bei allen Laufwerken wird der Zustand des Schreibschutzes als Signal an den Controller gemeldet. Dies Signal wird vom Eingang WP/TS bei Schreib/Leseoperationen überprüft. Das Signal TS wird bei Seek-Operationen überprüft. Es wird nur in Verbindung mit Doppelkopflaufwerken benötigt.

WDA: WRITE DATA

Über diesen Anschluß werden die seriellen Schreib-Daten an das Laufwerk geliefert. Das können sowohl die beim Schreiben eines

Sektors vorkommenden Daten wie auch alle beim Formatieren benötigten Informationen sein.

PS0,1: PRE SHIFT 0/1

Über diese Anschlüsse teilt der FDC bei Double Density-Format (MFM) einer geeigneten Elektronik mit, wie der serielle Datenstrom auf die Diskette geschrieben werden muß. Möglich sind die drei Zustände EARLY, NORMAL und LATE für die Precompensation.

RD: READ DATA

Über diesen Eingang werden die von der Diskette gelesenen Informationen in den FDC gegeben. Aus diesem seriellen Bitstrom werden die ursprünglich geschriebenen Bytes zurückgewonnen.

RDW: READ DATA WINDOW

Dieses Signal wird aus den gelesenen Daten in einem Datenseparator gewonnen. Näheres finden Sie im Kapitel Grundlagen der Diskettenspeicherung.

VCO: VCO SYNC

Dieses Signal wird zur Steuerung des VCO im PLL-Datenseparator benötigt.

MFM: MFM MODE

Dieser Anschluß signalisiert, ob der Controller im Single Density-Format (MF) oder im Double Density-Format (MFM) arbeitet.

Stromversorgung und Taktsignale

Vcc: +5 Volt

Über diesen Anschluß erhält der FDC seine Versorgungsspannung. Die Spannung von 5 Volt sollte im Bereich von +- 5%

konstant sein. Der benötigte Strom des FDC beträgt max. 150 mA.

GND: GROUND

Masseanschluß des FDC

CLK: CLOCK

Der FDC benötigt einen Takt. Je nach Laufwerken muß dieser Takt 4 MHz (bei 5¼" und kleiner) oder 8 MHz (bei 8") betragen.

WCK: WRITE CLOCK

Die Frequenz dieses Signals muß je nach Datenformat gewählt werden. Bei MF muß der Takt 500 kHz, bei MFM 1 MHz betragen. Diese Frequenz bestimmt die Übertragungsgeschwindigkeit der Daten von und zur Floppy.

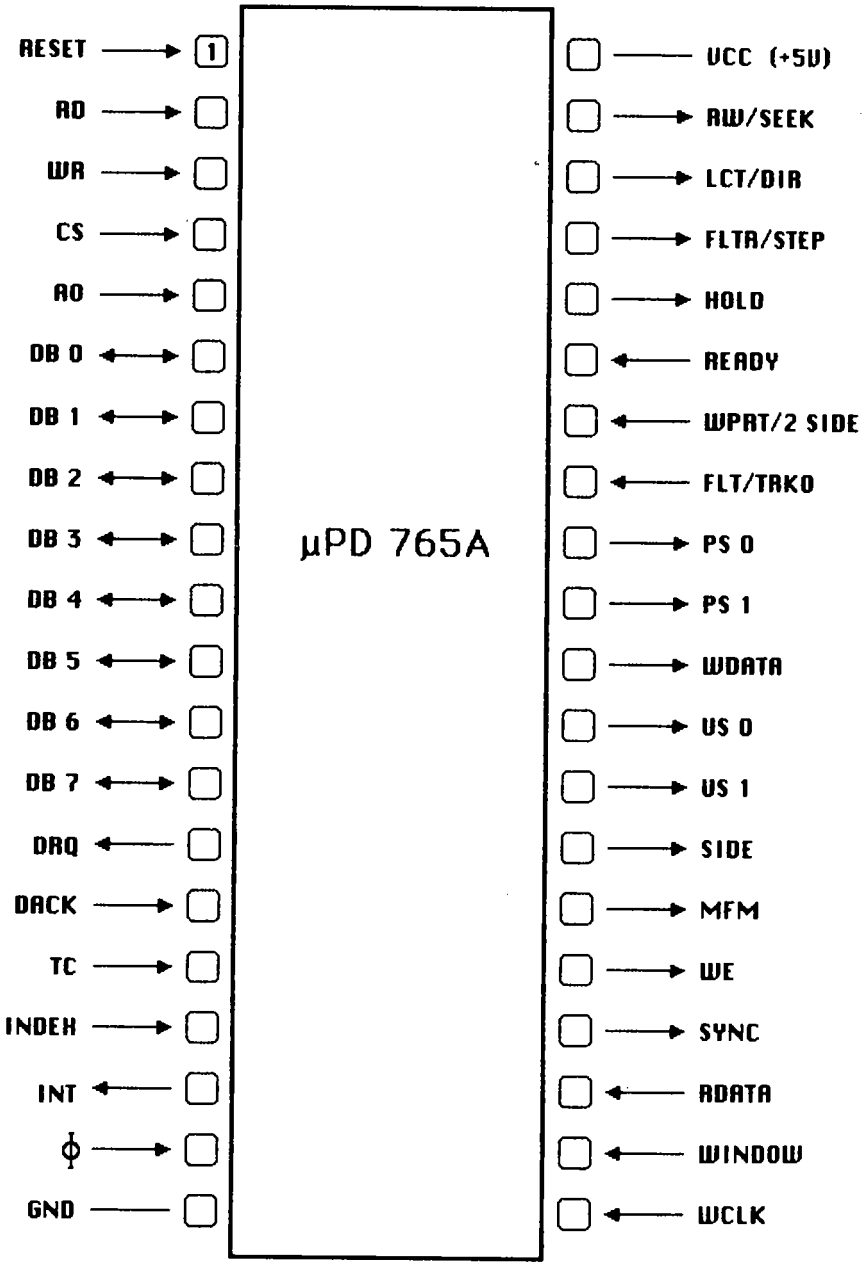

RESET
RD
WR
CS
A0
DB 0
DB 1
DB 2
DB 3
DB 4
DB 5
DB 6
DB 7
DRQ
DACK
TC
INDEX
INT
φ
GND
1
μPD 765A
VCC (+5V)
RW/SEEK
LCT/DIR
FLTR/STEP
HOLD
READY
WPRT/2 SIDE
FLT/TRK0
PS 0
PS 1
WDATA
US 0
US 1
SIDE
MFM
WE
SYNC
RDATA
WINDOW
WCLK

Abb. 17

3.1.2.2 Die Programmierung des FDC 765

Der FDC 765 verfügt nach außen hin über nur zwei Adressen oder Register. Welches Register zur Verfügung steht, wird durch den Pegel des Signals A0 bestimmt. Liegt A0 auf Masse-Potential, dann kann auf das Haupt-Status-Register zugegriffen werden. Dieses Haupt-Status-Register kann nur ausgelesen werden. Ein High-Pegel an A0 erlaubt den Zugriff auf das Datenregister. Das Datenregister kann sowohl beschrieben als auch gelesen werden. Über dieses Register wird der FDC programmiert, alle Schreib- und Lesedaten transferiert und die Daten der Ergebnisphase an den Prozessor geliefert.

Grundsätzlich müssen bei den Kommandos des FDC drei verschiedene Phasen unterschieden werden. Die erste Phase ist die Kommando- oder Befehls-Phase. In dieser Phase werden dem FDC alle für das Kommando benötigten Parameter übergeben. Das können bei einigen Kommandos bis zu neun Bytes sein. Sobald alle Bytes dieser Phase dem FDC mitgeteilt wurden, beginnt die Ausführungs- oder Execute-Phase. Konkret bedeutet dies, daß z.B. nach einem Lesebefehl jetzt die Daten von der Diskette kommen. Nachdem die Ausführungsphase beendet ist, startet die letzte, die Ergebnis- oder Result-Phase. Während der Ergebnis-Phase liefert der FDC bis zu sieben Statusinformationen, die alle vom Prozessor gelesen werden müssen.

Allerdings trifft das gezeigte Schema nicht auf alle Befehle zu. Bei einigen Befehlen gibt es keine Result-Phase, andere Befehle kennen keine Execute-Phase. Ein Befehl verzichtet sogar auf beide Phasen. Ihm genügt die Befehlsphase.
Bevor wir auf die einzelnen Kommandos zu sprechen kommen, wollen wir zunächst einen Blick auf die Statusinformationen werfen.

Insgesamt verfügt der FDC 765 über 5 Status-Register. Das Haupt-Status-Register belegt, wie erwähnt, eine eigene Adresse (A0=Low) und kann, wie auch die anderen StatusRegister, nur gelesen werden. Allerdings ist auf dieses Register jederzeit ein Zugriff möglich, auch während der Bearbeitung eines Befehls.

Die anderen vier Statusregister sind nur in der Ergebnisphase einiger Befehle zugänglich. Das erste Byte in der Ergebnisphase dieser Befehle zeigt den Zustand von Statusregister 0 (ST0). Die Zustände von ST1 und ST2 werden als Byte Zwei und Drei in der Ergebnisphase hinterher geliefert. Den Zustand von ST3, dem letzten Status-Register, erhält man nur auf besondere Aufforderung mit einem speziellen Befehl, dessen einzige Aufgabe es ist, den Zustand von ST3 zu liefern. Die Bedeutung der StatusRegister werden wir etwas später kennenlernen.

Insgesamt verfügt der FDC 765 über 15 verschiedene Kommandos. Allein durch diese Anzahl wird klar, daß dieser Controller mehr als einfaches Lesen und Schreiben beherrscht. Welche Möglichkeiten er detailliert bietet, soll jetzt untersucht werden. Sollten Ihnen einige in diesem Kapitel nicht erklärte Begriffe wie z.B. Sektor ID, Gap oder Data Adress Mark nichts sagen, so können Sie diese Begriffe im Kapitel über die physikalische Aufzeichnung der Daten nachlesen.

Daten lesen

Bevor der FDC Daten von der Diskette lesen kann, müssen ihm in der Befehlsphase 9 Bytes übermittelt werden. Nach der Angabe aller Daten wird das Signal Head Load aktiviert und die programmierte Head Load Time abgewartet. Danach liest der FDC solange Sektor-IDs, bis er die ID des angegebenen Sektors findet, oder der Index-Impuls zum zweiten Mal nach Beginn der Suche erscheint. Im ersten Fall beginnt er mit der Ausführungsphase, im zweiten Fall wird das Kommando beendet und die Result-Phase beginnt.

Sobald er den Sektor identifiziert hat, beginnt er mit der Ausführungsphase. In der Ausführungsphase werden die Daten von der Diskette gelesen und über den Datenbus an den Prozessor geliefert. Die dabei auftretenden Zeiten sind äußerst kurz. Etwa alle 26 Mikrosekunden steht ein Byte zur Verfügung und muß vom Prozessor gelesen werden.

Nach der Übertragung des letzten Bytes des gewünschten Sektors muß an den FDC ein TERMINAL COUNT-Impuls (TC, Pin 16)

angelegt werden, um die Result-Phase einzuleiten, da ohne TC-Impuls ein sogenannter Multi-Sector-Read durchgeführt wird.

Multi-Sector-Read bedeutet, daß der Controller Daten liest, bis er den letzten Sektor der Spur gelesen hat. Allerdings kann auf den TC-Impuls unter gewissen Voraussetzungen verzichtet werden. In der Befehlsphase zu 'Sektor Lesen' und einigen anderen Kommandos muß nicht nur der gewünschte, sondern auch der letzte Sektor der Spur angegeben werden. Sind nun beide Werte identisch, so bricht der FDC das Lesen automatisch am Ende des gewünschten Sektors ab und beginnt mit der Result-Phase.

MT Multitrack-Bit, wenn gesetzt, wird bei Multi-Sektor-Funktionen die Funktion auf der zweiten Seite der Diskette fortgesetzt, nur bei Doppellaufwerken verfügbar, im AMSDOS immer 0.

MF MFM-Mode-Bit, wenn gesetzt, dann arbeitet der FDC in Double Density, im AMSDOS immer 1.

SK Skip-Bit, wenn gesetzt, werden gelöschte Sektoren (deleted DAM) übersprungen. Unter AMSDOS und CP/M nicht einsetzbar. Immer 0.

HD Head-Select-Bit. Über dieses Bit wird bei Doppellaufwerken die Seitenauswahl vorgenommen. Im AMSDOS immer 0.

US 0,1 Unit Select. Über diese Bits werden die Laufwerke ausgewählt. Unter AMSDOS 0 für Drive A, 1 für Drive B.

R Read, Lesen

W Write, Schreiben

Die genannten Bezeichnungen und Abkürzungen gelten für alle folgenden Darstellungen gleichermaßen.

Phase	R/W	DATEN BUS D7 D6 D5 D4 D3 D2 D1 D0	Bemerkungen
		DATEN LESEN	
Kommando	W	MT MF SK 0 0 1 1 0	Kommando Codes
	W	X X X X X HD US1 US0	
	W	——— Spurnummer ———	Sektor ID Information vor
	W	——— Kopfadresse ———	Kommandoausführung
	W	——— Sektoradresse ———	
	W	——— Sektorgröße ———	
	W	– Letzte Sektor Nr. auf Spur–	
	W	-Lücke zwischen ID u. Daten-	
	W	Sektorlänge wenn Sektorgr.=0	
Ausführung			Daten Transfer zwischen FDD und System
Ergebnis	R	———Status 0———	Zustands Information nach
	R	———Status 1———	Kommandoausführung
	R	———Status 2———	
	R	———Spurnummer———	Sektor ID Information nach
	R	———Kopfadresse———	Kommandoausführung
	R	———Sektornummer———	
	R	———Sektorgröße———	

Abb. 18

In der Result-Phase werden 7 Bytes vom FDC an den Prozessor geliefert, die dieser auch tunlichst alle abholen, sprich lesen, sollte. Bevor nicht das letzte Byte der Result-Phase gelesen worden ist, akzeptiert der FDC kein neues Kommando.

Die ersten drei gelieferten Bytes sind die Zustände der Status-Register 0 bis 2. An Hand dieser drei Register kann der Programmierer alle relevanten Daten über Erfolg oder Mißerfolg des Befehls ablesen.

Weiterhin werden die aktuelle Tracknummer, die Kopfadresse (wichtig bei Doppelkopf-Laufwerken), die Sektornummer und die Sektorgröße geliefert. Erst danach kann ein beliebiges neues Kommando gestartet werden.

Daten schreiben

Wie auch beim Lesen werden beim Schreiben von Sektoren 9 Bytes benötigt. Nachdem dem FDC alle Bytes übermittelt wurden, beginnt er (nach Ablauf der Head Load Time) die Suche nach dem gewünschten Sektor. Dazu liest er solange Sektor-IDs, bis er den gewünschten Sektor identifiziert hat, oder der Index-Impuls zum zweiten Mal nach Beginn der Suche auftritt. Im letzteren Fall wird das Kommando abgebrochen und die Result-Phase beginnt augenblicklich.

Phase	R/W	DATEN BUS D7 D6 D5 D4 D3 D2 D1 D0	Bemerkungen
		DATEN SCHREIBEN	
Kommando	W	MT MF 0 0 0 1 0 1	Kommando Codes
	W	X X X X X HD US1 US0	
	W	——— Spurnummer ———	Sektor ID Information vor
	W	——— Kopfadresse ———	Kommandoausführung
	W	——— Sektoradresse ———	
	W	——— Sektorgröße ———	
	W	– Letzte Sektor Nr. auf Spur–	
	W	-Lücke zwischen ID u. Daten-	
	W	Sektorlänge wenn Sektorgr.=0	
Ausführung			Daten Transfer zwischen FDD und System
Ergebnis	R	———Status 0———	Zustands Information nach
	R	———Status 1———	Kommandoausführung
	R	———Status 2———	
	R	———Spurnummer———	Sektor ID Information nach
	R	———Kopfadresse———	Kommandoausführung
	R	———Sektornummer———	
	R	———Sektorgröße———	

Abb. 19

Wird jedoch der gewünschte Sektor gefunden, dann fordert der FDC jetzt die zu schreibenden Daten vom Prozessor (im Nicht-DMA-Betrieb) an. Nachdem alle Daten geschrieben worden sind, beginnt die Result-Phase. Sie unterscheidet sich nicht von der des Kommandos Daten lesen.

Gelöschte Daten lesen

Die Bezeichnung für diesen Befehl ist etwas irreführend. Gelöschte Daten sind in diesem Zusammenhang nicht als die Daten einer gelöschten Datei zu verstehen. Von Dateien und Directorys hat der FDC bei aller Intelligenz keine Ahnung. Die Bezeichnung 'gelöscht' bezieht sich vielmehr auf die Möglichkeit, Sektoren durch den Eintrag eines gelöschten Data Adress Mark als gelöscht zu markieren. Solche Sektoren werden dann beim normalen Lesen und Schreiben ignoriert. Das Lesen gelöschter Daten geht ansonsten wie das Lesen normaler Daten vor sich.

Phase	R/W	DATEN BUS D7 D6 D5 D4 D3 D2 D1 D0	Bemerkungen
GELÖSCHTE DATEN LESEN			
Kommando	W	MT MF SK 0 1 1 0 0	Kommando Codes
	W	X X X X X HD US1 US0	
	W	——— Spurnummer ———	Sektor ID Information vor
	W	——— Kopfadresse ———	Kommandoausführung
	W	——— Sektoradresse ———	
	W	——— Sektorgröße ———	
	W	– Letzte Sektor Nr. auf Spur–	
	W	-Lücke zwischen ID u. Daten-	
	W	Sektorlänge wenn Sektorgr.=0	
Ausführung			Daten Transfer zwischen FDD und System
Ergebnis	R	———Status 0———	Zustands Information nach
	R	———Status 1———	Kommandoausführung
	R	———Status 2———	
	R	———Spurnummer———	Sektor ID Information nach
	R	———Kopfadresse———	Kommandoausführung
	R	———Sektornummer———	
	R	———Sektorgröße———	

Abb. 20

Gelöschte Daten schreiben

Dieser Befehl unterscheidet sich nicht wesentlich vom normalen Schreiben von Daten. Einzige Ausnahme ist die spezielle Behandlung der Data Adress Mark, hier wird eine gelöschte Data Adress Mark eingetragen.

Phase	R/W	DATEN BUS D7 D6 D5 D4 D3 D2 D1 D0	Bemerkungen
GELÖSCHTE DATEN SCHREIBEN			
Kommando	W	MT MF 0 0 1 0 0 1	Kommando Codes
	W	X X X X X HD US1 US0	
	W	——— Spurnummer ———	Sektor ID Information vor
	W	——— Kopfadresse ———	Kommandoausführung
	W	——— Sektoradresse ———	
	W	——— Sektorgröße ———	
	W	– Letzte Sektor Nr. auf Spur–	
	W	-Lücke zwischen ID u. Daten-	
	W	Sektorlänge wenn Sektorgr.=0	
Ausführung			Daten Transfer zwischen FDD und System
Ergebnis	R	———Status 0———	Zustands Information nach
	R	———Status 1———	Kommandoausführung
	R	———Status 2———	
	R	———Spurnummer———	Sektor ID Information nach
	R	———Kopfadresse———	Kommandoausführung
	R	———Sektornummer———	
	R	———Sektorgröße———	

Abb. 21

Lesen eines Track

Dies Kommando ist mit dem Lesen eines Sektors vergleichbar. Allerdings werden bei diesem Kommando alle Datenbytes des Tracks gelesen. Beendet wird dieses Kommando entweder, wenn der als letzter Sektor angegebene Sektor gelesen wurde, oder, falls ein solcher Sektor nicht gefunden wurde, das Index-Loch einen zweiten Impuls nach Beginn des Kommandos erzeugt.

Phase	R/W	DATEN BUS D7 D6 D5 D4 D3 D2 D1 D0	Bemerkungen
		SPUR LESEN	
Kommando	W	0 MF SK 0 0 0 1 0	Kommando Codes
	W	X X X X X HD US1 US0	
	W	——— Spurnummer ———	Sektor ID Information vor
	W	——— Kopfadresse ———	Kommandoausführung
	W	——— Sektoradresse ———	
	W	——— Sektorgröße ———	
	W	– Letzte Sektor Nr. auf Spur–	
	W	-Lücke zwischen ID u. Daten-	
	W	Sektorlänge wenn Sektorgr.=0	
Ausführung			Daten Transfer zwischen FDD und System. FDC ließt alles was sich auf der Spur zwischen Indexloch und EOT befindet.
Ergebnis	R	———Status 0———	Zustands Information nach
	R	———Status 1———	Kommandoausführung
	R	———Status 2———	
	R	———Spurnummer———	Sektor ID Information nach
	R	———Kopfadresse———	Kommandoausführung
	R	———Sektornummer———	
	R	———Sektorgröße———	

Abb. 22

Formatieren eines Tracks

Das Formatieren eines Tracks ist mit dem 765 sehr einfach. Dazu wird dem FDC in der Befehlsphase eine Kette von 6 Bytes übermittelt. Erkennt der FDC nach der vollständigen übermittlung der sechs Bytes den physikalischen Anfang des Track am Index-Impuls, so beginnt er automatisch, die Spur mit allen benötigten Adress Marks, Gaps und IDs zu formatieren.

Das Byte Nummer 4 der Befehlskette gibt an, wie viele Sektoren auf dem Track untergebracht werden sollen. Für jeden Sektor fordert der FDC vier weitere Bytes an. Eins dieser Bytes ist die Sektornummer, die in der Sektor ID vermerkt wird. Dadurch ist es möglich, die Sektoren in unterschiedlicher Reihenfolge zu formatieren. Durch diese Maßnahme können bei geschickter Wahl der Reihenfolge die späteren Lesezugriffe deutlich beschleunigt werden.

Phase	R/W	DATEN BUS D7 D6 D5 D4 D3 D2 D1 D0	Bemerkungen
		SPUR FORMATIEREN	
Kommando	W	0 MF 0 0 1 1 0 1	Kommando Codes
	W	X X X X X HD US1 US0	
	W	——— Sektorgröße ———	Bytes/Sektor
	W	——— Sektoren pro Spur ———	Sektoren/Spur
	W	-Lücke zwischen ID u. Daten-	Lücke #3
	W	—Datenmuster für Sektor—	Füllbyte
Ausführung			FDC formatiert eine gesammte Spur.
Ergebnis	R	——— Status 0 ———	Zustands Information nach
	R	——— Status 1 ———	Kommandoausführung
	R	——— Status 2 ———	
	R	——— Spurnummer ———	In diesem Fall hat die ID
	R	——— Kopfadresse ———	Information keine Bedeutung
	R	——— Sektornummer ———	
	R	——— Sektorgröße ———	

Abb. 23

ID lesen

Dieser Befehl ermöglicht es, nach Angabe von zwei Bytes in der Befehlsphase, die nächstmögliche ID von der Diskette zu lesen. Jeder Sektor hat beim Formatieren seine eigene ID erhalten. In dieser ID sind die Werte der Sektornummer, des Track, der Diskettenseite und der Sektorgröße enthalten. Diese Werte werden in der abschließenden Result Phase zusammen mit den Zuständen der drei Status-Register ST0 bis ST2 an den Prozessor übergeben.

Phase	R/W	DATEN BUS D7	D6	D5	D4	D3	D2	D1	D0	Bemerkungen
ID LESEN										
Kommando	W	0	MF	0	0	1	0	1	0	Kommando Codes
Ausführung	W	X	X	X	X	X	HD	US1	US0	Die erste richtige ID Information einer Spur in Datenregister speichern
Ergebnis	R	Status 0								Zustands Information nach Kommandoausführung
	R	Status 1								
	R	Status 2								
	R	Spurnummer								Sektor ID Information während Ausführungsphase
	R	Kopfadresse								
	R	Sektornummer								
	R	Sektorgröße								

Abb. 24

Die Scan-Kommandos, testen eines Sektors

Diese Befehle kommen in ihrer Auswirkung einem Verify, also dem Prüfen der geschriebenen mit den zu schreibenden Daten gleich. Die möglichen Testbedingungen sind 'Gleichheit', 'Größer Gleich' und 'Kleiner Gleich'. Nach dem Übermitteln von 9 Bytes in der Befehlsphase werden vom FDC Daten aus dem gewählten Sektor gelesen. Gleichzeitig fordert der FDC Daten vom Prozessor. Je ein Byte von der Diskette wird mit einem Byte vom Prozessor nach den angegebenen Testbedingungen verglichen. Beendet wird dieses Kommando, wenn entweder die Testbedingung im angegebenen Sektor erfüllt ist oder der letzte Sektor des Track geprüft wurde oder ein TC-Impuls an den Pin 16 gelegt wurde.

Phase	R/W	DATEN BUS D7 D6 D5 D4 D3 D2 D1 D0	Bemerkungen
		SCAN EQUAL	
Kommando	W	MT MF SK 1 0 0 0 1	Kommando Codes
	W	X X X X X HD US1 US0	
	W	——— Spurnummer ———	Sektor ID Information vor
	W	——— Kopfadresse ———	Kommandoausführung
	W	——— Sektoradresse ———	
	W	——— Sektorgröße ———	
	W	– Letzte Sektor Nr. auf Spur–	
	W	-Lücke zwischen ID u. Daten-	
	W	Sektorlänge wenn Sektorgr.=0	
Ausführung			Datenvergleich zwischen FDD und System
Ergebnis	R	———Status 0———	Zustands Information nach
	R	———Status 1———	Kommandoausführung
	R	———Status 2———	
	R	———Spurnummer———	Sektor ID Information nach
	R	———Kopfadresse———	Kommandoausführung
	R	———Sektornummer———	
	R	———Sektorgröße———	

Abb. 25

Phase	R/W	DATEN BUS D7	D6	D5	D4	D3	D2	D1	D0	Bemerkungen
SCAN LOW OR EQUAL										
Kommando	W	MT	MF	SK	1	1	0	0	1	Kommando Codes
	W	X	X	X	X	X	HD	US1	US0	
	W	——— Spurnummer ———								Sektor ID Information vor
	W	——— Kopfadresse ———								Kommandoausführung
	W	——— Sektoradresse ———								
	W	——— Sektorgröße ———								
	W	– Letzte Sektor Nr. auf Spur–								
	W	-Lücke zwischen ID u. Daten-								
	W	Sektorlänge wenn Sektorgr.=0								
Ausführung										Datenvergleich zwischen FDD und System
Ergebnis	R	———— Status 0 ————								Zustands Information nach
	R	———— Status 1 ————								Kommandoausführung
	R	———— Status 2 ————								
	R	———— Spurnummer ————								Sektor ID Information nach
	R	———— Kopfadresse ————								Kommandoausführung
	R	———— Sektornummer ————								
	R	———— Sektorgröße ————								

Abb. 26

Phase	R/W	DATEN BUS D7	D6	D5	D4	D3	D2	D1	D0	Bemerkungen
SCAN HIGH OR EQUAL										
Kommando	W	MT	MF	SK	1	1	1	0	1	Kommando Codes
	W	X	X	X	X	X	HD	US1	US0	
	W	——— Spurnummer ———								Sektor ID Information vor
	W	——— Kopfadresse ———								Kommandoausführung
	W	——— Sektoradresse ———								
	W	——— Sektorgröße ———								
	W	– Letzte Sektor Nr. auf Spur–								
	W	-Lücke zwischen ID u. Daten-								
	W	Sektorlänge wenn Sektorgr.=0								
Ausführung										Datenvergleich zwischen FDD und System
Ergebnis	R	———— Status 0 ————								Zustands Information nach
	R	———— Status 1 ————								Kommandoausführung
	R	———— Status 2 ————								
	R	———— Spurnummer ————								Sektor ID Information nach
	R	———— Kopfadresse ————								Kommandoausführung
	R	———— Sektornummer ————								
	R	———— Sektorgröße ————								

Abb. 27

Recalibrate, Spur Null suchen

Bei diesem Kommando wird der Kopf des ausgewählten Laufwerks so lange auf Spur Null zu bewegt, bis entweder der Track0-Impuls des Laufwerks dem FDC das Erreichen der Spur signalisiert, oder der FDC 77 Stepimpulse geliefert hat. Wie dieses Kommando beendet wurde, kann an Hand der Status-Register festgestellt werden.

Phase	R/W	DATEN BUS D7	D6	D5	D4	D3	D2	D1	D0	Bemerkungen
SPUR 0 SUCHEN										
Kommando	W	0	0	0	0	0	1	1	1	Kommando Codes
	W	X	X	X	X	X	HD	US1	US0	
Ausführung										Kopf auf Spur 0 zurücksetzen

Abb. 28

Das SEEK-Kommando, Suchen einer Spur

Floppy-Laufwerke bewegen den Schreib/Lesekopf mit Hilfe sogenannter Steppermotoren. Diese Motoren sind nicht ständig in Bewegung, sondern werden mit Impulsen zu sehr definierten Winkeländerungen der Achse bewegt. Die Erzeugung dieser Impulse geschieht in der Regel durch teilweise recht aufwendige Digital-Schaltungen in den Floppy-Laufwerken selber. Nach außen hin verfügt ein Laufwerk über zwei Anschlüsse. An einem Anschluß werden Impulse angelegt, die den Steppermotor so ansteuern, daß der Kopf bei jedem auftretenden Impuls um genau eine Spur bewegt wird. Der zweite Eingang an den Laufwerken bestimmt dazu die gewünschte Richtung.

Diese Kopfbewegung wird mit dem Seek-Kommando gesteuert. Der FDC verfügt über insgesamt vier interne Register, in denen er die momentane Kopfposition der vier möglichen Laufwerke speichert. Diese Register stehen nach dem Recalibrate-Kommando auf Null, werden also gelöscht. Wird nun zu einem

bestimmten Laufwerk ein Seek-Kommando abgeschickt, so wird der Inhalt des zugehörigen Positionsregisters mit dem geforderten Wert verglichen. Sind die beiden Werte gleich, so wird keine weitere Aktion nötig. Ergibt sich jedoch eine Differenz zwischen diesen beiden Werten, dann wird entsprechend der benötigten Richtung die Polarität des Signals DIR am Pin 38 beeinflußt und Stepimpulse am Pin 37 erzeugt. Der zeitliche Abstand zwischen den Stepimpulsen ist mit dem Kommando 'Laufwerkdaten angeben' in weiten Grenzen programmierbar.

Bei diesem Befehl und dem Recalibrate-Kommando gibt es keine Result Phase. Der Programmierer sollte nach einem Seek immer das Kommando 'Interruptstatus lesen' angeben, um das Kommando ordnungsgemäß zu beenden. Dieses Kommando liefert in der Result-Phase den Zustand des ST0, das auch Interrupt-Status genannt wird. Ohne dieses Kommando nimmt der FDC keine Schreib/Lesebefehle mehr an.

Phase	R/W	DATEN BUS D7	D6	D5	D4	D3	D2	D1	D0	Bemerkungen
Spur suchen										
Kommando	W	0	0	0	0	1	1	1	1	Kommando Codes
	W	X	X	X	X	X	HD	US1	US0	
	W	——— Spurnummer ———								
Ausführung										Der Kopf wird über die gesuchte Spur auf der Diskette positioniert

Abb. 29

Sense Interrupt Status, Status-Register 0 abfragen

Interrupts werden vom FDC im Non-DMA-Betrieb bei folgenden Ereignissen erzeugt:

- *während der Execution-Phase,*
- *zu Beginn der Result-Phase,*
- *am Ende eines Seek oder Recalibrate,*
- *bei Änderung des Ready-Signals eines der Laufwerke.*

Können die beiden ersten Interupt-Ursachen leicht vom Prozessor erkannt werden, so muß bei den beiden folgenden Interrupt-Ursachen das Kommando 'Sense Interrupt Status' ausgeführt werden, um die Ursache zu ermitteln. An Hand der Bits im gelieferten Wert des ST0 kann leicht die Interruptquelle ermittelt werden.

Phase	R/W	DATEN BUS D7 D6 D5 D4 D3 D2 D1 D0	Bemerkungen
INTERRUPTSTATUS ABFRAGEN			
Kommando	W	0 0 0 0 0 0 0 0	Kommando Code
Ergebnis	R	Status 0	Status Information am Ende der Such-Operation über FDC
	R	Spurnummer nach Suchbefehl	

Abb. 30

Sense Drive Status, Laufwerkstatus abfragen

Dieser Befehl stellt die einzige Möglichkeit dar, den Inhalt des Status-Registers ST3 zu ermitteln. Dieses Register gibt Aufschluß über den Zustand des ausgewählten Laufwerks und kann zu jeder Zeit mit dem entsprechenden Kommando ausgelesen werden.

Phase	R/W	DATEN BUS D7	D6	D5	D4	D3	D2	D1	D0	Bemerkungen
LAUFWERKSTATUS ABFRAGEN										
Kommando	W	0	0	0	0	0	1	0	0	Kommando Codes
	W	X	X	X	X	X	HD	US1	US0	
Ergebnis	R	——— Status 3 ———								Status Information über FDD

Abb. 31

Das Specify-Kommando, Laufwerkdaten angeben

Obwohl in unserer kurzen Darstellung der Befehle des FDC an letzter Stelle, sollte dieser Befehl doch der erste Befehl nach einem Reset oder nach dem Einschalten des FDC sein. Mit Hilfe dieses Befehls können die unterschiedlichsten Laufwerke an den FDC angepasst werden. Alle in Frage kommenden Zeiten werden mit diesem 3-Byte Kommando dem FDC mitgeteilt. Zusätzlich zu den Wartezeiten wird mit einem Bit entschieden, ob der FDC im DMA-Modus oder im Interrupt-Modus arbeiten soll.

Doch betrachten wir die Laufwerkdaten. Da ist zunächst die sogenannte Step Rate Time. Diese Zeit wartet der FDC selbsttätig zwischen den einzelnen Stepimpulsen. Da die unterschiedlichen Laufwerksfabrikate ganz unterschiedliche Zeiten zwischen den Impulsen erwarten, kann diese Zeit exakt den benötigten Werten angepasst werden.

Die zweite einstellbare Zeit ist die Wartezeit, die der FDC nach der Aktivierung des Head Load-Signals automatisch einlegt. Diese Head Load Time genannte Wartezeit ist nur bei 8"-Laufwerken relevant, bei kleineren Laufwerken wird der Kopf fast immer mit dem Motor On-Signal geladen.

Die dritte einstellbare Zeit ist die Head Unload Time. Diese programmierbare Zeit wird nach dem Diskettenzugriff gewartet, bis das Head Load-Signal des FDC wieder inaktiv wird. Auch diese Zeit ist nur bei Laufwerken von Bedeutung, die tatsächlich die Kopfsteuerung über den genannten Anschluß des FDC vornehmen.

Phase	R/W	DATEN BUS D7 D6 D5 D4 D3 D2 D1 D0	Bemerkungen
LAUFWERKDATEN ANGEBEN			
Kommando	W	0 0 0 0 0 0 1 1	Kommando Code
	W	— Steprate ⟶⟵ Kopf abheben	Status Information am Ende
	W	——— Kopf laden ⟶ kein DMA	der Such-Operation über FDC

Steprate bei 5¼"					
Bit	7	6	5	4	Zeit
	0	0	0	0	32 ms
	0	0	0	1	30 ms
	0	0	1	0	28 ms
			:		:
	1	1	0	1	6 ms
	1	1	1	0	4 ms
	1	1	1	1	2 ms

Kopf abheben					
Bit	3	2	1	0	Zeit
	0	0	0	0	0 ms
	0	0	0	1	32 ms
	0	0	1	0	64 ms
			:		:
	1	1	0	1	416 ms
	1	1	1	0	448 ms
	1	1	1	1	480 ms

Kopf laden								
Bit	7	6	5	4	3	2	1	Zeit
	0	0	0	0	0	0	0	4 ms
	0	0	0	0	0	0	1	8 ms
	0	0	0	0	0	1	0	12 ms
				:				:
	1	1	1	1	1	0	1	500 ms
	1	1	1	1	1	1	0	504 ms
	1	1	1	1	1	1	1	508 ms

DMA - Bit
1 = DMA - Modus
0 = Non DMA - Modus

Abb. 32

3.1.2.3 Die Statusregister des FDC 765

Wie bereits erwähnt, verfügt der 765 über 5 interne Status-Register. Das Hauptstatus-Register ist zu jedem Zeitpunkt auslesbar. Den Inhalt der Status-Register 0 bis 2 erhält man im Abschluß an alle Schreib- und Lesebefehle. Ein gezielter Zugriff auf die Register 1 und 2 ist nicht möglich. Nur der Inhalt der Status-Registers 0 und 3 kann mit entsprechenden eigenen Befehlen gezielt ausgelesen werden.

Die in der folgenden Beschreibung verwendeten Spezialbezeichnungen und Abkürzungen haben wir dem NEC-Datenblatt des 765 entnommen. Leider sind diese Bezeichnungen selbst bei NEC nicht durchgängig verwendet worden. So werden im Applikationsbericht zum 765 teilweise andere Bezeichnungen und Abkürzungen als im Datenblatt verwendet.

Bevor wir nun die Status-Register detailliert betrachten, wollen wir noch kurz einen verwendeten Begriff erklären. Die Bezeichnung Cylinder bedeutet nichts anderes als der Begriff Track oder Spur. Warum in den Datenblättern beide Begriffe parallel auftauchen, ist für uns nicht erklärbar. Wir haben versucht, den Begriff Cylinder möglichst zu vermeiden. An einigen Stellen jedoch ergäben die verwendeten Abkürzungen keinen Sinn mehr. Hier wurde der Begriff Cylinder beibehalten.

Das Hauptstatus-Register

In diesem Register werden die wichtigsten Daten über den momentanen Zustand des FDC dargestellt. Auch wird mit diesem Register das Hand-Shaking zwischen Prozessor und FDC geregelt. Die acht Bits dieses Registers geben folgende Daten und Zustände an:

Bit 7 RQM: **Request For Master**
Ist dieses Bit gesetzt, dann ist der FDC bereit, ein neues Byte über das Datenregister zu lesen oder zu schicken. Bei gelöschtem RQM kann dagegen derzeit kein Datentransfer geschehen.

Bit 6 DIO: **Data Input/Output**
Zeigt DRQ an, daß ein Datentransfer möglich ist, so signalisiert DIO die benötigte Datenrichtung. Ein gesetztes DIO bedeutet, daß der FDC ein Datenbyte für den Prozessor hat, ein gelöschtes DIO zeigt an, daß der FDC ein Byte vom Prozessor erwartet.

Bit 5 EXM: **Execution Mode**
Dieses Bit wird nur im Non-DMA-Mode verwendet. Beim DMA-Betrieb ist dieses Bit grundsätzlich gelöscht.
Im Non-DMA-Modus wird das EXM-Bit gesetzt, wenn die Ausführungsphase begonnen hat. Zu Beginn der Result-Phase wird EXM wieder gelöscht. Mit Hilfe dieses Bits kann also unterschieden werden, ob es sich bei gelieferten Werten um Sektorinformationen oder die Bytes aus der Result-Phase handelt.

Bit 4 CB: **FDC Busy**
Ein gesetztes CB-Bit signalisiert, daß der FDC derzeit einen Schreib- oder Lesebefehl bearbeitet und keine weiteren Kommandos entgegen nehmen kann. Mit dem Empfang des ersten Bytes einer Kommandokette wird dieses Bit gesetzt und bleibt bis zum Auslesen des letzten Bytes der Result-Phase gesetzt. Danach wird dieses Bit automatisch zurückgesetzt.

Bits 3-0 DB: **FDD3-0 Busy**
Diese vier Bits sind den vier möglichen Laufwerken zugeordnet. Wird auf einem dieser Laufwerke ein SEEK- oder Recalibrate-Kommando gestartet, so wird das zugehörige Bit im Hauptstatus-Register gesetzt. Sobald eins dieser Bits gesetzt ist, kann kein Schreib- oder Lesebefehl an den FDC geschickt werden. Weitere Seek- oder Recalibrate-Befehle sind jedoch bei den anderen Laufwerken möglich. Es wird dann

zu jedem weiteren Befehl das entsprechende Laufwerksbit gesetzt.
Diese Bits werden nicht automatisch am Ende des Befehls gelöscht. Um diese Bits wieder auf 0 zu setzen, muß das Kommando 'Interruptstatus lesen' an den FDC geschickt werden. Dieser Befehl löscht die Bits, wenn das entsprechende Kommando beendet war.

Das Status-Register 0

Das Status-Register 0 wird als Interrupt-Status-Register bezeichnet, da es im Non-DMA-Betrieb die Ursache eines Interrupts angibt.

Bit 7,6 IC: **Interrupt Code**
In diesen beiden Bits gibt der FDC Aufschluß über den Verlauf eines Kommandos. Vier Möglichkeiten sind mit den zwei Bits gegeben:

Bit
7 6

0 0 Kommando erfolgreich beendet. Diese Meldung sollten Sie sich immer wünschen, denn sie bedeutet, daß z.B. ein Lesezugriff erfolgreich war. Aber Achtung!

0 1 Kommando abgebrochen. In diesem Fall wurde das Kommando zwar gestartet, aber nicht erfolgreich beendet. Diese Meldung kann z.B. bei Lesefehlern auf der Diskette auftreten, wird jedoch im CPC auch nach jedem Lesen oder Schreiben eines Sektors gemeldet. Ursache ist der Verzicht auf das TC-Signal und die dadurch nötige Programmierung des letzten Tracksektors gleich dem zu lesenden Sektor. Beim Auftreten dieser Kombination ist also

nicht unbedingt von einem wirklichen Fehler auszugehen.

1 0 Ungültiges Kommando. Das von Ihnen angegebene Kommando konnte nicht gestartet werden, da es sich um ein illegales Kommando handelt.
Diese Meldung erhalten Sie auch, wenn das Kommando 'Sense Interrupt Status' abgeschickt wurde, zu dieser Zeit jedoch kein Interrupt vorliegt.

1 1 Kommando abgebrochen. Ursache für diese Meldung ist eine Änderung des Ready-Signals des gewählten Laufwerks während eines Kommandos. Dann ist zwar das Kommando begonnen, aber nicht komplett beendet worden. Diese Meldung erhalten Sie z.B., wenn Sie während eines Lesezugriffs die Diskette aus dem Laufwerk nehmen.

Bit 5 SE: **Seek End**
Sobald ein Seek-Kommando beendet wurde, setzt der FDC dies Bit auf eins.

Bit 4 EC: **Equipment Check**
Dieses Status-Bit zeigt zum einen, ob das Laufwerk einen Fehler meldet. Im Fehlerfall wird das EC-Bit gesetzt. Die zweite Ursache eines gesetzen EC-Bits ist das Ausbleiben des TRK0-Signals nach einem Recalibrate. Beim Recalibrate wird ja der Kopf so lange in Richtung auf die Spur Null zu bewegt, bis entweder der Spur-0-Sensor eine Meldung an den FDC liefert oder 77 Step-Impulse an das Laufwerk geschickt wurde. Dieser Fall kann bei 80-Track-Laufwerken dann auftreten, wenn der Schreib/Lesekopf auf den innersten Spuren 78

bis 80 steht. In diesem Fall kann das Kommando 'Recalibrate' einfach wiederholt werden.

Bit 3 NR: **Not Ready**
Wenn das ausgewählte Laufwerk bei einem Schreib- oder Lesebefehl meldet, nicht Ready zu sein, so wird dies Flag gesetzt. Auch der Zugriff auf den nicht vorhandenen zweiten Kopf eines einseitigen Laufwerks setzt dieses Bit.

Bit 2 HD: **Head Adress**
Dieses Bit gibt Aufschluß über den gewählten Kopf zum Zeitpunkt des Interruptereignisses.

Bit 1, 0 US: **Unit Select**
Diese beiden Bits zeigen an, welches Laufwerk zum Zeitpunkt des Interrupts aktiv ist.

Das Status-Register 1

Dieses Register gibt in der Result-Phase Aufschluß über den Ablauf der Ausführungsphase (Execution Phase).

Bit 7 EN: **End of Track**
Dieses Flag wird vom FDC gesetzt, wenn er einen Zugriff auf einen Sektor nach dem programmierten Ende der Spur versucht.

Bit 6 **nicht benutzt, immer Null**

Bit 5 DE: **Data Error**
Beim Schreiben von Daten wird automatisch vom FDC eine Checksumme nach dem Prinzip des 'Cyclic Redundancy Check' erzeugt und mit den Daten auf der Diskette gespeichert. Diese Checksummen werden beim Lesen der Daten ebenfalls gebildet und mit den gespeicherten Werten verglichen. Stellt der FDC eine Differenz zwischen beiden Checksummen in den

Datenfeldern oder ID-Feldern fest, so wird das DE-Flag gesetzt.

Bit 4 OR: **Over Run**
Der Datentransfer zwischen Prozessor und FDC muß beim Lesen oder Schreiben der Daten in einer bestimmten maximalen Zeit erfolgen. So werden die Daten beim Lesen in Abständen von nur 26 us an den Prozessor geliefert. Kann der Prozessor aus irgendwelchen Gründen diese Geschwindigkeit nicht einhalten, so steht unter Umständen ein neues Byte zum Lesen bereit, bevor das alte Byte vom Prozessor gelesen wurde. In einem solchen Fall spricht man von Over Run, das OR-Bit wird gesetzt.

Bit 3 **nicht benutzt, immer Null**

Bit 2 ND: **No Data**
Dieses Flag kann aus mehreren Gründen gesetzt werden.
Bei der Ausführung eines Schreib-, Lese- oder Scan-Kommandos wird dieses Flag gesetzt, wenn der Controller den angegebenen Sektor nicht finden kann.
Bei der Ausführung des Kommandos 'Sektor ID lesen' wird das ND-Flag gesetzt, wenn der Controller ein ID-Feld nicht fehlerfrei lesen kann. Auch in diesem Fall ist die Fehlerursache ein Checksum-Fehler.
Die dritte mögliche Ursache tritt im Zusammenhang mit dem Kommando 'Track lesen' auf, wenn der angegebene Startsektor auf dem Track nicht gefunden werden konnte.

Bit 1 NW: Not Writable
Wird bei der Ausführung der Kommandos 'Sektor schreiben', 'gelöschten Sektor schreiben' oder 'Track formatieren' festgestellt, daß die Diskette mit einem Schreibschutz versehen ist, so wird dies Flag gesetzt.

Bit 0 MA : Missing Adress Mark
Dieses Flag wird immer dann gesetzt, wenn der FDC beim Lesen von Daten die Sektor-ID nicht innerhalb einer vollständigen Diskettenumdrehung gefunden hat. Auch das Fehlen des Data Adress Mark oder gelöschten Data Adress Mark wird durch Setzen dieses Bits signalisiert. Zusätzlich und gleichzeitig wird noch das MD-Flag im Status-Register 2 gesetzt.

Das Status-Register 2

Ähnlich wie im ST1 werden in ST2 Hinweise auf den Erfolg oder Mißerfolg eines Kommandos gegeben.

Bit 7 **nicht benutzt, immer Null**

Bit 6 CM: **Control Mark**
Findet der FDC beim Lesen von Daten oder bei Scan-Kommandos einen Sektor mit einer gelöschten Data Adress Mark, so setzt er dieses Bit.

Bit 5 DD: **Data Error in Data Field**
Ähnlich wie beim DE-Flag (Bit 5 im ST1) wird dieses Bit bei CRC-Fehlern gesetzt. Allerdings wird dieses Bit nur bei Fehlern in Datenfeldern gesetzt.

Bit 4 WC: **Wrong Cylinder (Track)**
Beim Formatieren einer Spur muß ja zu jedem Sektor die Setornummer, die Spurnummer, die

Kopfnummer und die Sektorgröße angegeben werden. Diese Daten werden in der Sektor-ID gespeichert und bei einem Lese-Kommando mit gelesen. Stellt jetzt der FDC eine Differenz zwischen gelesener Tracknummer und angegebener Tracknummer fest, so setzt er das WC-Flag.

Bit 3 SH: **Scan Equal Hit**
Wird ein Scan-Kommando gestartet, das die Sektorinformation mit den vom Prozessor gelieferten Daten auf Gleichheit prüft, dann wird dieses Bit gesetzt, wenn die Daten tatsächlich gleich sind.

Bit 2 SN: **Scan not Satisfied**
Findet der FDC bei einem beliebigen Scan-Kommando keinen Sektor, der mit den angegebenen Daten in der geforderten Weise entspricht, so wird das SN-Bit gesetzt.

Bit 1 BC: **Bad Cylinder**
Dies Flag hat eine ähnliche Bedeutung wie das WC-Flag. Es wird dann gesetzt, wenn die aus der ID gelesene Tracknummer nicht mit der im Befehl angegebenen übereinstimmt und die aus der ID gelesene Tracknummer &FF ist.

Bit 0 MD: **Missing Adress Mark in Data Field**
Kann der FDC beim Lesen von Daten die Data Adress Mark oder gelöschte Data Adress Mark nicht finden, so wird dieses Bit gesetzt.

Das Status-Register 3, der Laufwerkstatus

Der Inhalt dieses Registers kann ausschließlich mit dem Befehl 'Laufwerkstatus bestimmen' an den Prozessor übergeben werden. Die Bits in diesem Regsiter reflektieren den Zustand des beim Kommando ausgewählten Laufwerks.

Bit 7 FT: **Fault**
Dieses Flag spiegelt den Zustand des bei einigen Laufwerken vorhandenen Fault-Signals wieder. Wenn das Laufwerk über einen solchen Anschluß verfügt, und dieses Bit gesetzt ist, dann ist ein Fehler im Laufwerk aufgetreten.

Bit 6 WP: **Write Protected**
An diesem Flag kann man sehen, ob die eingelegte Diskette schreibgeschützt ist. Ein gesetztes WP-Flag bedeutet, daß die Diskette nicht beschreibbar ist.

Bit 5 RY: **Ready**
Dieses Bit wird benutzt, um den Zustand der Ready-Leitung des Laufwerks zu bestimmen. Ein gesetztes RY-Flag signalisiert den Zustand 'Drive Ready'.

Bit 4 T0: **Track 0**
Befindet sich der Schreib/Lesekopf des ausgewählten Laufwerks zum Zeitpunkt des Kommandos auf der Spur Null, dann ist das T0-Flag gesetzt.

Bit 3 TS: **Two Side**
Doppelkopflaufwerke legen diesen Anschluß an Masse. Bei einfachen Laufwerken dagegen ist dieses Signal normalerweise High. Am Zustand des TS-Bits kann das Programm erkennen, welcher Laufwerkstyp angeschlossen ist.

Bit 2 HD: **Head Adress**
Dies Bit spiegelt den Zustand des Head Select-Signals des FDC (Pin 27) wieder.

Bit 1, 0 US: **Unit Select**
Der Zustand dieser beiden Bits ist identisch mit den Pegeln auf den beiden US-Leitungen des FDC (Pin 28 und 29)

3.1.2.4 Einsatz des FDC 765 im CPC

Leider haben die Entwickler lange nicht alle Möglichkeiten des FDC genutzt. So können nur zwei statt der möglichen vier Laufwerke angeschlossen werden. Auch der Betrieb von Doppelkopf-Laufwerken ist nicht möglich, da das HEAD-SELECT-Signal zwar herausgeführt, aber nicht benutzt wird. Schlimmer noch ist es dem Signal HEAD LOAD ergangen, es ist nirgends angeschlossen. Dieser Mangel lässt sich wohl noch am einfachsten verschmerzen, da ein Betrieb von 8"-Laufwerken nicht nur für den 'durchschnittlichen' Anwender durch die enormen physikalischen Ausmaße dieser Drives uninteressant, sondern auch durch weitere Schaltungsdetails im Controller unmöglich ist.
Trotz dieser Einschränkungen ist der Controller für den Verwendungszweck, dem problemlosen Betrieb zweier 3"-Laufwerke, sehr durchdacht konstruiert. Mit nur minimalem Hardware-Aufwand ist ein Controller geschaffen worden, der durchaus exzellente Leistungsdaten bietet.

Bei aller Sparsamkeit der Entwickler hat man zudem dankenswerter Weise nicht die Zuverlässigkeit des Gerätes eingeschränkt. So hat man dem FDC 765 einen Baustein als 'Gehilfen' verpasst, der Hardware-Experten mindestens ein anerkennendes Kopfnicken entlockt. Wir meinen den integrierten Datenseparator SMC 9229 (20-polig im CPC 464) bzw. SMC 9216 (8-polig im CPC 664 und CPC 6128). Bis auf das Signal zum Einschalten der Laufwerksmotoren werden alle Signale für das Floppy-Interface vom FDC und vom Datenseparator erzeugt.

Obwohl der DMA-Betrieb die einfachste und eleganteste Methode darstellt, den Floppy-Controller anzuschließen, hat man sich, wohl aus Kostengründen, für einen anderen Weg entschieden. Der FDC wird gepolled. Das bedeutet, daß der Prozessor an Hand des Haupt-Status-Registers den Datentransfer synchronisiert. Die vom Controller erzeugten Interrupts werden nicht benutzt. Tatsächlich ist der Interrupt-Anschluß des FDC nicht beschaltet.

Adressmäßig liegt der FDC auf den Portadressen &FB7E und &FB7F. Auf der ersten Adresse befindet sich das HauptStatus-Register, die zweite Adresse gehört zum Datenregister.

Eine dritte Adresse wird vom Controller Board belegt. Auf dem Port &FA7E befindet sich ein Flip-Flop, über das die Laufwerksmotoren gesteuert werden. Schreibt man auf diesen Port eine 1 (OUT &FA7E,1 in BASIC), so werden die Motoren aller angeschlossenen Laufwerke eingeschaltet, schreibt man dagegen eine 0, so werden die Motoren wieder ausgeschaltet.

Interessant ist die Beschaltung des Terminal CountAnschlusses und des Reset Pins. Beide Anschlüsse sind zusammengeschaltet und erhalten im Falle eines Reset gemeinsam einen positiven Impuls. Die Erzeugung eines separaten TC-Impulses ist im Controller nicht vorgesehen. Das wirft natürlich die Frage auf, wie ein Lesezugriff auf die Diskette beendet wird. (Anm.: die folgenden Ausführungen gelten nicht nur für das Lesen von Daten, sondern sind für Schreib- und Scan-Kommandos gleichermaßen zutreffend.)

Üblich ist ja das Anlegen eines TC-Impulses nach dem Lesen des gewünschten Sektors. Ohne diesen Impuls wird ein Multi Sektor I/O durchgeführt, das heißt, der Controller liest Daten, bis er den letzten Sektor der Spur erreicht hat. Da aber der letzte Sektor der Spur in den neun Bytes des Lesekommandos mit programmiert werden muß, haben sich die Entwickler eines Tricks bedient. Sie setzen einfach die Nummer des letzten Sektors gleich der Nummer des zu lesenden Sektors. Nachdem alle Daten des Sektors gelesen wurden, beendet der FDC automatisch den Lesevorgang und beginnt die Result-Phase.

Bei diesem Vorgehen ergibt sich allerdings ein Punkt, der bei der Programmierung der Controllerroutinen zu berücksichtigen ist. Wird mit diesem Verfahren ein Sektor gelesen, so meldet das Status-Register 0 einen Fehler an das Betriebssystem zurück. Der Fehler lautet 'Kommando gestartet aber nicht ordnungsgemäß beendet', Bit 6 des ST0 ist gesetzt. Die genaue Fehlerursache findet sich im ST1, hier ist das Bit 7 gesetzt. Das bedeutet im Klartext: Ende der Spur erreicht, Zugriffsversuch auf einen Sektor hinter dem Ende der Spur. Dieser Fehler muß vom Betriebssystem ignoriert werden, dabei dürfen aber keine eventuellen anderen Fehler unterschlagen werden.

Durch die vorliegende Beschaltung ist ein Betrieb von 8"-Zoll-Laufwerken nicht möglich. Aber auch die Routinen im AMSDOS sind nicht für den Betrieb von 8"-Laufwerken ausgelegt. Überhaupt ist es unter AMSDOS nicht, oder jedenfalls nur sehr schwer möglich, Laufwerke mit anderen Leistungsdaten wie z.B. 80 Track oder Doppelkopf, anzuschließen. Der Anschluß von solchen Laufwerken ist zwar grundsätzlich möglich. Das entsprechende Signal zur Kopfauswahl für Doppelkopf-Drives wird sogar zur Floppy-Schnittstelle herausgeführt. Allerdings müssen dann wesentliche Teile des DOS umgeschrieben werden, da das AMSDOS das Signal HS nicht unterstützt.

3.2 Der Aufbau der Diskette

3.2.1 Die drei Diskettenformate

Bekanntlich unterscheidet der CPC drei verschiedene Diskettenformate, das Standard-CPC-Format, das Datenformat und das IBM-Format. Welches der drei Formate beim Formatieren der Diskette benutzt wird, kann durch Angabe von S oder V (für System und Vendor), D (für Daten) oder I (für IBM) ausgewählt werden. Wir wollen jetzt einmal etwas genauer untersuchen, wie die Formate aufgebaut sind und wie sie sich unterscheiden.

Bei jedem Zugriff auf eine Diskette wird vom AMSDOS das Format selbsttätig festgestellt. Dies geschieht allerdings nur, wenn keine Datei auf der Diskette geöffnet ist. Diese Ein-

schränkung kann aber ohne weiteres hingenommen werden, da Disketten mit offenen Dateien bekanntlich nicht aus dem Laufwerk entnommen werden sollten. Das Unterscheidungskriterium, die bei jedem Format unterschiedlichen Sektornummern, haben wir ja bereits erwähnt.
Eine weitere Möglichkeit zur Unterdrückung dieses automatischen Log-Ins, wie das Bestimmen der Diskettenparameter auch genannt wird, besteht im Setzen bestimmter Speicherzellen im RAM. Diese Speicherzellen finden Sie in der Auflistung des Floppy-RAMs in einem spätern Kapitel.

Allen drei Formaten sind einige Dinge gemeinsam. Das ist zunächst einmal die Anzahl der Tracks auf der Diskette. Die Anzahl beträgt immer 40, wobei die einzelnen Tracks von 0 bis 39 durchnummeriert sind. Auch die Sektorgröße ist bei allen drei Formaten mit 512 Bytes/Sektor gleich. Außerdem können auf den Disketten in allen Formaten maximal 64 Files angelegt werden.

Damit aber enden auch bereits die Gemeinsamkeiten.

3.2.1.1 Das Standard-CPC- oder System-Format

Bei diesem Format handelt es sich wohl um das übliche und gebräuchlichste Diskettenformat des CPC. In diesem Format enthält ein Track 9 Sektoren mit den Sektornummern &41 bis &49. Außerdem sind auf der Diskette die ersten beiden Tracks für das CP/M reserviert. Wenn Sie also mit CP/M arbeiten wollen, so sollten Sie dieses Format benutzen, da nur hier ein Start oder Warm Boot (Control C) des CP/M möglich ist.

Die reservierten Tracks sind wie folgt belegt:

Track 0, Sektor &41: Boot Sektor.
Track 0, Sektor &42: Configurationssektor.
Track 0, Sektor &43 bis &47: nicht verwendet.
Track 0, Sektor &48, &49 und
Track 1, Sektor &41 bis &49: CCP und BDOS.

Wenn wir bisher von drei Formaten gesprochen haben, so ist das zwar grundsätzlich richtig, das Programm 'FORMAT.COM' kennt aber nicht nur die drei Angaben S, D ound I, sondern einen vierten Parameter, das 'V'. Mit dieser Angabe wird die Diskette zwar wie mit S formatiert, allerdings sind die System-Tracks nicht beschrieben. Diese Option ist in der Hauptsache für den Vertrieb von CP/M-Software gedacht, die aus Copyright-Gründen ohne CP/M verkauft werden muß. Der Anwender muß nach dem Erwerb die Systemspuren selbst beschreiben. Dazu kann das Programm SYSGEN.COM dienen.

3.2.1.2 Das Datenformat

Auch in diesem Format sind 9 Sektoren auf jedem der 40 Tracks formatiert. Allerdings lauten die Sektornummern &C1 bis &C9. Bei diesem Format gibt es keine reservierten Systemspuren, so daß Ihnen in diesem Format 9216 Bytes mehr auf der Diskette zur Verfügung stehen.

3.2.1.3 Das IBM-Format

Dieses Format ist identisch mit dem Format des IBM-PC unter CP/M 86. Falls Sie rein zufällig ein solches Gerät neben dem CPC als Zweit-Computer besitzen, so können Sie die Disketten des IBM auch auf Ihrem CPC lesen und schreiben. Allerdings setzt das voraus, daß beide Computer mit derselben physikalischen Floppygröße arbeiten.
Im IBM-Format ergibt sich ein etwas geringerer Speicherplatz als bei den beiden vorherigen Formaten, da nur 8 Sektoren auf einem Track formatiert sind. Diese geringere Sektorzahl wird jedoch ein wenig dadurch kompensiert, daß nur eine, die erste Spur als Systemspur reserviert ist.

3.2.2 Der Aufbau des Directory

Bei allen drei Formaten können maximal 64 Files auf einer Diskette abgespeichert werden. In diesem Abschnitt wollen wir einmal etwas näher betrachten, wo und in welcher Form die Dateinamen auf der Diskette gespeichert sind.

Die grundsätzliche Struktur des Directory ist durch das CP/M vorgegeben. Da die Disketten ja sowohl unter CP/M wie auch unter AMSDOS zu lesen und zu schreiben sind, mußte das AMSDOS an die Directory-Struktur des CP/M angepasst werden. Wenn wir also in den folgenden Ausführungen von AMSDOS sprechen, so gelten diese wegen der Kompatibilität der beiden Betriebssysteme auch für CP/M, ohne daß wir jedes Mal besonders darauf hinweisen.

Das automatische Führen eines Inhaltsverzeichnisses oder Directorys gehört zu den ganz wesentlichen Aufgaben eines jeden Disketten-Betriebssystems. Nur dank eines solchen Directorys besteht die Möglichkeit, die Daten in so kurzer Zeit auf der Diskette zu finden. Wenn jedoch nur die Namen der Dateien vermerkt werden, so bringt dies noch nicht den gewünschten schnellen Zugriff. In diesem Fall hätten Sie sich selbst um die ordnungsgemäße Verwaltung der einzelnen Sektoren auf der Diskette zu kümmern. Wie komplex eine solche Aufgabe ist, werden Sie spätestens dann einmal feststellen, wenn aus irgendwelchen Gründen das Directory Ihrer Lieblingsdiskette unlesbar geworden ist, und Sie die Daten der Diskette von Hand retten müssen.

Im Directory sollte also zusätzlich zum Namen der Datei auch die physikalische Lage der Daten auf der Diskette vermerkt sein. Genau diese beiden Daten werden in einem Directory-Eintrag einer Datei vom AMSDOS abgespeichert. Bevor wir jedoch die Art und Weise der Abspeicherung näher betrachten, wird es an dieser Stelle nötig, einige im folgenden häufig benutzte Begriffe zu erklären.

Da gibt es zunächst einmal den Begriff SEKTOR. Ein Sektor ist der Bereich, der beim Formatieren der Diskette für die Daten

angelegt wird. Im AMSDOS sind Sektoren immer 512 Bytes groß, andere Betriebssystem verwenden z.B. Sektoren mit 128, 256 oder sogar 1024 Bytes.

Sollen Daten von der Diskette gelesen werden, so muß immer ein ganzer Sektor gelesen werden. Es ist nicht möglich, ganz gezielt nur bestimmte Bytes direkt von der Diskette zu lesen. Der Sektor ist also der Datenbereich, der auf der untersten Ebene angesprochen werden kann.

Ein RECORD ist ein kleinerer Datenblock von genau 128 Bytes. Jeder Sektor (des AMSDOS) enthält demnach genau 4 Records. Wozu diese Unterteilung? Nun, dies hat seinen Grund in der Entstehungsgeschichte des CP/M. Ursprünglich wurde CP/M für Computer mit 8"-Laufwerken entwickelt. Auf diesen Laufwerken war zu der Zeit der Entwicklung ein Sektor auch immer 128 Bytes groß. Erst später wurden Formate entwickelt, bei denen die Sektoren größer als 128 Bytes wurden. Um die Kompatibilität zum vorher benutzten Format zu wahren, wurde der größere Sektor einfach programmtechnisch vom BIOS in kleinere Einheiten zu 128 Bytes unterteilt. Damit war die Kompatibilität gewahrt. Auch AMSDOS arbeitet logisch auf der Record-Ebene. Das bedeutet, daß AMSDOS und CP/M eigentlich gar keine Sektoren kennt.

Ein dritter Begriff bedarf der Erklärung. Es ist der Begriff BLOCK. Auch dieser Begriff entstammt wieder der Urzeit des CP/M. Wenn man Dateien von einigen 10K auf Diskette speichern will, so muß man sich eine Unmenge von der Datei belegter Sektoren merken. Diese Anzahl lässt sich jedoch drastisch reduzieren, wenn man mehrere Records zu Blocks zusammenfasst und sich die Blocknummern merkt. Die Größe der Blocks lässt sich im CP/M frei definieren, üblich sind Werte von 1K (so bei AMSDOS) oder 2K. Um die dafür benötigte Rechnerei zu bewältigen, werden einfach alle freien, d.h. nicht von den Systemspuren belegte Records der Reihe nach von den unteren Spuren beginnend durchnummeriert.
In der Praxis sieht es bei einer Diskette im S-Format so aus, daß der Sektor &41 der Spur 2 die Records 0 bis 3 der Diskette enthält. Auf dem Sektor &42 befinden sich die Records 4 bis 7

u.s.w. Da nun, wie gesagt, ein Block unter AMSDOS eine Größe von 1K hat, enthält er 8 Records. Somit ist der Block 0 auf den Sektoren &41 und &42 des Track 2 zu finden, der Block 1 belegt die Sektoren &43 und &44 und z.B. der Block 4 belegt von Track 2 den Sektor &49 sowie den Sektor &41 des Track 3.
Zugegeben, diese Rechnerei ist reichlich verwirrend, aber unter CP/M und AMSDOS einfach unerläßlich. Doch kehren wir jetzt zu unserem Directory zurück.

Haben Sie sich auch schon einmal gefragt, warum selbst das kleinste Programm auf der Diskette immer 1K beansprucht, selbst wenn es nur ein Byte groß ist und somit nicht einmal einen Record oder Sektor füllt? Die Ursache haben wir gerade kennengelernt. Es sind die von der Datei belegten Blocknummern, die im Directory vermerkt werden.

Sehen wir uns so einen Directory-Eintrag einmal genauer an. Für jeden Eintrag werden 32 Bytes benötigt. Beginnen wir mit den ersten 16 Bytes.

```
00 46 4F 52 4D 41 54 20 20 43 4F 4D 00 00 00 15 .FORMAT  COM....
```

Es handelt sich bei dem DIR-Eintrag um den Eintrag des Programms FORMAT.COM. Soviel läßt sich auch ohne detailierte Kenntnis des AMSDOS sagen. Der Punkt zwischen Filename und Extension wird scheinbar nicht mit abgespeichert, da der Filename mit zwei Leerzeichen (ASCII-Wert 32 oder hex 20) aufgefüllt ist. Was aber sagt der Wert 0 vor dem Filenamen aus. Erinnern Sie sich bitte an die Usernummer, die zu jeder Datei vergeben werden kann. Diese Nummer wird im ersten Byte des Eintrags vermerkt und regelt den Zugriff auf die Datei. Außerdem hat dieser Wert noch eine besondere Bedeutung, die wir etwas später kennenlernen werden.
Auf den Filenamen folgen drei weitere Null-Bytes und ein Byte mit dem Wert &15. Deren Bedeutung kann man nicht so ohne weiteres herausbekommen. Wichtig sind nur die Bytes 12 und 15, deren Bedeutung wir jedoch an dieser Stelle noch nicht verraten wollen. Betrachten wir zunächst die noch fehlenden 16 Bytes des Eintrags.

```
55 56 57 00 00 00 00 00 00 00 00 00 00 00 00 00 UVW.............
```

Auch die Bedeutung dieser Bytes kann nicht so ohne weiteres erraten werden. Allerdings würde es uns bei der Lösung des Problems ein wenig helfen, einen CAT-Befehl mit eingelegter CP/M-Master-Diskette zu versuchen. Im Catalog wird das File FORMAT.COM mit einer Größe von 3K geführt. Da 1K einem Block entspricht, und drei der gezeigten 16 Bytes einen anderen Wert als 0 enthalten, liegt die Vermutung nah, daß es sich bei den Nummern um die Blocknummern handelt. Mit dieser Vermutung liegen wir genau richtig. Tatsächlich wird die Belegung der Blocks einer Datei in den Bytes 16 bis 31 eines Directory-Eintrags vermerkt.

Das wirft einen ganzen Haufen neuer Fragen auf. Die wichtigste Frage muß lauten: Was geschieht, wenn eine Datei größer als 16K wird? Nun, die Antwort ist recht einfach, AMSDOS spendiert solch großen Dateien einfach einen weiteren Dateieintrag. Ein solcher Extent genannter Erweiterungseintrag unterscheidet sich nur in wenigen Punkten vom ersten Eintrag. In der Hauptsache wird in den Bytes 16 bis 31 die weitere Blockbelegung vermerkt. Usernummer und Filename sind in beiden Einträgen gleich.

Die nächste Frage, die sich stellt, lautet: Woran erkennt das AMSDOS, daß ein Extent folgen muß? Dazu müssen wir jetzt die Bedeutung des Byte 15 eines Eintrags kennen. Ein kleines Rechenexempel ist nötig.

Unser Beispiel FORMAT.COM besteht aus 3 Blocks. Wie wir gesagt haben, beinhaltet ein Block 8 Records. Damit kann die Datei FORMAT.COM maximal 24 Records (hex 18) groß sein. Um es kurz zu machen, die 15 ist die Anzahl der Records der Datei FORMAT.COM in hexadezimal.
Wird eine Datei so groß, daß sie mit einem Eintrag nicht auskommt, so errechnet sich der Wert in Byte 15 des Eintrags aus 16 (Blocks) * 8 (Records) = 128 oder &80. Sobald dieser Wert an der Stelle eingetragen ist, geht das AMSDOS ganz automatisch davon aus, daß noch ein Extent folgt.

Um nun nicht bei ganz großen Dateien mit mehreren Extents durcheinander zukommen, werden die Extents im Byte 12, also unmittelbar hinter dem Filenamen aufsteigend nummeriert. Damit liegt die Reihenfolge fest, in der die Extents zu lesen sind. Durch diese Organisation kann ein File entweder so groß werden, bis im Directory kein Platz für weitere Einträge mehr ist, oder bis die Diskette voll ist.

Doch kommen wir noch einmal kurz auf das erste Byte des Eintrags zu sprechen. Wie bereits erwähnt, wird in diesem Byte die User-Nummer der Datei vermerkt.
Wenn Sie das Programm zum Lesen von beliebigen Sektoren oder auch den Disk-Monitor bereits ausprobiert und die Sektoren des Directory einiger Ihrer Disketten damit inspiziert haben, so werden Sie möglicherweise bei einigen Files an Stelle einer gültigen Usernummer (0 bis 15) hier den Wert &E5 gefunden haben. Die zugehörigen Filenamen tauchen aber im Directory beim CAT-Befehl nicht auf, kein Wunder, Sie haben sie höchst wahrscheinlich mit Hilfe des ERA-Kommandos selbst gelöscht.
AMSDOS ist nun beim Löschen von Dateien sehr vorsichtig. Es wird nur der Name der angegebenen Datei mit allen eventuell vorhandenen Extents gesucht und die Usernummer auf den Wert &E5 gesetzt. Fortan ist diese Datei für AMSDOS nicht mehr existent, auch wenn noch alle Daten auf der Diskette vorhanden sind. Diese vorsichtige Vorgehensweise des AMSDOS ist sehr angenehm, wenn Sie aus Versehen wichtige Dateien gelöscht haben. Lesen Sie in solchen Fällen einfach die Sektoren des Directory mit einem Disk-Monitor und setzen die Usernummern der gelöschten Files wieder auf einen vernünftigen Wert. Nach dem Zurückschreiben sind die Dateien wiederbelebt. Übrigens ist der Wert &E5 als GelöschtMarkierung nicht zufällig gewählt worden. Nach dem Formatieren sind alle Sektoren mit dem Wert &E5 beschrieben, also auch die DIR-Sektoren. Wird jetzt ein Zugriff auf das Directory vorgenommen, so findet das AMSDOS für jeden möglichen Eintrag das Gelöscht-Kennzeichen, ohne daß dazu spezielle Vorgänge nötig wären.

Das bringt uns zum letzten wichtigen Punkt des Directory, der Lage auf der Diskette. Wie wir bereits erwähnt haben, können maximal 64 Files im Directory eingetragen werden. Bei einem

Bedarf von 32 Bytes pro Eintrag ergibt sich ein Speicherbedarf von 2048 Bytes. Das entspricht 2 Blocks oder 4 Sektoren oder auch 16 Records. Grundsätzlich werden bei jedem der drei Formate die ersten 2 Blocks der ersten freien Spur benutzt. Im S-Format finden Sie also das Directory auf den Sektoren &41 bis &45 des Track 2, im D-Format auf den Sektoren &C1 bis &C5 des Track 0 und im I-Format auf den Sektoren &01 bis &05 des Track 1.

Zum guten Schluß des Kapitels über das Directory noch zwei Tips.

Wird das Bit 7 des ersten Zeichens einer Extension, also des neunten Bytes des Eintrags, gesetzt, dann erhält das File das Attribut READ-ONLY. Derart markierte Files können unter AMSDOS und CP/M weder gelöscht noch umbenannt werden. Zum Setzen des Bits gibt es (mindestens) zwei verschiedene Wege. Die erste Möglichkeit bietet das transiente CP/M-Kommando STAT. Die zweite Möglichkeit bietet der Disk-Monitor. Haben Sie den zu schützenden Eintrag in den Directory-Sektoren gefunden, so addieren Sie zum ASCII-Wert des ersten Zeichens den Wert 128 oder &80, modifizieren das Byte entsprechend dem Ergebnis und schreiben den Sektor einfach zurück.

Mit dem zweiten Tip können Sie eine Datei vor unbefugten Dazu müssen Sie, ähnlich wie beim READ-ONLY-Attribut, das Bit 7 des zweiten Zeichens der Extension setzen. Auch diese Aufgabe kann unter CP/M mittels STAT oder unter AMSDOS mit unserem kleinen Disk-Monitor geschehen. Vergessen Sie aber nicht den Namen der solchermaßen geschützten Datei. Mit CAT oder DIR bekommen Sie ihn nicht heraus.

3.2.3 Der Aufbau der Dateien

Nachdem wir im vorigen Kapitel einen Überblick über die im Directory gespeicherten Informationen gewonnen haben, wollen wir jetzt einmal sehen, wie die eigentlichen Daten auf der Diskette stehen.

Zunächst stellt sich die Frage, woher beim Lesen einer Datei die Informationen wie Filetyp, Länge des Files oder Startadresse des geladenen Programms geholt werden. Diese Header-Informationen sind ja nicht im Directory-Eintrag enthalten. Also können sie nur zusammen mit den Daten abgespeichert werden.

Das wirft aber das Problem auf, in welcher Form diese zusätzlichen Daten gespeichert werden sollen, ohne daß es ein Durcheinander zwischen Header und Daten gibt. Wir wollen einmal einige Versuche durchführen, die Aufschluß über die Art der Datenspeicherung geben.

Geben Sie doch einmal die Zeile:

```
SAVE "X1.BIN",B,&1000,1024
```

auf Ihrem CPC ein und betätigen Sie die ENTER-Taste. Der Inhalt der Datei wird uns im Moment nicht beschäftigen, daher können Sie als Startwert jede beliebige Adresse wählen. Wichtig ist jedoch die Länge des Speicherbereichs. Wie wir zuvor gesehen haben, werden Dateien unter AMSDOS immer in 1K-Blöcken gespeichert. Da wir als Länge genau 1K gewählt haben, sollte im Directory das File die Länge von 1K anzeigen. Denkste! Die Datei wird 2K groß.

Geben Sie die Zeile jetzt noch einmal ein, verringern Sie jedoch den Wert der Dateilänge um 100 auf 924. Zur allgemeinen Überraschung ist die Datei scheinbar immer noch nicht kleiner geworden, es werden weiterhin 2K angezeigt.
Machen wir einen dritten Versuch. Als Länge geben Sie den Ausdruck 1024-128 an. Und siehe da, es ist vollbracht. Unsere Datei X1.BIN belegt nur noch 1K auf der Diskette. Jede auch nur ein Byte größere Längenangabe führt aber wieder zur Vergrößerung der Datei auf 2K.

Ein Record oder anders ausgedrückt 128 Bytes werden also zusätzlich abgespeichert. In diesem Record befinden sich alle Header-Informationen. Der Header-Record ist der erste Record fast jeder Datei. Warum die Einschränkung? Probieren Sie doch einmal das folgende kleine Programm:

```
10 OPENOUT "X1.DAT"
20 FOR i=1 TO 1024
30 PRINT#9,"x";
40 NEXT i
50 CLOSEOUT
```

Nach den bisherigen Erfahrungen könnte man vermuten, daß die Datei X1.DAT im Directory mit 2K erscheint. Dem ist aber nicht so. Tatsächlich ist die Datei nur 1K groß, genau die Größe, die auch durch die Anzahl der in die Datei geschriebenen Zeichen vorgegeben wurde.

Daraus ergibt sich folgende Regel:

> Alle reinen ASCII-Dateien, die mit dem File-Typ-Kennzeichen &16 abgespeichert wurden, haben keinen HeaderRecord. (Anm.: Es wird im Betriebssystem nur das LowNibble ausgewertet, d.h. &16 und &36 werden gleichermaßen als reine ASCII-Dateien behandelt)
>
> Bei allen anderen Dateien, das sind z.B. BASIC- oder Maschinensprache-Programme, ist der erste Record der Header-Record, der beim Laden in den Header-Bereich kopiert wird.

So weit, so gut. Wie aber soll ein Programm feststellen, ob es sich bei der Datei um eine reine ASCII-Datei handelt. Schließlich gibt es im Directory keinen Hinweis auf den Filetypen. Dieses Problem hat die Programmierer des AMSDOS sicher einiges Kopfzerbrechen gekostet. Letztendlich hat man sich für einen Weg entschieden, der einigermaßen passabel erscheint.

Grundsätzlich wird beim Eröffnen einer Input-Datei eine 16-Bit Prüfsumme über die ersten 66 Bytes des ersten Records gebildet. Das Ergebnis wird mit dem Inhalt der Bytes 67 und 68 des Records verglichen. Sind die Werte gleich, so handelt es sich mit ziemlicher Sicherheit bei diesem Record um einen Header-Record. Sollte jedoch zufällig in einer ASCII-Datei diese Prüfbedingung auch erfüllt werden, so ist der erste Record der Datei verloren. Diese Möglichkeit ist aber verschwindend gering und wird nicht weiter abgeprüft.

3.2.4 Die pysikalische Aufzeichnung der Daten auf der Diskette

Gehören Sie zu den Menschen, die den Dingen gern bis auf den Grund gehen? Dann ran an die folgenden Seiten! Wir wollen Ihnen zeigen, wie die Daten tatsächlich auf der Diskette stehen. Dabei verstehen wir in diesem Kapitel unter Daten nicht den Inhalt von irgendwelchen Dateien, sondern alles, was in irgend einer Form auf einer Spur untergebracht ist. In diesem Zusammenhang werden dann auch Begriffe wie Sektor-ID, Gap oder Adress Mark geklärt.

3.2.4.1 MF, MFM, Bits und Magnetismus

Wie Sie ja bereits wissen, ist die Oberfläche der Diskette in 40 Abschnitte oder Spuren unterteilt. Der Schreib/Lesekopf der Floppystation kann mit Hilfe eines Steppermotors über jeder dieser Spuren gezielt platziert werden. Ähnlich wie beim Recorder werden die Daten Bitweise auf die Diskette gespeichert. Die bitweise Speicherung der Daten und die Verwendung magnetischer Datenträger stellen aber auch schon die gesamten Gemeinsamkeiten zwischen den beiden Systemen dar.
Bei der Floppy wird die magnetische Schicht der Diskette voll durchmagnetisiert. Beim Cassettenrecorder hätte man die schönste mögliche Übersteuerung mit einem Klirrfaktor von 100 Prozent. Dieser 'traumhafte' Klirrfaktor stört jedoch bei der Floppy überhaupt nicht. Im Gegenteil, je besser die Durchmag-

netisierung beim Schreiben klappt, um so einfacher wird das spätere Lesen der Daten.

Um das im CPC verwendete Aufzeichnungsformat zu verstehen, müssen wir einen kleinen Abstecher in die Physik unternehmen. Der Schreib/Lesekopf eines Floppy-Laufwerks besteht in der Hauptsache aus einer Spule. Spulen gehören zu den interresantesten Bauteilen der Elektronik, auch wenn sie leider oft verkannt werden. Zu den ganz wichtigen Eigenschaften von Spulen gehört die Tatsache, daß ein Magnetfeld in einer Spule eine Spannung erzeugt. Allerdings ist es sehr wichtig, daß sich das Magnetfeld dauernd ändert, um eine kontinuierliche Spannung in der Spule zu erzeugen. Ein statisches, sich also nicht wechselndes Magnetfeld dagegen bewirkt keine Spannung. Dieses Prinzip findet man z.B. im Fahrrad-Dynamo.

Interessanterweise ist diese Eigenschaft auch umkehrbar. Eine an der Spule angelegte Spannung erzeugt ein Magnetfeld. Wird eine Wechselspannung angelegt, dann entsteht ein sich im selben Rythmus wechselndes Magnetfeld.

Beide Eigenschaften der Spule werden im Schreib/Lesekopf des Floppy-Laufwerks angewendet. Soll eine Information auf die Diskette geschrieben werden, dann wird eine Wechselspannung an den Kopf gelegt, die ein Magnetfeld erzeugt. Dieses Magnetfeld magnetisiert die Beschichtung der Diskette, es wird gespeichert. Im umgekehrten Fall wird durch das sich wechselnde Magnetfeld auf der Diskette in der Spule eine Spannung erzeugt, die von einer geeigneten Elektronik verstärkt und aufbereitet wird.

Jetzt könnte man auf die Idee verfallen, ein Lowbit durch keine Magnetisierung, ein Highbit dagegen durch sich ständig wechselnde Magnetfelder zu kennzeichnen. Damit erhielte man beim Lesen direkt die digitalen Signale zurück. Ganz so einfach geht es aber nicht. Die dafür benötigte Elektronik und Laufwerksmechanik müßte extrem präzise arbeiten, da bei mehreren aufeinander folgenden Low oder High-Bits exakt die Zeiten für Low und High gemessen und ausgewertet werden müßten.

Einfacher wird die Angelegenheit, wenn zusätzlich zu den Daten ein fester Bezugstakt mit aufgezeichnet wird. Durch diesen Takt ist dann sehr einfach die Länge einer Bitzelle zu bestimmen. Aber auch das Aufzeichen sich schnell ändernder Magnetfelder im Fall eines High-Bits ist in dieser Form nicht zu gebrauchen. Statt dessen wird einfach folgendes gemacht: soll ein High-Bit gespeichert werden, so erzeugt man einen Impuls genau zwischen zwei Taktimpulsen, bei einer Null steht dort einfach nichts. Wie eine solche Impulskette aussehen kann, zeigt das Diagramm in Abb. 32.

Mit dieser Impulskette wird ein Flip-Flop angesteuert, das mit jedem Eingangsimpuls den Zustand seines Ausgangs umkehrt. Die resultierende Ausgangsspannung sehen Sie im Diagramm Abb. 33.

Dieses resultiernde Ausgangssignal eignet sich vorzüglich zur Ansteuerung des Schreib/Lesekopfes. Ist die Ausgangsspannung des Flip-Flops High, so fließt ein Strom in der einen Richtung durch die Spule, die Magnetschicht der Diskette wird in einer Richtung magetisiert. Ändert sich aber die Ausgangsspannung des Flip-Flops, so kehrt sich auch die Stromrichtung in der Spule um, was auch eine Umkehrung der Magnetisierungsrichtung auf der Diskette bewirkt.

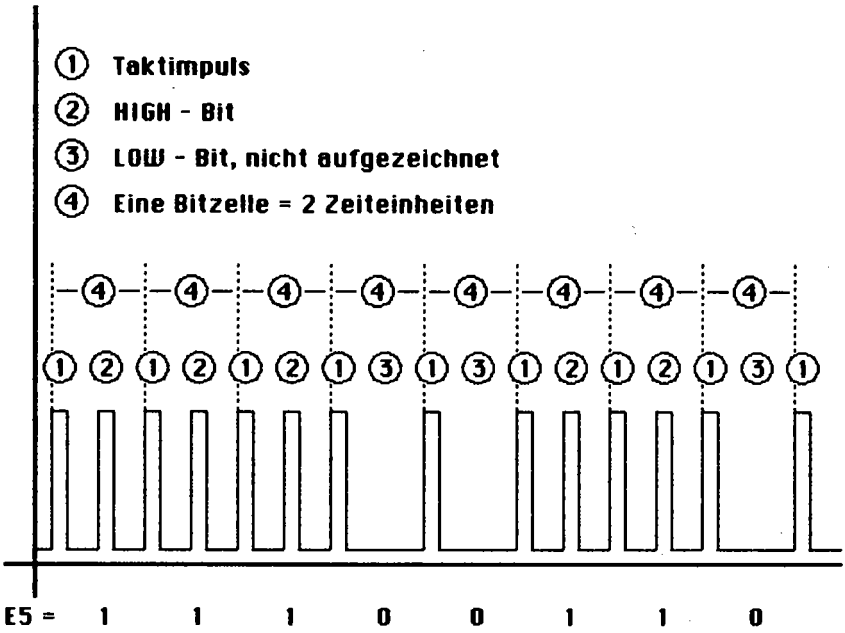


Abb. 32

Signalverlauf am R/W - Kopf beim Schreiben (Single Density)

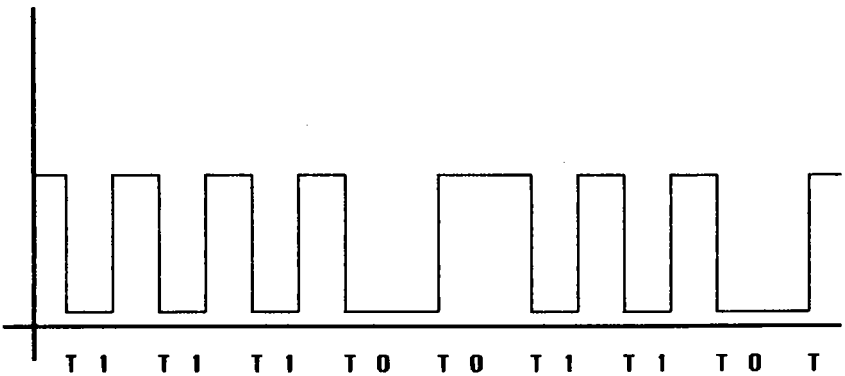


Abb. 33

Werden so Daten, also Low-High-Unterschiede aufgezeichnet, so befinden sich auf der Diskette anschließend viel kleine Magnete, die, je nach Aufzeichnung, in die eine oder andere Richtung magnetisiert sind. Darstellung Abb. 34 soll diese Aneinanderreihung der Magnete einmal grafisch verdeutlichen.

Was aber geschieht beim Lesen der Daten? Wir haben ja gesagt, daß die Diskette voll durchmagnetisiert wird. Durch die Drehung der Diskette werden die vielen kleinen Magnete am Lesekopf vorbeigeführt. Bei jeder Änderung des Magnetfeldes entsteht in der Spule ein kleiner Impuls, der sich verstärken und aufbereiten läßt. In der verbleibenden Zeit ändert sich das Magnetfeld praktisch nicht, es wird auch keine Spannung erzeugt. Das verstärkte Ausgangssignal soll einmal im Diagramm Abb. 35 dargestellt sein. Die Ähnlichkeit mit dem ersten Diagramm ist nicht zufällig. Wir haben unsere Information vollständig wieder von der Diskette gelesen.

Die Auswerte-Elektronik hat es bei diesem Verfahren mit nur zwei verschiedenen Zeiten zu tun. Die erste, längere Zeit ist die Zeit zwischen zwei Taktimpulsen, die immer bei Low-Bits auftritt, die zweite kürzere Zeit spezifiziert den Abstand zwischen Takt- und Datenimpuls bei High-Bits. Da die Zeiten relativ genau eingehalten werden, können mit Hilfe einfacher Mono-Flop-Schaltungen die Daten- von den TaktImpulsen getrennt werden, da zur Synchronisation immer ein Takt zur Verfügung steht.

Noch ein Wort zu den auftretenden Zeiten: ein Taktimpuls ist genau 500 Nanosekunden (eine halbe Millionstel Sekunde, mit Armbanduhren nicht immer zuverlässig messbar), der Abstand zwischen zwei Taktimpulsen 8 Microsekunden lang. Ein Byte benötigt also bei dem beschriebenen Verfahren 8 * 8 = 64 Microsekunden zum Schreiben oder Lesen. Bei einer nominalen Umdrehungszeit der Diskette von 200 Millisekundn passen somit 40 (Tracks) * 200 (Millisekunden)/64 (Microsekunden /Byte) = 125000 Bytes auf die Diskette.

Auf Ihren Disketten haben Sie etwa 40000 bis 50000 Bytes mehr zur Verfügung? Richtig, wir haben Ihnen zunächst das Single

Density- oder MF-Format vorgestellt. Aufbauend auf dem beschriebenen Format wollen wir jetzt das beim CPC verwendete MFM- oder Double Density-Format beschreiben.

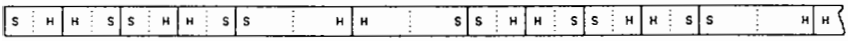


Abb. 34

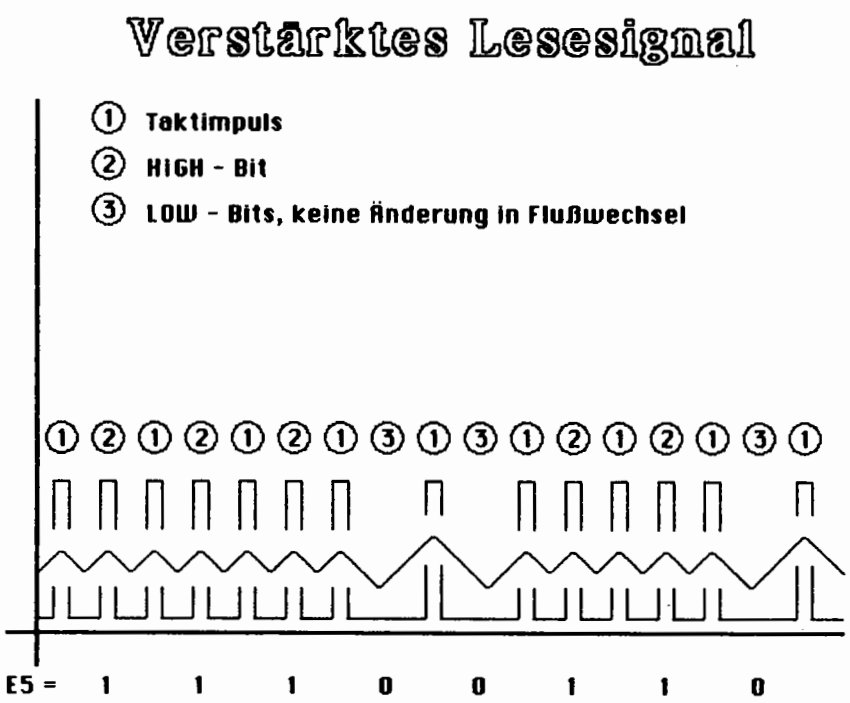


Abb. 35

Einige kluge Köpfe waren mit dem zuvor beschriebenen Verfahren nicht zufrieden. Es wurde weiter getüfftelt, bis man ein allerdings etwas aufwendigeres Verfahren zur annähernden Verdopplung der Speicherkapazität entwickelt hatte. Dabei ging man von der Idee aus, daß es möglich sein müsste, nach Möglichkeit auf die Taktimpulse zu verzichten, auch wenn das nicht vollständig möglich ist. Bei der Aufzeichnung von Hi-Bits ergeben sich keine besonderen Schwierigkeiten, da jedes aufzuzeichnende Eins-Bit einen Flußwechsel auf der Diskette bewirkt. Leider kommen jedoch auch Null-Bits vor, die aufgezeichnet werden müssen. Wenn dann noch mehrere Null-Bits aufeinander folgen, so ist die Synchronisation sehr schnell zum Teufel.
Als Ausweg werden bei mehreren aufeinander folgenden Nullen einfach Takt-Bits aufgezeichnet. Den Unterschied zwischen den Daten- und Taktbits machen wieder die Zeiten, allerdings muß jetzt zwischen drei verschiedenen Zeiten unterschieden werden. Die möglichen Zeiten finden Sie in den Darstellungen 36 und 37 grafisch dargestellt, wobei das erste Diagramm die Eingangsinformation, die zweite Grafik die Aufzeichnung auf der Diskette darstellt.
Ein einfacher Zeitabschnitt ist die Zeit zwischen zwei aufeinander folgenden Einsen, ein doppelt so langer Zeitabschnitt kennzeichnet die Bitfolge 1-0-1. Folgen mehrer Nullen aufeinander, so wird der dritte mögliche Zeitabschnitt benötigt. Diese eineinhalbfache Zeit signalisiert einen Übergang von Takt- zu Datenimpuls oder umgekehrt.

Leider kommt in den Grafiken der unmittelbare Vorteil des MFM-Verfahrens durch den gewählten Maßstab nicht so sehr gut zum Vorschein. Halten Sie sich jedoch vor Augen, daß die Zeiteinteilung in den beiden Darstellungen nicht gleich sind. Wenn man sich vergegenwärtigt, daß bei diesem Verfahren die physikalische Aufzeichnung mit der selben möglichen Anzahl von Flußwechseln, also Änderungen des Magnetfelds arbeitet, so ergibt sich eine effektive Verdopplung der Diskettenkapazität.
Es ist klar, daß bei diesem Verfahren die Auswerte-Elektronik um einiges präziser sein muß als beim einfacheren MF-Format. Im CPC wird jedoch die Auswertung in äußerst zuverlässiger Weise vom schon erwähnten Datenseparator vorgenommen.

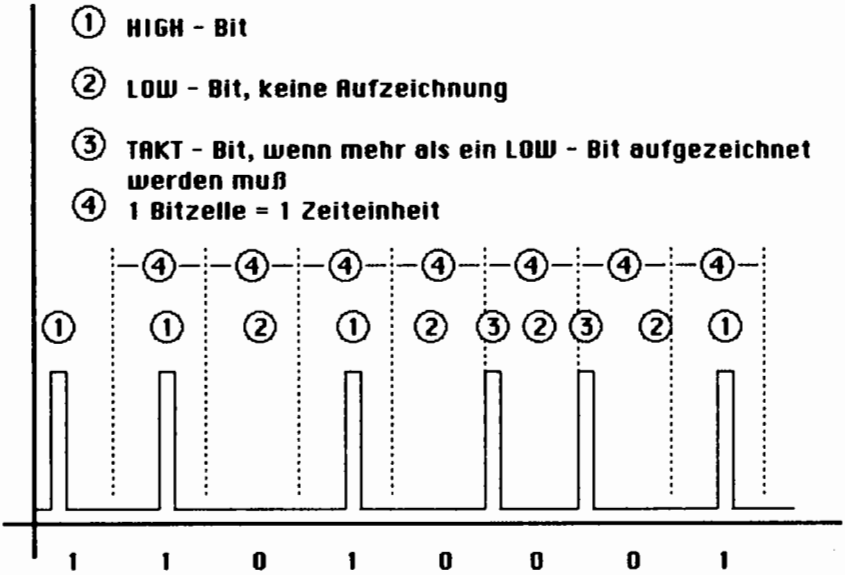


Abb. 36

Signalverlauf am R/W - Kopf beim Schreiben (Double Density)

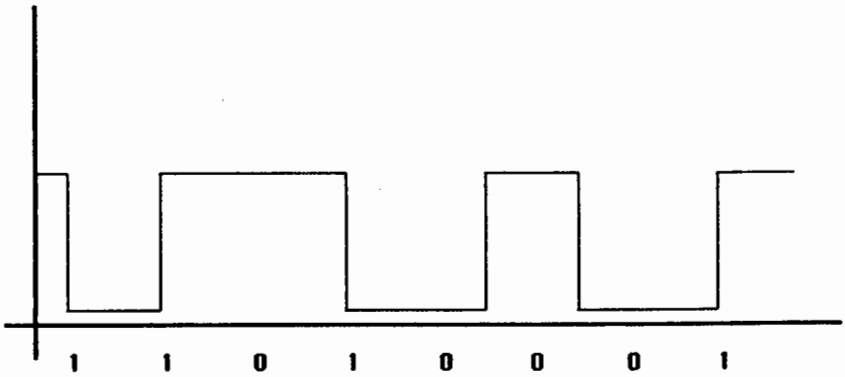


Abb. 37

Eine Frage aber bleibt noch offen. Da eine im MF-Format formatierte Diskette eine Kapazität von 125000 Bytes hat, sollte eine im MFM- oder Double Density-Format beschriebene Diskette eigentlich 250000 Bytes speichern können. Es werden aber nur etwa 182000 Bytes als Kapazität angezeigt. Wo bitte bleibt der REST?

3.2.4.2 Von GAPs, IDs, und Adress-Marks

Im vorigen Abschnitt haben wir die Diskette zunächst in einzelne Spuren oder Tracks unterteilt. Das ist durch die Verwendung eines Schrittmotors zur Kopfsteuerung nötig gewesen. In diesem Abschnitt wollen wir die Diskette wie eine Torte, allerdings ohne Messer, in einzelne Abschnitte, die Sektoren, aufteilen.

Würden wir die Unterteilung nicht vornehmen, so könnten wir auf einem Track kaum mehr als eine Datei ablegen. Dabei wäre aber unabhängig von dem tatsächlich benötigten Platz der Datei immer der ganze Track mit immerhin 6250 Bytes belegt. Das aber ist gerade bei sehr kleinen Dateien kein ausgeprochen ökonomisches Verfahren. Beim Arbeiten mit sequentiellen Dateien müsste zudem mindestens ein Buffer mit 6250 Bytes im Speicher des Rechners angelegt werden. Soll gleichzeitig aus einer Datei gelesen und in eine andere Datei geschrieben werden, so erhöht sich der Speicherbedarf auf stolze 12500 Bytes. Das ist selbst für Rechner mit 64K ein nicht unerheblicher Bedarf.

Wie man sieht, ist die Idee der Unterteilung in kleinere Bereiche recht sinnvoll. Um aber diese Bereiche deutlich voneinander zu trennen, muß ein gewisser Aufwand betrieben werden. Dieser Aufwand benötigt einigen Platz auf der Diskette, so daß nicht mehr die ursprüngliche Kapazität zur Datenspeicherung zur Verfügung steht. Man spricht bei der ersten Angabe von der unformatierten Kapazität, die effektiv verfügbare Kapazität wird mit formatierter Kapazität bezeichnet.

Diese Formatierung und die daraus resultierende Kapazität wollen wir etwas genauer betrachten. Die beschriebenen Vorgänge

finden Sie in der Abbildung 38 dargestellt. Die Grafik soll den kompletten Inhalt eines Tracks darstellen.

Die Erkennung des pysikalischen Anfangs einer Spur bereitet dem FDC durch das Indexloch in der Diskette und einer entsprechend justierten Lichtschranke keine besondere Schwierigkeit. Die Index-Lichtschranke erzeugt einen Impuls, wenn durch das Index-Loch der Lichtstrahl auf den Empfänger gelangt. Das Ende des Impulses ist für den FDC das Signal unmittelbar mit der Formatierung zu beginnen. Seine erste Tat besteht darin, 80 Bytes mit dem Wert &4E auf die Diskette zu schreiben. Dieser Bereich wird GAP 4A genannt.

Ein Gap ist wörtlich übersetzt ein Lücke. Wozu dient diese Lücke? Die Antwort ist recht einleuchtend. Die Lücke gleicht Toleranzen zwischen verschiedenen Laufwerken aus. Trotz der unbestreitbar hohen Präzision der Drives ergeben sich doch geringe Unterschiede bei der Justage der Index-Lichtschranken verschiedener Drives. Das GAP 4A ist nun so groß gewählt, daß diese Justage-Unterschiede in gewissen Grenzen ohne Bedeutung sind. Dadurch können die Datenträger, also die Disketten ohne Schwierigkeiten auf unterschiedlichen Laufwerken gelesen und beschrieben werden.

Nach dem GAP 4A wird ein Sync-Bereich von 12 Bytes mit dem Wert von 0 geschrieben. Bei mehreren 0-Bytes wird ja bekanntlich im sowohl im MF- wie auch im MFM-Format der Takt aufgezeichnet. Die Bezeichnung SYNC bedeutet, daß der FDC auf den Takt synchronisiert wird.

Im Anschluß an die 12 Sync-Byte schreibt der FDC drei Bytes, die als Index-Adress-Mark bezeichnet werden. Diese drei Bytes sind nach einem ganz speziellen Schema aufgebaut und können vom FDC durch eine besondere Hardware auf dem Chip erkannt werden. Die Index-Adress-Markierung wird nur einmal zu Beginn der Spur, also unmittelbar hinter dem Index-Loch, geschrieben, und stellt gewissermaßen eine zusätzliche Kennung für den Beginn der Spur dar. Im Abschluß an die Index-Adress-Markierung wird ein Byte mit dem Wert von &FC und ein weiteres GAP, das GAP 1, mit 50 Bytes &4E formatiert. Dieses

GAP wird benötigt, um dem FDC beim späteren Lesen ausreichend Zeit zur internen Bearbeitung des Index-AM zu geben. Damit ist der typische Track-Anfang formatiert. Jetzt kommen Bereiche, die für jedem Sektor extra vorhanden sind.

Jeder Sektor beginnt mit 12 Sync-Bytes. Wie zuvor werden Nullen, also nur die Takt-Information, geschrieben. Nach den Sync-Bytes folgt eine Adress-Markierung, diesmal die ID-Adress-Mark. Auch die ID-AM ist 3 Bytes lang und kann vom FDC hardwaremäßig decodiert werden. Nach der ID-AM folgt ein Byte mit dem Wert &FE.

Bis zu diesem Punkt sind alle Sektoren der Diskette identisch. Um aber später auf die Tracks und Sektoren gezielt zugreifen zu können, besteht die Notwendigkeit, diese auch entsprechend zu markieren. Genau diese Aufgabe übernehmen die jetzt folgenden vier Bytes, die als ID-Feld bezeichnet werden. In der Reihenfolge der Beschreibung werden hier die Tracknummer, die Diskettenseite (also 0 oder 1), die Sektornummer und die Sektorgröße eingetragen. Die letzte Angabe kann natürlich nicht direkt eingetragen werden, da sonst für 512 Byte große Sektoren ein Zwei-Byte-Feld benötigt würde. Die entsprechende Verschlüsselung entspricht den Angaben, die beim Formatieren gemacht werden. 512 Byte große Sektoren werden mit einer 2 in diesem Byte markiert.
Damit hat man jedem Sektor seine spezielle Identifikation, die Sector ID, gegeben, an Hand derer ein späterer gezielter Zugriff möglich ist. Allerdings haben die Entwickler des Systems nichts dem Zufall überlassen. Um gegen eventuelle Fehler gewappnet zu sein, wird über die 3 Bytes der ID-AM, das eine &FE und die 4 Bytes des ID-Feldes eine 2 Byte große Checksumme nach einem speziellen Verfahren gebildet. Das verwendete Verfahren wird Cyclic Redundancy Check oder kurz CRC genannt und erlaubt die äußerst zuverlässige Erkennung von Lesefehlern. Die zwei gewonnenen zwei CRC-Bytes werden unmittelbar hinter das ID-Feld geschrieben.

Um dem FDC beim späteren Lesen der Daten ausreichend Zeit zum Prüfen der ID und CRC-Bytes zu geben, folgt erst einmal ein weiteres GAP, das GAP 2 mit einer Länge von 22 Bytes.

Auch bei diesem GAP ist der (unwesentliche) Datenwert wieder &4E. Das GAP 2 hat eine weitere Bedeutung. Alle Daten eines Sektors werden beim späteren Schreiben eines Sektors hinter dem GAP 2 geschrieben. Das GAP 2 erlaubt dem FDC, die Umschaltung zwischen Lesen und Schreiben vorzunehmen, ein Vorgang, der doch immer etwas Zeit beansprucht.

Ob Sie wohl erraten, was auf das GAP 2 folgt? Richig, es ist ein Sync-Bereich mit 12 Null-Bytes, der jetzt endlich den Beginn des eigentlichen Datenbereichs ankündigt. Aber noch werden nicht die Daten auf die Diskette geschrieben. Nach einem Sync folgt ja immer ein Adress-Mark-Bereich. Der AM, der jetzt formatiert wird, ist der DATA-Adress-Mark, der den gleichen Aufbau wie der ID-AM hat und somit ebenfalls vom FDC decodiert werden kann. Als Unterscheidung zum ID-AM folgt aber auf den DATA-AM ein Byte mit dem Wert &FB.
Und jetzt gehts los. Endlich kommt der Datenbereich, der beim Formatieren mit dem Wert des Füll-Bytes beschrieben wird. Die Größe dieses Bereichs ist unterschiedlich, aber bei der Formatierung durch die Angaben an den FDC festgelegt. So sind Datenbereiche zwischen 128 und 4096 Bytes möglich. Im letzten Fall passt aber nur ein einziger Sektor auf eine Spur, so daß man fast immer kleinere Sektorgrößen wählen wird. Im CPC ist die Sektorgröße auf 512 Bytes festgelegt.

Auch über den Datenbereich wird eine 2-Byte Checksumme nach dem CRC-Verfahren gebildet. Um auch Fehler in der DATA-ID zu erkennen, sind diese Bytes in der Bildung des CRC mit eingeschlossen, so das bei einer Sektorlänge von 512 Bytes insgesamt 516 Bytes geprüft werden.

Den Abschluß eines Sektors bildet das GAP 3. Die Länge dieses Gaps kann bei der Formatierung angegeben werden. Im CPC werden beim Formatieren 82 &4E-Bytes geschrieben. Dieses Gap hat eine ganz besondere Bedeutung. Werden auf einer formatierten Diskette später Sektoren mit neuen Daten überschrieben, so wäre es äußerst unwahrscheinlich, das sich die Diskette

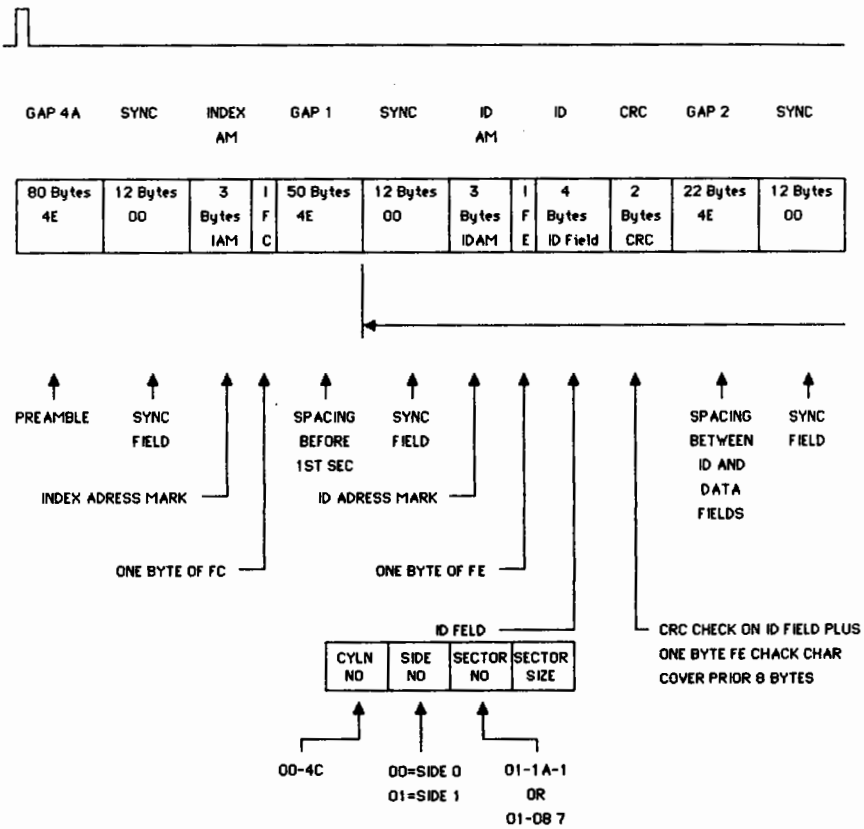
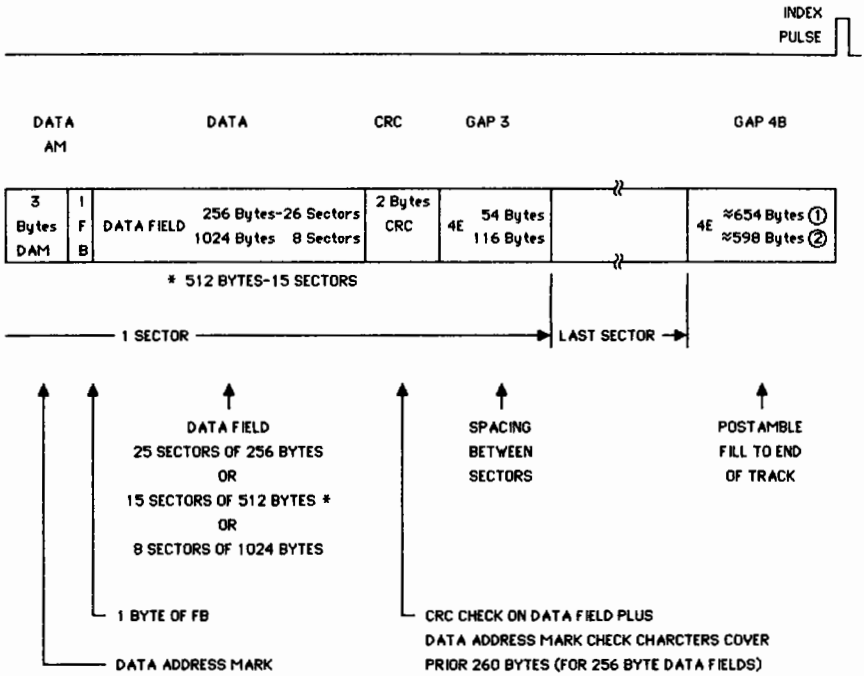

GAP 4A
SYNC
INDEX AM
GAP 1
SYNC
ID AM
ID
CRC
GAP 2
SYNC
80 Bytes 4E
12 Bytes 00
3 Bytes IAM
IFC
50 Bytes 4E
12 Bytes 00
3 Bytes IDAM
IFE
4 Bytes ID Field
2 Bytes CRC
22 Bytes 4E
12 Bytes 00
PREAMBLE
SYNC FIELD
INDEX ADRESS MARK
ONE BYTE OF FC
SPACING BEFORE 1ST SEC
SYNC FIELD
ID ADRESS MARK
ONE BYTE OF FE
ID FELD
CRC CHECK ON ID FIELD PLUS ONE BYTE FE CHACK CHAR COVER PRIOR 8 BYTES
SPACING BETWEEN ID AND DATA FIELDS
SYNC FIELD
CYLN NO
SIDE NO
SECTOR NO
SECTOR SIZE
00-4C
00=SIDE 0
01=SIDE 1
01-1A-1 OR 01-08 7

Abb. 38



① FOR 256 BYTES/SECTOR
② FOR 1024 BYTES/SECTOR
* 512 BYTES -15 SECTORS
NOT AN IBM STANDARD
BUT OFTEN USED

mit exakt derselben Geschwindigkeit wie beim Formatieren dreht. Ist die Geschwindigkeit aber auch nur geringfügig höher, so wird zwangsläufig der vom Sektor benötigte Kreisbogen auf dem Track länger, da die Aufzeichnungsgeschwindigkeit ja konstant ist. Ohne das GAP 3 würde bei einer solchen längeren Aufzeichnung ein unmittelbar folgender Sektor-ID einfach überschrieben, der nächste folgende Sektor wäre nicht mehr lesbar. Die durch die Gewschindigkeitsschwankungen erzeugten Längenunterschiede der Sektoren können durch das GAP 3 aber aufgefangen werden, obwohl auch beim Schreiben eines Sektors das GAP 3 erzeugt wird. Allerdings wird die Länge des beim Sektorschreiben erzeugten GAPs einfach drastisch reduziert. Tatsächlich werden nur noch 42 Gap-Bytes geschrieben, so das genügend 'Luft' für die autretenden Differenzen bleibt.
Damit ist ein Sektor vollständig formatiert, jetzt fordert der FDC die Werte für den nächsten Sektor an und schreibt auch diesen auf die Diskette. Sind so nacheinander alle Sektoren formatiert, dann werden bis zum Auftreten des Index-Impulses GAP-Bytes geschrieben. Mit dem Auftreten des Index-Impulses ist die Formatierung dieses einen Tracks beendet.

Wie man sieht, hat der FDC ganz schön zu tun, um einen Track für die Benutzung vorzubereiten. Andere FDC als der 765 sind da lange nicht so hilfreich. So benötigen z.B. die häufig eingesetzen Controller der Serie 197x beinah jedes Byte vom Prozessor, das Formatierprogramm wird sehr aufwendig und umfangreich. Durch die große Unterstützung des 765 jedoch wird das Formatieren mit diesem FDC fast zum Kinderspiel.

Kapitel 4
Das ROM und RAM des AMSDOS

Zugegeben, es ist nicht jedermanns Sache, sich mit den Innereien eines Betriebssystems zu befassen. Vielen Computerbesitzern ist es vollkommen ausreichend, wenn die vorhandenen Programme fehlerfrei und zuverlässig funktionieren. Diese Computer-Benutzer können getrost das folgende Kapitel überschlagen. Wenn jedoch eines Tages die vom AMSDOS zur Verfügung gestellten Möglichkeiten für bestimmte Aufgaben nicht mehr ausreichen, dann sollten Sie sich an die jetzt folgenden Seiten erinnern. In kompakter Form finden Sie hier die wesentlichen Einsprünge des Floppy-Betriebssystems und detailierte Beschreibungen der einzelnen Routinen. Wenn Ihnen irgendwelche Kommentare nicht klar werden, so können Sie mit Hilfe eines Disassemblers das Betriebssystem disassemblieren. In Verbindung mit den Kommentaren können Sie dann sicher schnell die gewünschte Information erlangen.

4.1 Das System-RAM des AMSDOS

Wie Sie ja wissen, steht Ihnen nach dem Einschalten des CPC und der Floppy ein Speicher von 42249 Bytes für BASIC-Programme zur Verfügung. Ohne die Floppy, d.h. ohne aktiviertes AMSDOS, sind es jedoch 43533 Bytes, wie z.B. der CPC 464 ohne Floppy-Interface auf Anfrage mitteilt. Die Differenz von 1284 Bytes geht zwar nicht vollständig zu Lasten der Floppy. Diese reserviert sich beim Einschalten aber immerhin den mit 1024 Bytes größten Happen. Dieser 1024 Bytes große Bereich beginnt üblicher Weise bei &A700. Die Einschränkung hat seinen besonderen Grund in der Verwaltung externer ROMs durch den CPC. Insgesamt kann der CPC maximal 252 externe ROMs im Speicherbereich von &C000 bis &FFFF verwalten. Dabei wird zwischen drei verschiedenen Typen von ROMs unterschieden, den Vordergrund-ROMs, den Hintergrund-ROMs und den Extension-ROMs. Ausschlaggebend für den Typ eines bestimmten ROMs ist das erste Byte (&C000) eines solchen ROMs.

Nach dem Einschalten wird zunächst versucht, ob sich ein externes Vordergrund-ROM mit der Adresse 0 am CPC befindet. Dazu wird ein OUT &DF00,0 erzeugt. Stellt das Betriebssystem fest, daß sich an dieser Stelle etwas 'rührt', so wird das externe Rom initialisiert und übernimmt damit die Kontrolle über die Maschine. Sollte sich jedoch kein externes Rom mit dieser ROM-Select-Adresse am Expansion-Port befinden, so wird das ON-Board-ROM, das BASIC als ROM 0 initialisiert. Nach dieser Initialisierung muß das aktivierte Vordergrundprogramm eventuell vorhandene Hintergrund-ROMs aktivieren. Dazu wird von der ROM-Adresse 7 beginnend jede ROM-SelectAdresse abgefragt, ob ein solches Background-ROM vorhanden ist. Im Falle eines solchen ROMs wird dies dann automatisch initialisiert.

Hat beim Einschalten des Rechners ein anderes als das eingebaute Vordergrund-ROM die Kontrolle über den CPC übernommen, so ist es ihm freigestellt, sich einen Bereich des RAMs als Arbeitsspeicher zu reservieren. Dazu kann es z.B. die RAM-Obergrenze, also das HIMEM nach unten verschieben. Bei der anschließenden Reservierung des Speichers für das AMSDOS verschiebt sich damit auch der vom AMSDOS belegte Speicher. In diesem Fall könnte z.B. das AMSDOS-RAM bei &A000 beginnen.

Diese Flexibilität des Rechners hat natürlich seinen Preis. Alle im Folgenden gemachten Adressangaben im Bereich ab &A700 sind nicht absolut zu sehen. Sie können unter den geschilderten Umständen durchaus in einem anderen Bereich liegen. Will man also Programmme mit direktem Zugriff auf das AMSDOS-RAM möglichst flexibel gestalten, so sollte man auf die Variablen nur indirekt zugreifen. Dazu muß man sich den tatsächlichen Anfang des AMSDOS-RAMs holen und auf die Variable mit dem Offset zum Beginn des Speichers zugreifen.

Was sich in der Beschreibung recht kompliziert anhört, wird in einem etwas später folgenden kleinen Beispiel deutlicher. Zunächst jedoch muß noch auf eine weitere Besonderheit der Floppy beziehungsweise des RAMs des AMSDOS hingewiesen werden. Es existiert nämlich nicht nur der schon beschriebene

Bereich von 1 K, in dem die meisten Systemvariablen untergebracht sind. Im Bereich von &BE40 bis &BE7F werden weitere 64 Bytes für die Floppy verwendet. Dieser Bereich ist nicht verschblich, liegt also immer definitiv fest.

Dieser Bereich ist allerdings etwas gefährdet. Der Stack des Prozessors wird normalerweise von &C000 abwärts angelegt. In Programmen, die diesen Stack nicht ordnungsgemäß verwalten, oder z.B. wegen rekursiver Programmierung einen sehr großen Stack benötigen, kann es durchaus vorkommen, das durch zu viele PUSHs oder CALLs der Stack so groß wird, das er den Speicherbereich der Floppy überschreibt. Im ersten Fall sollte man schleunigst sein Programm korrigieren. Im zweiten Fall muß man den Stack in einen anderen Speicherbereich legen.
Doch kommen wir jetzt zu der Liste der 64 Bytes, soweit uns die Bedeutung klar geworden ist.

Referenzliste Systemram 0BE40H bis 0BE7FH

BE40,BE41	Vector auf Disk Parameter Header Drive A
BE42,BE43	Vector auf Disk Parameter Block Drive A
BE44,BE45	Wartezeit für den Hochlauf des Drive-Motors nach MOTOR ON
BE46,BE47	Nachlaufzeit des Floppymotors nach dem letzten Zugriff
BE48	Wert für Warteschleife bei Track formatieren
BE49,BE4A	Werte für lange Verzögerungsschleife
BE4B	Anzahl Bytes bei Interrupt Status auslesen
BE4C-BE52	Buffer für Bytes aus Result Phase FDC
BE53	Drive (HS/US)
BE54	Track
BE55	Record
BE56	Drive (HS/US)
BE57	Track
BE58	Sector
BE59	Anzahl Records/Track (Blockmask+1)
BE5A	Drive (HS/US)

BE5B	Track
BE5C	Record
BE5D	
BE5E	Flag Sector lesen/schreiben
BE5F	Motor an/aus-flag
BE60,BE61	Vector auf Record-I/O-Buffer
BE62,BE63	Vector auf Sector-I/O-Buffer
BE64,BE65	Stack-Zwischenspeicher
BE66	Anzahl der Leseversuche
BE67,BE6C	Tick Block
BE67,BE68	Ticker Chain, Verkettung der Ticker-Liste
BE69,BE6A	Ticker Count, Anzahl der Ticker, um den Event zu kicken
BE6B,BE6C	Reload Count
BE6D,BE73	Event-block
BE6D,BE6E	Event Chain, Verkettung der Events
BE6F	Event Count
BE70	Event Class
BE71,BE72	Adresse der Event-Routine
BE73	ROM-Select der Far-Adresse für die Event-Routine
BE74	Buffer für gewünschte Tracknummer
BE75	Buffer für OP-Code an den FDC
BE76,BE77	Vector auf Sector-I/O-Buffer
BE78	Flag für Fehlermeldungen des Controllers ON/OFF
BE79-BE7C	unter AMSDOS in der vorliegenden Version nicht verwendet
BE7D,BE7E	IY-Speicher, Lowadress Memory Pool für Floppy
BE7F	Vektor für Manipulation der DISK-Routinen (DISC OUT OPEN etc.), normaler Weise &C9, RETURN

Zu diesen Adressen sollen noch einige Anmerkungen gemacht werden. Da sie, wie bereits erwähnt, absolut sind, ihre Lage im Adressraum also nicht verändern, kann hier mittels einfacher POKEs eingegriffen werden. Allerdings werden einige Variablen nur während der Laufzeit von Disk-Routinen verändert, so daß

von BASIC aus nicht sinnvoll eingegriffen werden kann. An anderen Stellen lassen sich jedoch recht wesentliche Eingriffe mit wenigen POKEs vornehmen.

POKEen Sie doch einmal einen Wert von 1 in die Speicherzelle &BE45. Geben Sie anschließend den Befehl CAT ein. Die Auswirkungen sind verblüffend. Wenn Sie genug Geduld hatten, so haben Sie einen ganz normalen CATALOG erhalten. Allerdings hat es wesentlich länger als sonst üblich gedauert. Die Werte in den Speicherzellen &BE44 und &BE45 bestimmen nämlich die Wartezeit nach dem Einschalten des Motors. Normalerweise wird eine Zeit von ca. einer Sekunde gewartet, wir haben diese Zeit jedoch drastisch erhöht.
Eine weitere, in Einzelfällen durchaus sinnvolle Manipulation von Wartezeiten können Sie durch POKEs in die Speicherzellen &BE46/&BE47 erzielen. Die Werte dieser Speicherzellen bestimmen die Nachlaufzeit der Floppy-Motoren nach dem letzten Zugriff auf die Diskette. Nun kann es durchaus passieren, das beim Lesen und Bearbeiten von Daten eines Files die Wartezeit gerade abläuft. Dann wird für den nächsten Zugriff der Motor erneut gestartet, gleichzeitig aber wieder eine Sekunde für die Hochlaufzeit gewartet. Wenn die Nachlaufzeit so verlängert wird, das der Motor zwischen den Zugriffen nicht mehr abschaltet, so kann eine deutliche Erhöhung der Verarbeitungsgeschwindigkeit festgestellt werden. Probieren Sie es doch einfach einmal.

Recht interessant sind auch die Bytes in den Adressen &BE4C bis &BE52. Hierhin werden nach allen Operationen des FDC die Angaben der Result-Phase abgelegt. Diese Werte können Sie nach Belieben selbst interpretieren, was im Fall des Ausschaltens der Controller-Fehlermeldungen durchaus wichtig ist.
Damit sind wir bei der nächsten, schon früher einmal erwähnten Speicherzelle &BE78. Wird hier ein Wert von &FF eingetragen, so werden die Fehlermeldungen des FDC nicht auf dem Bildschirm ausgegeben. In diesem Fall müssen Sie an Hand der Bytes aus der Result-Phase selbst auf die Fehler reagieren.
Sehr wichtig für den Zugriff auf den verschieblichen 1K-RAM-Bereich des AMSDOS sind die Speicherzellen &BE7D und &BE7E. Hier ist die Start-Adresse des 1K-Bereichs vermerkt.

Üblicher Weise steht hier die Adresse &A700. Sollte sich jedoch aus irgendwelchen Gründen der AMSDOS-RAM-Start einmal verändern, so wird in diesen Speicherzellen die richtige Start-Adresse zu finden sein.

Für die absoluten Spezialisten ist sicher die Speicherzelle &BE7F von großem Interesse. Hier finden Sie in der Regel den Wert &C9. Das ist der Z80-OP-Code für Return. Jeder Zugriff auf die gepatchten CAS-Routinen geht über diesen Return. Damit besteht eine einfache Möglichkeit, sowohl vor als auch nach der Ausführung der Routinen einzugreifen.

Zwar besteht auch die Möglichkeit, die gepatchten CAS-Routinen noch einmal zu patchen, also mit den Adressen Ihrer eigenen Routinen zu versorgen, dies Verfahren ist jedoch recht aufwendig.
Doch kommen wir jetzt zum wesentlich umfangreicheren Teil des Systemrams. In der folgenden Aufstellung sind wir davon ausgegangen, daß keine weitere Erweiterung angeschlossen ist. Somit beziehen wir uns auf die Adressen im Bereich ab A700

Die CP/M-Spezialisten unter Ihnen müssen allerdings selbst ein wenig forschen, hier ist die Organisation des vom BIOS benötigten RAMs anders. So liegen die FDCs an den Standard-CP/M-Adressen (&5C bis &7c). Die Disc Parameter Header liegen ab &AE58, die Disc Parameter Blöcke ab &ADD8 im Speicher.

Referenzliste Systemram 0A700H bis 0AA80H

A700	angemeldetes Drive
A701	angemeldeter User
A702	aktives Drive
A703,A704	Zeiger auf Disk Parameter Header (a910/a920) aktives Drive
A705	Flag, ob OPEN auf angemeldetem Drive aktiv

A706-A706	Zwischenspeicher für Stack-Pointer für alle logischen Routinen
A708-A72B	erweiterter File Control Block für OPENIN
A709-A728	File Control Block (FCB) Buffer OPENIN
A708	Flag für OPENIN ff = kein OPENIN aktiv 00 = OPENIN auf Drive A 01 = OPENIN auf Drive B
A709	Usernummer für OPENIN
A70A-A714	Filename für OPENIN, 8 Zeichen Filename, 3 Zeichen Extension
A715	&00 erster Eintrag, sonst Nummer des Extents
A716	&00
A717	&00
A718	Anzahl der Records zu diesem Extent
A719-A728	Nummern der Blocks zu diesem Extent
A729-A72B	Anzahl der bisher gelesenen Records bei INPUT
A72C-A74F	erweiterter File Control Block für OPENOUT
A750-A799	File Control Block (FCB) Buffer OPENOUT
A72C	Flag für OPENOUT ff = kein OPENOUT aktiv 00 = OPENOUT auf Drive A 01 = OPENOUT auf Drive B
A72D	Usernummer für OPENOUT
A72E-A738	Filename für OPENOUT, 8 Zeichen Filename, 3 Zeichen Extension aufgefüllt mit Blanks
A739	&00 erster Eintrag, sonst Nummer des Extents
A73A	&00
A73B	&00
A73C	Anzahl der Records zu diesem Extent

A73D-A74C	Nummern der Blocks zu diesem Extent
A74D-A74F	Anzahl der bisher geschriebenen Records bei OUTPUT
A750-A799	Fileheader OPENIN
A750	1 = Disk In Char 2 = Disk In Direkt
A751-A752	Vector auf Anfang des 2K-OPENIN-Buffers
A753-A754	Vector auf aktuelles Zeichen im OPENIN-Buffer
A755	Usernummer des Files, Bestandteil des Filenamens
A756-A764	Filename für den Fileheader, mit Nullen aufgefüllt
A755	Block Number
A766	Last Block
A767	File Type INPUT-File
A768-A769	Data Length
A76A-A76B	Data Location
A76C	First Block
A76D-A76E	Logical Length
A76F-A770	Entry Adress
A771-A794	User Fields Frei für den Anwender)
A795-A797	Drei-Byte Zähler Anzahl gelesene Zeichen
A798-A799	Zwei-Byte Checksumme über Fileheader OPENIN (a755-a797)
A79A-A7E3	Fileheader OPENOUT
A79A	1 Disc Out Char 2 Disc Out Direkt
A79B-A79C	Vector auf Anfang des 2K OPENOUT-Buffers
A79D-A79E	Vector auf aktuelles Zeichen im OPENOUT-Buffer
A79F	Usernummer des Files, Bestandteil des Filenamens

A7A0-A7AE	Filename für den Fileheader, mit Nullen aufgefüllt
A7AF	Block Number
A7B0	Last Block
A7B1	File Type OUTPUT-File
A7B2-A7B3	Data Length
A7B4-A7B5	Data Location
A7B6	First Block
A7B7-A7B8	Logical Length
A7B9-A7BA	Entry Adress
A7BD-A7BE	Länge des Datenblocks bei Disk Out Direkt
A7BF-A7C0	Entry Adresse bei Disk Out Direkt
A7C1-A7DE	User Fields, frei für den Anwender
A7DF-A7E1	Drei-Byte Zähler Anzahl geschriebene Zeichen
A7E2-A7E3	Zwei-Byte Checksumme über Fileheader OPENOUT (a79f-a7e1)
A7E4-A8E3	temporärer Buffer/Record-Buffer. Dieser Buffer wird gleichermaßen als Record-Buffer als auch als Buffer für die Überprüfung und Expandierung des eingegebenen Flenamens verwendet.
A874-A88A	Buffer für ursprünglich angelegte Tape-Vectoren, werden beim 'ITAPE'-Kommando wieder nach BC77... eingetragen
A8B-A88D	Far Adress für gepatchte Tape-Vectoren. Wird eingesetztem RST 4 benötigt. Zeigt auf CD30H in Rom 7.
A890-A8A8	extended Disk Parameter Block Drive A
A890,A891	SPT Records pro Track (36)
A892	BSH Block Shift (3)

A893	BLM	Block Maske (7)
A894	EXM	Extend Maske (0)
A895,A896	DSM	maximale Blocknummer (170)
897,A898	DRM	Max Einträge im Directory -1 (63)
A899,A89A	AL0,1	Verzeichnisgröße (c000h) binär codiert, entspricht 2 Blocks
A89B,A89C	CKS	Anzahl der im Verzeichnis zu prüfenden Einträge (0010) 16 Einträge
A89D,A89E	OFF	Spuroffset (2) belegte Systemspuren
A89F-A8A8	FDC-Parameter	
A89F	FSC	Erster Sector jeder Spur (41h)
A8A0	PST	Physikalische Sectoren pro Track (9)
A8A1	GPS	Länge Gap3 bei Sector read/write (2ah)
A8A2	GPT	Länge Gap3 bei Track formatieren (52h)
A8A3	FLB	Filler-Byte bei Track formatieren (e5h)
A8A4	BPS	Bytes/Sector (2) entspricht 512 Bytes
A8A5	RPS	Anzahl Records/Sector (4)
A8A6	Buffer für aktuellen Track	
A8A7	Flag für Spur 0 suchen, Read/Write Recalibrate	
A8A8	Flag, ob bei jedem Disk-Zugriff Login erfolgen soll	
A8A9-A8B8	CSA	16 Bytes für Checksummen
A8B9-A8CE	ALT	22 Bytes Allocation Table, Blockbelegung Drv A
A8D0-A8E8	extended Disc Parameter Block Drive B Belegung wie DPB Drive A	
A8E9-A8F8	CSA	16 Bytes für Checksummen

A8F9-A90E	ALT 22 Bytes Allocation Table, Blockbelegung Drv B
A910-A91F	Disk Parameter Header Drive A
A910-A911	XLT Tabelle Umsetzung Skew-Faktor (nicht genutzt)
A912-A913	TRACK BIOS-Speicher aktueller Track. Achtung! Wird von AMSDOS als DIRNUM verwendet
A914-A915	SECTOR BIOS-Speicher für akt. Sektor
A916-A917	DIRNUM BIOS-Speicher für aktuelle DIR-Nummer
A918,A919	DIRBUF Pointer auf 128-Byte I/O-Buffer (a930)
A91A,A91B	DBP Pointer auf DPB Drive A (a890)
A91C,A91D	CSV Pointer auf Speicher für Checksum Bildung (a8a9)
A91E,A91F	ALV Pointer auf Belegungstabelle (a8b9)
A920-A92F	Disk Parameter Header Drive B
A920-A921	XLT Tabelle Umsetzung Skew-Faktor (nicht genutzt)
A922-A923	TRACK BIOS-Speicher aktueller Track. Achtung! Wird von AMSDOS als DIRNUM verwendet
A924-A925	SECTOR BIOS-Speicher für akt. Sektor
A926-A927	DIRNUM BIOS-Speicher für aktuelle DIR-Nummer
A928,A929	DIRBUF Pointer auf 128-Byte I/O-Buffer (a930)
A92A,A92B	DBP Pointer auf DPB Drive B (a8d0)
A92C,A92D	CSV Pointer auf Speicher für Checksum Bildung (a8e9)
A92E,A92F	ALV Pointer auf Belegungstabelle (a8f9)
A930-A9AF	DIRREC 128 Bytes Buffer für einen Directory Record.

	Wird aus dem DIR-Sektor hierher übertragen
A9B0-ABAF	SECBUF Buffer für den physikalischen Datentransfer von und zur Floppy

Auch bei diesen Adressen 'juckt' es Sie sicher, die Auswirkungen von Manipulationen einiger Speicherzellen zu beobachten. Nur zu, es kann bekanntlich nichts am Rechner zerstört werden. Allerdings solten Sie nicht unbedingt Ihre ersten Versuche mit eingelegter und nicht schreibgeschützter Lieblingsdiskette unternehmen. Wenn die Daten auf der Diskette erst einmal ruiniert sind, dann ist es für das lange aufgeschobene Backup zu spät.

Wenn man den Speicherbereich genau betrachtet, so kann man sehr deutlich die Abgrenzung einzelner Bereiche feststellen. Diese Bereiche haben wir in der Aufstellung mit Namen benannt, die aber sicher nicht allen Lesern geläufig sind. Wir haben diese Begriffe aus dem CP/M entnommen, da viele Datenbereiche in ihren Funktionen mit solchen im CP/M identisch sind. So sind die Bereiche Disc Parameter Header und Disc Parameter Block (fast) exakt in dieser Form im CP/M vorgeschrieben. Jeder CP/M-Rechner hat DPBs und DPHs.

Auch FDCs, die File Control Blocks werden im CP/M genutzt. Allerdings sind einige Bereiche unter AMSDOS erweitet worden. Ein Standard-CP/M-DPB umfasst nur die 15 Bytes SPT bis OFF, die Erweiterungen sind AMSDOS-spezifisch und nicht auf andere CP/M-Rechner, ja nicht einmal auf das CP/M im CPC übertragbar. Ebenfalls AMSDOS-spezifisch sind die Fileheader für OPENOUT und OPENIN.

4.2 Das AMSDOS-ROM

Auf den nun folgenden Seiten finden Sie zwar kein vollständiges ROM-Listing. Statt dessen werden 'nur' die Kommentare zu den einzelnen Befehlen abgedruckt, und ergeben für Sie möglicherweise keinen unmittelbaren Sinn. In Verbindung mit einem Monitor-Programm oder einen Disassembler jedoch können Sie ohne weiteres unsere Kommentare durch Hinzufügen der Mnemonics erweitern. Da Sie bereit und willens sind, sich mit einem Betriebssystem auseinander zusetzen, wird Ihnen diese 'Kommentierung' unserer Kommentare sicher nicht schwer fallen.

Wie bereits erwähnt, belegt das AMSDOS nicht einmal die Hälfte der verfügbaren 16K des ROMs. Ganze 8K werden durch Teile des mit der Floppy gelieferten LOGO belegt. Damit halbiert sich schon einmal der verfügbare Bereich.

Dieser LOGO-Bereich wird im Folgenden aus mehreren Gründen ignoriert. Zum einen halten Sie ja ein Floppy- und kein LOGO-Buch in Händen. Des weiteren jedoch ist der LOGO-Teil im ROM nur ein kleiner Ausschnitt des gesamten LOGO-Interpreters. Weitere 32K dieser im übrigen sehr interressanten Programmiersprache finden Sie auf der CPC-System-Diskette. Würden wir den LOGO-Teil des ROMs abdrucken und kommentieren, so könnte kein Mensch damit etwas anfangen, da es sich ja nur um einen kleinen Ausschnitt des gesamten Interpreters handelt.

Aber auch die verbleibenden 8K werden nicht vollständig vom AMSDOS benutzt. Ganze 1024 Bytes im Bereich von &DC00 bis &DFFF werden gar nicht benutzt. Dieser Bereich ist möglicherweise für zukünftige Erweiterungen des AMSDOS vorgesehen. Jetzt verfügen wir nur noch über 7K, die aber immer noch nicht vollständig dem AMSDOS zur Verfügung stehen. Auch Teile des CP/M sind in diesem Bereich integriert. Dabei benut-

zen CP/M und AMSDOS viele Routinen im ROM gemeinsam, andere wiederum werden ausschließlich vom AMSDOS oder vom CP/M genutzt. So sind im ROM die kompletten Treiber-Programme für zwei serielle Schnittstellen eingebaut, die unter CP/M in vielfältiger Weise über das I/O-Byte den verschiedenen Devices zugeordnet werden können.

Nach Abzug aller ausschließlich von CP/M genutzten Speicherbereiche bleiben nur knapp 6K für das AMSDOS übrig bleibt. Diese 6K haben es aber in sich. Aber sehen Sie selbst:

```
****** Präfix für CP/M ROM
C000        ROM Type, Background ROM
C001        ROM Mark Number
C002        ROM Version Number
C003        ROM Modification Level

****** Adresse der Kommando-Tabelle
C004

****** Jump-Block AMSDOS-Befehle
C006        CPM ROM
C009        CPM
C00C        DISC
C00F        DISCIN
C012        DISCOUT
C015        TAPE
C018        TAPEIN
C01B        TAPEOUT
C01E        A:
C021        B:
C024        DRIVE
C027        USER
C02A        DIR
C02D        ERA
C030        REN

****** Jump-Block Disc Controller-Befehle
C033        ^81 Fehlermeldungen enable/disable
C036        ^82 Laufwerk-Daten angeben
C039        ^83 Disc-Format bestimmen
C03C        ^84 Sector lesen
C03F        ^85 Sector schreiben
C042        ^86 Track formatieren
C045        ^87 Track suchen
C048        ^88 Laufwerkstatus bestimmen
C04B        ^89 Anzahl Leseversuche festlegen

****** Jump-Block CP/M Einsprünge
C04E
C051


****** Jump-Block serielle I/O-Routinen für CP/M
C054        vollständige Initalisierung SIO & 8253
C057
C05A        Chan. A RX-Buffer voll?
C05D        Chan. A ein Zeichen holen
C060        Chan. A TX-Buffer leer?
C063        Chan. A ein Zeichen senden
C066        Chan. B RX-Buffer voll?
C069        Chan. B ein Zeichen holen
C06C        Chan. B TX-Buffer leer?
C06F        Chan. B ein Zeichen senden
```

```
****** Tabelle der DOS-Befehle
C072
C0B3

****** Disc Controller-Befehle
C0B6
C0BF        Kennzeichen Ende der Tabelle

****** Interupt-Vector & Portadresse GA retten
C0C0        INT-Vector (RST 7)
C0C3        im Ram merken
C0C6        JMP-Opcode
C0C8        für CALL-Handler
C0CB
C0CC
C0CF        INT verbieten für Benutzung
C0D0        des alternativen Registersets
C0D1        Portadresse GA und ROM-Konfig. retten
C0D5        Registersatz zurückschalten
C0D6
C0D9

******
C0DB
C0E8        für Nutzung des alt. Reg.-Sets nötig
C0E9
C0EA
C0EB        Portadresse GA und ROM-Select wieder holen
C0EF        Carry löschen
C0F0
C0F1
C0F2
C0F5        Adresse des Sprungs bei AD33
C0F8        Interrupts wieder freigeben
C0F9

****** 'CALL AD33'-Handler, alt. Reg.-Set bleibt erhalten
C0FA        Interrupts verbieten
C0FB        um mit alternativem Reg.-Set
C0FC        zu arbeiten
C0FD        hl merken
C100        Return-Adresse nach hl
C101        Stack-Pointer sichern
C105        und initialisieren
C108        alle Register auf initialisierten Stack
C109
C10B
C10D        (Port-Adresse Gate Array)
C111
C112        Adresse nach 'CALL AD33' holen und CALLen
C115        könnte wieder zugelassen sein
C116        Register tauschen
C117
```

```
C118        hat sich evtl geändert
C11C        INT-Vector wieder verbiegen
C11F
C122        Register zurückholen
C124
C126
C127        hl war nicht gePUSHed
C12A
C12B        auf Std. Reg.-Set umschalten
C12C        alten Stackpointer restaurieren
C130
C131        gewünschte Routine ausgeführt

****** 'CALL AD33'-Handler, alt. Reg.-Set wird nicht gerettet
C132        INT verbieten
C133        alternativer Reg.-Set
C134
C135        Rücksprungadresse vom Stack
C136        Stackpointer merken
C13A        Stack initialisieren
C13D        Adresse nach CALL AD33 holen und JPen
C140        könnte wieder zugelassen sein
C141        Reg.-Set umschalten
C142        INT-Vector wieder verbiegen
C145
C148        Reg.-Set umschalten
C149        Stackpointer restaurieren
C14D
C14E        gewünschte Routine ausgegführt

****** JP auf die Adresse hinter 'CALL AD33'
C14F        (System Interrupt Vector)
C153        INT-Vector verbiegen
C157        Basisadresse für Disc-Ram
C15B        Folgebytes nach CALL AD33
C15C        nach de holen
C15D
C15E        auf den Stack
C15F
C161
C162        und via RET anspringen

******
C163
C166        Sprung auf INT-Vector

******
C168        hl merken
C16B        RET-Adresse nach hl
C16C
C16D        um zwei erhöhen, also hinter Adressangabe
C16E
C16F        mit RET-Adresse austauschen
```

```
C170        originalen Wert auf Stack
C171        hl restaurieren
C174

****** System-INT-Vector eintragen, JP (DE)
C177
C17E

****** BIOS Jump-Block, unter CP/M im Ram ab 0AD00H
C17F        COLD BOOT
C182        WARM BOOT
C185        CONSOLE STATUS
C188        CONSOLE INPUT
C18B        CONSOLE OUTPUT
C18E        PRINTER OUTPUT
C191        PUNCHER
C194        READER
C197        TRACK 0
C19A        SELECT DRIVE
C19D        SELECT TRACK
C1A0        SELECT SECTOR
C1A3        INSTALL BUFFER
C1A6        READ SECTOR
C1A9        WRITE SECTOR
C1AC        PRINTER STATUS
C1AF        SEKTORNUMMER üBERSETZEN

****** CPM-COLD BOOT
C1B2        KL CURR SELECTION
C1B5        ROM Selection
C1B6        Entry Point Adresse
C1B9        MC START PROGRAM

****** CPM ROM
C1BC
C1BE        KL CURR SELECTION
C1C1        ROM Select Adresse auf 0 prüfen
C1C2        => LK 1 auf Contr. Board offen, CP/M
C1C4        Adresse himem
C1C6        Adresse lomem
C1C7        hl = himem
C1CA        wird um 0400h herunter gesetzt
C1CB        neuen himem merken
C1CC
C1CD
C1CE        iy = himem+1
C1D0        FDC und Event initialisieren
C1D3        Tape-Vektoren patchen
C1D6        die neuen Werte für lomem
C1D7        und himem an KL START
C1D8        PROGRAM übergeben
```

```
C1DA        Kennzeichen Initialisierung OK
C1DB

****** ENTRY CP/M-Kaltstart
C1DC        Stack initialisieren
C1DF
C1E6
C1E9        löscht (de) bis (de+bc)
C1EC
C1EF
C1F0        IO-Byte Standardeinstellung
C1F2
C1F5        Drive und User
C1F6
C1F9        Jump-Tabelle Controller Befehle &81-&89
C1FC        nach BE80 copieren
C1FF        insgesamt 3fh Bytes
C202        übertragen
C204        INT-Vector & Portadresse GA retten
C207        FDC und Event initialisieren
C20A        Sectornummer
C20C        Track und Drive
C20F        Buffer-Adresse
C212        Sector lesen
C215        => gelesener Sector leer?
C218        Fehler aufgetreten
C21A
C21E
C221        INT-Vector auf C163, JP (DE)

****** Fehler beim Laden des BOOT-Sectors
C224        Msg. 15 'Failed to load Boot sector'
C226        'CHAN., IGN. or RETRY' abfragen
C229

****** CP/M CCP und BDOS von Disc laden, Warm Boot
C22B
C22E
C231        b=Sectorzähler, c=Sectornummer
C234        Track und Drive
C237        Bufferadresse merken
C238        gewünschte Anzahl Sectoren lesen
C23B        Anfang Buffer wieder nach hl
C23C        prüft, ob gelesener Sector leer
C23F        Fehler aufgetreten
C241
C24B
C24C        Sector-Länge
C24F
C251        Bufferadresse
C252        b=Sectorzähler, c=Sectornummer
C255        Track und Drive
C258        gewünschte Anzahl Sectoren lesen
```

```
C25B    Fehler aufgetreten
C25D
C266
C267    OP-Code für JP
C269
C26C    BDOS-Entry, 8F00
C26F    JP - Code
C272
C275
C276    BIOS-Entry, AD00
C279    Tabelle der CP/M-Vektoren
C27C    51 Bytes lang
C27F    nach AD00
C281    User und Drive
C284
C285    Drive ausmaskieren
C287    Drive 0 oder 1?
C289    ist 0 oder 1 =>
C28B    sonst auf 0 setzen
C28D
C28E
C28F    INT-Vector auf C163, JP(de)

****** Fehler beim laden von CP/M aufgetreten
C292    Msg. 14 'Faild to load CPM'
C294    'CHAN., IGN. or RETRY' abfragen
C297    WARM BOOT


****** läd fortlaufend Sectoren, Anzahl in b, Sector in c,Trk in d
C299    Sector laden
C29C    Fehler aufgetreten
C29D    Sectornummer in Akku
C29E    nächsten Sector
C29F    war der letzte Sector Nr 49h?
C2A1    => war nicht 49h
C2A3    Sector 41h
C2A5    auf nächstem Track
C2A6    Bufferzeiger 2 Pages höher setzen
C2A7
C2A8    alle Sectoren gelesen?
C2AA    Kennzeichen alles OK
C2AB

****** prüft, ob gelesener Sector leer ist
C2AC    Bufferadresse
C2AD    512 Bytes
C2B0    erstes Zeichen aus Buffer in Akku
C2B1    (Bufferzeiger)
C2B2    Zeiger erhöhen
C2B3    Kennzeichen OK
C2B4    wenn ungleich, dann =>
```

```
C2B6        Schleife über 256 Bytes
C2B8        mal zwei = ein Sector
C2B9
C2BB        Carry löschen
C2BC        Bufferzeiger wiederholen
C2BD

****** WARM BOOT
C2BE        'JP C22B'
C2C1

****** CONSOLE INPUT
C2C3        (hl)=> Console In Zuordnung
C2C6

****** CONSOLE OUTPUT
C2C8        (hl)=> Console Out Zuordnung
C2CB

****** PRINTER STATUS
C2CD        (hl)=> List Device Status Zuordnung
C2D0

****** PRINTER OUTPUT
C2D2        (hl)=> List Device Output Zuordnung
C2D5

****** PUNCHER
C2D7        (hl)=> Puncher Zuordnung
C2DA

****** READER
C2DC        (hl)=> Reader Zuordnung
C2DF

****** CONSOLE STATUS
C2E1        (hl)=> Console Status Zuordnung
C2E4        'JP C46A' Zuordnung über I/O-Byte
C2E7

****** TRACK 0 SUCHEN
C2E9        'JP C51F'
C2EC
C2F1

****** SELECT DRIVE
C2F2        'JP C4F0'
C2F5

****** READ SECTOR
C2F7        'JP C54C'
C2FA
```

```
****** WRITE SECTOR
C2FC        'JP C52E'
C2FF

******
C301
C312

****** Tastatur-Status prüfen, ist Zeichen verfügbar?
C313
C31A
C31B        TXT CURSOR ON
C31E        KM READ CHAR, ein Zeichen von Tastatur
C321        wenn Zeichen verfügbar, KM RETURN CHAR
C324        0ffh=> Zeichen verfügbar, 00 kein Zeichen
C325

****** Console Input, ein Zeichen von der Tastatur holen
C326        Keyboard Modus Flag
C329
C32A        Flag prüfen
C32B        => Keyboard Mode 'INPUT'
C32D        Keyboard Mode 'INKEY', KM READ CHAR
C330        => kein Zeichen bekommen
C332
C33D

******
C33E        Keyboard Mode auf 'INPUT'
C33F
C347

****** Console Input, auf ein Zeichen von Tastatur warten
C348
C34C
C34D        TXT CURSOR ON
C350
C352        KM WAIT CHAR, auf Zeichen warten

****** High Speed Reader als Reader, nicht implementiert
C355        EOF
C357

****** Status CRT als Printer, High Speed Reader als Reader
C358

****** High Speed Puncher als Puncher Device
C35A


****** CRT-Device, ein Zeichen auf Bildschirm ausgeben
C35B
C35E
```

```
C35F
C360        TXT CURSOR OFF
C363
C365        auszugebendes Zeichen in Akku
C366        TXT OUTPUT
C369        Leerzeichen
C36B        => kein Control-Code
C36C        TXT GET CURSOR
C36F        TXT VALIDATE
C372
C373        TXT PLACE CURSOR
C376        TXT REMOVE CURSOR

****** Line Printer Status, prüft, ob Centronics Busy
C379        MC BUSY PRINTER, Carry wenn Busy
C37C        Carry, wenn nicht Busy
C37D        0ffh => nicht Busy, 00 wenn nicht Busy
C37E

****** Line Printer Output, ein Zeichen zum Printer
C37F        Zeichen in Akku
C380        MC PRINT CHAR, ausgeben
C383        Zeichen erfolgreich geschickt
C384        Printer Busy, Keyboard testen
C387        neuen Versuch

****** Serial I/O initialisieren (hl)=> Parametertabelle
C389
C38A        SIO Channel A/Control-Register
C38D        Speicher für WR-Reg. 5, Channel A
C390        Channel Reset, Init. Chan. A
C393
C394        SIO Channel B/Control-Register
C395        Speicher für WR-Reg. 5, Channel B
C396        Channel Reset, Init. Chan. B
C399        Modus Timer 0 des 8253
C39B        Lowbyte für Portadresse Timer 0
C39D        Sende-Baudrate Cannel A einstellen
C3A0        Modus Timer 1 des 8253
C3A2        Lowbyte für Portadresse Timer 1
C3A3        Emfangs-Baudrate Channel A einstellen
C3A6        Modus Timer 2 des 8253
C3A8        Lowbyte für Portadresse Timer 2
C3A9        Emf.- und Sende-Baudrate Channel B
C3AC
C3AD

****** Baudrate-Generator 8253 init., (hl) => Lo-Hi Timerwerte
C3AE        Portadresse Control-Wort 8253
C3B1        Control-Wort an 8253 ausgeben
C3B3        Low-Byte Portadresse des gew. Timers
C3B4        Low-Byte Timerwert
C3B5
```

```
C3B6        in Timer laden
C3B8        High-Byte Timerwert
C3B9
C3BA        in Timer laden
C3BC

****** SIO-Channel in (BC) initialisieren
C3BD        OP-Code Channel rücksetzen
C3BF        an SIO ausgeben
C3C1        Write-Register 4 auswählen
C3C3
C3C5        Tabelleneintrag für
C3C6        Parity, Stop-Bits und Clock-Mode
C3C7        an SIO ausgeben
C3C9        Write-Register 5 auswählen
C3CB
C3CD        Tabelleneintrag für Handshake und
C3CE        Bits/Char. nach (de) merken (ADC6/ADC7)
C3CF
C3D0        und an SIO ausgeben, Sende (TX)-Parameter
C3D2        Write-Register 3 auswählen
C3D4
C3D6        Tabellenwert für Handshake und Bits/Char
C3D7
C3D8        an SIO ausgeben, Emfangsparameter, RX
C3DA


****** Channel A RX-Buffer voll?
C3DB        SIO Channel A, Control-Register
C3DE        (hl)=> Inhalt von Write-Reg. 5, Chan. A
C3E1

****** Channel B RX-Buffer voll?
C3E3        SIO Channel B, Control-Register
C3E6        (hl)=> Write-Reg. 5, Chan. B
C3E9        Read-Reg. 0 des gew. Chan. lesen
C3EB        Bit 0, RX-Character bereit?
C3EC
C3ED        => ein Zeichen vorhanden
C3EE        DTR-Bit setzen
C3F1        Read-Reg. 0 lesen
C3F3        Bit 0, RX-Character bereit?
C3F4
C3F5        DTR-Bit rücksetzen

****** SIO Channel A ein Zeichen holen
C3F7        SIO Channel A, Control-Register
C3FA        (hl)=> Inhalt von Write-Reg. 5, Chan. A
C3FD

****** SIO Channel B ein Zeichen holen
C3FF        SIO Channel B, Control-Register
```

```
C402        (hl)=> Write-Reg. 5, Chan. B
C405        Read-Reg. 0 lesen
C407        RX-Character verfügbar?
C408        => ein Zeichen emfangen
C40A        DTR-Bit setzen
C40D        Tastatur abfragen
C410        Control Z als Ende eingegeben?
C412        => DTR-Bit rücksetzen, RET
C414        Read-Reg. 0 lesen
C416        RX-Character verfügbar?
C417        => noch nicht verfügbar
C419        DTR-Bit rücksetzen
C41C        Portadresse Datenregister
C41D        emfangenes Zeichen auslesen
C41F


****** DTR-Bit rücksetzen, Emfang freigeben
C420        Bit 7 löschen, ist relevant
C422

****** DTR-Bit setzen, Emfang sperren
C424        Bit 7 setzen
C426
C427
C428        Write-Reg. 5
C42A        anwählen
C42C        (hl)=> Inhalt von WR-Reg 5
C42D        Bit 7 ausblenden
C42F        je nach Ansprung Bit 7 gesetzt/rückgesetzt
C430        in Write-Reg. 5 schreiben
C432
C434

****** testet, ob TX-Buffer Channel A leer
C435        SIO Channel A, Control-Register
C438

****** testet, ob TX-Buffer Channel B leer
C43A        SIO Channel B, Control-Register
C43D        Read-Reg 0 auslesen
C43F        TX-Empty-Bit ausmaskieren
C441        => Buffer ist nicht leer
C442        Kennzeichen Buffer leer
C443
C444

****** ein Zeichen über Channel A senden
C445        auszugebenes Zeichen in Akku
C446        SIO Channel A, Control-Register
C449
```

```
****** ein Zeichen über Channel B senden
C44B        auszugebenes Zeichen in Akku
C44C        SIO Channel B, Control-Register
C44F        Zeichen sichern
C450        Tastatur abfragen
C453        ist Sende-Buffer leer?
C456        => noch nicht leer
C458        Zeichen wieder in Akku
C459        Portadresse Datenregister
C45A        Zeichen in SIO schreiben
C45C

****** READER-Status über I/O-Byte bestimmen
C45D        READER-Status Tabelle
C460

****** READER-Input über I/O-Byte
C462        READER-Input Tabelle
C465

****** PRINTER-OUTPUT über I/O-Byte
C467        PRINTER-Output Tabelle

****** I/O-Device über I/O-Byte bestimmen, (hl)=> Zuordnungstabelle
C46A        Schleifenzahl für die vier Devices
C46B        (hl)=> erste Zuordnung
C46C        I/O-Byte, üblich &81
C46F
C470        (b) mal I/O-Byte nach links
C472        relevante Bits isolieren
C474
C476        ergibt Offset in Zuordnungstabelle
C477        zu Start addieren
C478        Adresse der I/O-Routine nach de
C479
C47B
C47C        Indirekter Sprung auf I/O-Routine

****** CONSOLE STATUS
C47D
C47E        JP 0C3D8H, Char. SIO Chan. A verfügbar?
C480        Character von Tastatur verfügbar?
C482        READER Status über I/O-Byte
C484        JP 0C3E3H, Char. SIO Chan. B verfügbar?

****** CONSOLE INPUT
C486
C487        JP 0C3F7H, Char. von SIO Chan. A holen
C489        Zeichen von Tastatur holen
C48B        READER Zeichen lesen über I/O-Byte
C48D        JP 0C3FFH, Char. von SIO Chan. B holen
```

```
****** CONSOLE OUTPUT
C48F
C490        JP 0C445H, Char. über SIO Chan. A senden
C492        Zeichen auf Bildschirm ausgeben
C494        PRINTER OUTPUT über I/O-Byte
C496        JP 0C44BH, Char. über SIO Chan. B senden

****** PRINTER STATUS
C498
C499        JP 0C435, Sendebuffer Chan. A leer?
C49B        nicht implementiert
C49D        testet Centronics Busy
C49F        JP 0C43A, Sendebuffer Chan. B leer?

****** PRINTER OUTPUT
C4A1
C4A2        JP 0C445H, Char. über SIO Chan. A senden
C4A4        Zeichen auf Bildschirm ausgeben
C4A6        gibt Zeichen auf Centronics-Port aus
C4A8        JP 0C44BH, Char. über SIO Chan. B senden

****** PUNCHER
C4AA
C4AB        JP 0C445H, Char. über SIO Chan. A senden
C4AD        nicht implementiert, RET
C4AF        JP 0C44BH, Char. über SIO Chan. B senden
C4B1        Zeichen auf Bildschirm ausgeben

****** READER Status
C4B3
C4B4        JP 0C3D8H, Char. SIO Chan. A verfügbar?
C4B6        nicht implementiert
C4B8        JP 0C3E3H, Char. SIO Chan. B? verfügbar?
C4BA        prüft, ob Zeichen von Keyb. vorhanden

****** READER Zeichen lesen
C4BC
C4BD        JP 0C3F7H, Char. von SIO Chan. A holen
C4BF        nicht implementiert, holt EOF
C4C1        JP 0C3FFH, Char. von SIO Chan. B holen
C4C3        holt Zeichen von Keyboard

******
C4C5        prüfen ob CONTROL C gedrückt
C4C8        ENTER ?
C4CA        => nicht ENTER
C4CB
C4CC
C4CD        ein weiteres Zeichen von Tastatur holen
C4D0
C4D2
```

```
****** prüfen, ob CONTROL C gedrückt wird
C4D3
C4D5
C4D6        Keyboard Status prüfen
C4D9
C4DA        => kein Zeichen verfügbar
C4DC        Zeichen von der Tastatur holen
C4DF        CONTROL C?
C4E1        => nicht Control C
C4E3        Systemmeldung 14 ..^C
C4E5        ausgeben
C4E8        Warm Boot

****** keine Aktion, CONTROL C nicht gedrückt
C4EB
C4EE

****** nicht benutzt
C4EF

****** SELECT DRIVE
C4F0        Drive-Nr. in Akku
C4F1        darf nur 0 oder 1 sein
C4F3
C4F6        Drive-Nr. zu groß
C4F7        bisherige Drive-Nr in Akku
C4F8        Bit 0 ins Carry
C4F9        Sprung, wenn Drive 1 bisher aktiv
C4FB        gewünschtes Drive nach e
C4FC
C4FE        Akku mit Disc-Param.-Block-Wert 18h laden
C501        wenn ungleich 0
C502        dann Sprung
C504
C505        sonst Disc Format bestimmen
C508
C509
C50A        Drive-Nummer in Akku
C50B        HS/US-Buffer
C50E        Offset zu Disc Parameter Header Drive A
C511        Wenn Drive A benutzt, dann Sprung,
C512
C514        sonst Offset DPH für Drive B
C517        hl=hl+iy, hl => Pointertabelle Anfang

****** Record-Bufferadresse eintragen
C51A        (bc):= Record-Buffer
C51E
```

```
****** Track 0 suchen
C51F
C524
C525        (Buffer für Tracknummer)
C528

****** Recordnummer an Controller
C529
C52A        (Buffer für Recordnummer)
C52D

****** Write Record
C52E
C530
C532        Drive, Track, Rec. von be53h nach be5ah
C535        testet, ob Drv, Trk & Rec in be53h = be5ah
C538        => sind gleich
C53B        Sector aus Record-Nummer bestimmen
C53E
C53F
C540        überträgt Record in Sector-Buffer
C543        Anzahl Records
C544
C545        Sectoroffset addieren, Sector schreiben
C548        => Fehler
C549
C54B

****** Read Record
C54C        Akku löschen und
C54D        in Blockmask+1 eintragen
C550        Sector aus Recordnr. bestimmen
C553        Record in Record-Buffer übertragen
C556
C559

****** Recordnummer übersetzen
C55A        überträgt nur bc nach hl
C55B
C55C

****** Sector ID lesen, evtl Fehler abfragen
C55D        Statusreg. FDC
C560        Code Sector-ID lesen
C562        Akku an FDC ausgeben
C565        Unit Select/Head Select
C566        Akku an FDC ausgeben
C569        Result Phase FDC, Drive READY?
```

```
****** ermittelt aus Sector ID das Diskettenformat
C56C        Motor ein, Add Ticker Laufzeit
C56F
C571        Akku mit Disc-P.-Block-Wert 16h laden
C574        das ist aktuelle Tracknummer
C575        Anzahl der Leseversuche
C577        Adresse Routine 'Sector ID lesen'
C57A        Track in d suchen
C57D        NC = Fehler
C57E        Sectornr. vom FDC in Result-Phase

****** ^83h Disc-Formattierung ermitteln
C581        enthält vom FDC gelesene Sectornummer
C582        Akku löschen
C583        Disc-Param.-Block Anfang nach hl
C586
C587
C588        Standard Disc-Parameter-Block (DPB) im Rom
C58B        22 Bytes
C58E        in aktuellen Disc-Param.-Block
C590        Tabellenanfang
C591        Sectornummer
C592        Bit 0 bis 5 löschen
C594        ist Bit 6 gesetzt?
C596
C597        dann Standard-Format-Tabelle benutzen
C598        Anfang DPB Daten-Format, Tab.2
C59B        Bit 6 und 7 gesetzt?
C59D        wenn ja, dann Daten-Format verwenden
C59F        Anfang DBP IBM-Format, Tab.1
C5A2        die ersten beiden Tabellenwerte
C5A3        in Disc-Param.-Block übertragen
C5A4
C5A9
C5AC        hl zeigt auf Disc-Parameter-Block +5
C5AD        Anzahl Blöcke/Disc
C5AE        die nächsten beiden Bytes
C5AF        in aktuellen Disc-Parameter-Block
C5B0        übertragen
C5B1
C5B4
C5B7        hl zeigt auf Disc-Parameter-Block +13
C5B8
C5B9        die restlichen 6 Bytes
C5BC        in Disc-Parameter-Block übertragen
C5BE
C5BF
```

****** Tabellenwerte für IBM-Format
C5C0 Anzahl Records/Track, SPT
C5C2 Anzahl Blocks/Disc, DSM
C5C4 Anzahl Spuren für Betriebssystem, OFF
C5C6 Sectoroffset
C5C7 Sectoren/Track
C5C8 Länge GAP 3 read/write
C5C9 Länge GAP 3 formatieren

****** Tabellenwerte für Format Datendisc
C5CA Anzahl Records/Track, SPT
C5CC Anzahl Blocks/Disc, DSM
C5CE Anzahl Spuren für Betriebssystem, OFF
C5D0 Sectoroffset
C5D1 Sectoren/Track
C5D2 Länge GAP 3 read/write
C5D3 Länge GAP 3 formatieren

****** Tabelle (7 Bytes) wird nach be44... copiert
C5D4 Hochlaufzeit für Discmotor
C5D6 Laufzeit Ticker für Floppymotor
C5D8
C5D9 Werte für lange Verzögerungsschleife

****** Tabelle (2 Bytes) wird bei &82 benötigt
C5DB Head Unload Time für FDC = 32 ms
C5DC Head Load Time für FDC = 16 ms

****** initialisiert DPHs, DPBs, FDC und Event
C5DD Adresse Ram für Controllerroutinen
C5E0 Anzahl der Bytes
C5E3 löscht (de) bis (de+bc)
C5E6 initialisiert Disc Parameter Block/Header
C5E9 Drive-Motor ausschalten
C5EC Tabelle FDC-Parameter
C5EF Drive-Parameter FDC initialisieren
C5F2 KL ASK CURR SELECTION
C5F5 Rom Selection für Event-Routine
C5F6 Event-Klasse asyncron
C5F8 Event-Block
C5FB Adresse der Eventroutine
C5FE KL INIT EVENT
C601

****** ^89 setzt Anzahl der Leseversuche, Anzahl im Akku
C603
C604 Anzahl der Leseversuche
C607
C60C

```
****** ^82 Laufwerkdaten spezifizieren
C60D        FDC-Ram +4
C610
C613
C615        Statusregister FDC
C618        OP-Code Specify Drive Parameter
C61A        an FDC ausgeben
C61D        Wartezeit in Millisekunden (12)
C620
C624
C625        ergibt A0h = 12 ms Step Rate
C627        Head Unload-Time in die Bits 0 bis 3
C628        an FDC ausgeben
C62B
C62C        Head Load Time
C62D        an FDC ausgeben

****** ^88 Laufwerkstatus bestimmen
C630        Routine Laufwerkstatus bestimmen
C633        => Fehler aufgetreten
C634        Akku mit FDC-Status 0 laden
C637

****** Routine Laufwerkstatus bestimmen
C638        Motor an, Add Ticker für Laufzeit
C63B        Akku enthält Drivenr.
C63C        Sense Interrupt Status FDC
C63F        Statusregister FDC
C642        OP-Code Sense Drive Status
C644        Akku an FDC ausgeben
C647        Drivenr.
C648        Akku an FDC ausgeben
C64B        Result Phase FDC auslesen

******  ^85  Sector schreiben e=Drv, d=Trk, c=Sec, hl=I/O-Buffer
C64E        OP-Code Sector schreiben
C650

****** ^86 Track formatieren
C652        OP-Code Track formatieren
C654        Motor an, Add Ticker für Laufzeit
C657        Anzahl Versuche
C659        Read/Write/Format cont'd
C65C
C665

****** ^84 Sector lesen, e=Drv, d=Trk, c=Sec, hl=I/O-Buffer
C666        Motor an, Add Ticker für Laufzeit
C669        OP-Code Sector lesen
C66B
```

```
****** Read/Write/Format cont'd
C66D        512-Byte I/O-Buffer
C670        OP-Code FDC
C671        Sectornummer
C672        Opcode und gewün. Sector merken
C675        Anzahl der Leseversuche
C676        Adresse FDC programmieren
C679        Track in d suchen, 'Call (hl)'

****** FDC für gewünschte Aktion programmieren
C67C        (Opcode FDC und gewünschter Sector)
C67F        Statusregister FDC
C682        FDC-Opcode
C683        an FDC ausgeben
C686        Head Select/Unit Select, also Drivenr.
C687        an FDC ausgeben
C68A        ausgegebenen OP-Code in Akku
C68B        Track formatieren?
C68D        Sprung wenn Sector lesen/schreiben
C68F
C691        Bytes/Sector aus Disc-Param.-Block an FDC
C694
C696        Sectors/Track aus Disc-Param.-Blk an FDC
C699
C69B        Länge GAP3 aus Disc-Param.-Block an FDC
C69E
C6A0        Filler Byte, Drv.Param.-Block-Wert 13h
C6A3        Read/Write/Format Execution Phase

****** Einsprung lesen/schreiben eines Sectors
C6A5        Track-Nummer
C6A6        Akku an FDC ausgeben
C6A9        Kopf-Nummer (für Doppelkopf-Drives)
C6AA        Akku an FDC ausgeben
C6AD        Sector-Nummer
C6AE        Akku an FDC ausgeben
C6B1        Bytes/Sector
C6B3        aus Disc-Parameter-Block an FDC
C6B6        Sector-Nummer als letzten Sector
C6B7        Akku an FDC ausgeben
C6BA        Länge GAP3 bei Read/Write
C6BC        aus Disc-Parameter-Block an FDC
C6BF        DTL, muß ffh sein

****** Read/Write/Format Execution Phase
C6C1        512 Byte Sector I/O
C6C4
C6C5        FDC-Status lesen, Drive Ready Write Prot?
C6C8
C6C9
C6CA        Statusreg.1 FDC
C6CD
C6CE        => Carry ist OK
```

```
C6CF
C6D0

****** Read/Write Sector, 512 Bytes
C6D1
C6D2        Akku an FDC ausgeben
C6D5
C6D6        512 Byte I/O-Buffer
C6D9        OP-Code Sector lesen?
C6DB        wenn nicht, zur Schreibschleife
C6DD        in die Leseschleife

****** Leseschleife, Daten lesen bis FDC Ende des Sectors meldet
C6DF        (bc) auf Datenreg. FDC
C6E0        Datenbyte lesen
C6E2        in Buffer (hl) speichern
C6E3        (bc) auf Status-Reg. FDC
C6E4        Bufferzeiger erhöhen
C6E5        Status-Byte holen
C6E7        Byte Ready-Meldung abwarten
C6EA        Ende Execution, Start Result?
C6EC        nächstes Byte holen
C6EE

****** Schreibschleife, Daten schreiben, bis FDC Ende meldet
C6EF        (bc) auf Datenreg. FDC
C6F0        Byte aus Buffer holen
C6F1        und auf Disc schreiben
C6F3        (bc) auf Status-Reg FDC
C6F4        Bufferzeiger erhöhen
C6F5        Status-Byte holen
C6F7        nächstes Byte erwünscht?
C6FA        Ende Execution, Start Result?
C6FC        nächstes Byte schreiben
C6FE

****** sucht den in d angegebenen Track
C6FF        Anzahl der Leseversuche
C702
C703        Track suchen
C706        => Track gefunden
C707        => 10 Versuche ohne Erfolg, dann READ FAIL
C709        Verbleibende Anzahl Versuche
C70A
C70C        Recalibrate-Flag setzen
C70E        gewünschter Track
C70F        Track 39
C711        Track suchen
C714        gewünschter Track
C715        neuen Versuch, Track zu finden
```

```
****** Recalibrate Flag auf Recalibrate setzen
C717
C718        Byte 17h in aktuellem Disc Parameter
C71A        Block bestimmen, Recalibrate Flag
C71D        auf Recalibrate setzen
C71F
C720

****** READ FAIL bei Track suchen ausgeben
C722
C723
C724        Drive nach c, Meldung ausgeben, CIR
C727
C728        Track in d suchen, call (hl)
C72A

******
C72B        Track in d suchen, CALL (HL)
C72E
C72F
C730        Sense Interrupt Status FDC
C733        Track in d suchen, CALL (HL)
C736
C737
C738        Track-Nummer
C739        40 Tracks
C73B
C73F
C740        Track in d suchen
C743
C744        Track in d suchen, CALL (HL)
C747
C748
C749        Track-Nummer
C74A        <>0?
C74B
C74E

******
C74F
C750        Track in d suchen
C753
C754        Track in d suchen
C757
C758
C759        CALL (HL)
C75C
C75E
C75F        Track in d suchen, CALL (HL)
C761
C762
```

```
****** ^87 Track im d-Register suchen
C763      Motor an, Add Ticker für Laufzeit
C766
C768
C769      Anzahl der Leseversuche
C76C
C76D      Byte 17h, Recalibrate-Flag
C76F      aus aktuellem Disc-Parameter-Block
C772      in Akku
C773
C774      => kein Recalibrate
C776      b:= Anzahl Leseversuche
C777      Statusregister FDC
C77A      Recalibrate Track 0
C77C      Akku an FDC ausgeben
C77F      Head Select/Unit Select
C780      Akku an FDC ausgeben
C783      40 mal 12 Millisekunden warten,
C785      dann FDC Interrupt Status lesen
C788
C78A      Byte 16h, aktueller DPB
C78C      das ist die aktuelle Tracknummer
C78F      löschen
C791      Byte 17 in Disc-Parameter-Block
C792      auf -1
C794      b:= Anzahl Leseversuche
C795
C796      erreichte Tracknummer
C797      mit gewünschtem Track vergleichen
C798      => Position erreicht
C79A      b:= Anzahl Leseversuche
C79B      Statusregister FDC
C79E      OP-Code Spur suchen
C7A0      Akku an FDC ausgeben
C7A3      Head Select/Unit Select
C7A4      Akku an FDC ausgeben
C7A7      gewünschte Track-Nummer
C7A8      Akku an FDC ausgeben
C7AB      erreichte Tracknummer abziehen
C7AC      Position erreicht
C7AE
C7B0
C7B1      warten, dann FDC Interrupt Status lesen
C7B4
C7B5      alles OK, Track gefunden
C7B7      neuen Versuch, evtl mit Recalibrate
C7B9      Versuche-Zähler abgelaufen?
C7BA      dann Fehlerbehandlung
C7BD      Sense Interrupt Status FDC
C7C0      neuen Versuch, evtl mit Recalibrate
```

```
****** Spur wurde gefunden
C7C2
C7C4
C7C5        Kennzeichen, daß alles OK
C7C6

****** wartet (Akku * 12)+ 16 ms, liest Int.Status FDC
C7C7
C7C8
C7CB        Warteschleife
C7CE
C7CF
C7D0        Warteschleife beendet?
C7D2
C7D5        Warteschleife
C7D8        OP-Code Sense Interrupt Status
C7DA        Akku an FDC ausgeben
C7DD        Int.Status FDC auslesen, Drive READY?

****** wartet Akku*1 Millisekunde
C7E0        Akku ist Schleifenzähler
C7E1        Verzögerungswert
C7E3        Akku auf Null zählen
C7E4        ca. 1 Millisekunde
C7E6        Schleifenzähler erniedrigen
C7E7
C7E8        evtl neue Schleife
C7EA

******
C7EB        HS/US-Buffer
C7EE        Drive-Nummer nach e
C7EF        Disc-Param.-Block-Wert 03h, Block Maske
C7F1        in Akku laden
C7F4        Blockmaske erhöhen
C7F5
C7F8        und merken
C7F9        Drive, Track und Recordnummer
C7FA        von be53h nach be5ah übertragen
C7FD
C7FF

******
C800
C806
C807        HS/US-Buffer
C80A        drei Bytes, Drive, Track und Sector
C80C        (hl) = (de)?
C80D
C80E        => ungleich, Fehler
C810
C811
C812        nächstes Byte
```

```
C814        Kennzeichen OK
C815

******
C816
C81A

****** noch Platz für diesen Record? sonst nächsten Track
C81B
C820
C821        Drivenummer nach e
C822
C823        Recordnummer im Track
C824        Recordnummer erhöhen
C825        Lo-Byte Records/Track
C826        aus Disc-Param.-Block in Akku
C829        max Recordnummer erreicht?
C82A        => noch nicht erreicht
C82C        Record im Track auf null setzen
C82E        hl = be5b, Tracknummer
C82F        Tracknummer erhöhen
C830
C831


******
C832
C838
C83B        Stack für Fehlerfall reparieren
C83C        => Fehler
C83D        kein Fehler, bc wieder auf Stack
C83E        prüft auf Sector Ende
C841
C842        => Record schon im Sector-Buffer
C844        Sectornr nach c, Sectorbuffer nach hl
C847        Sector lesen
C84A        evtl Fehler behandeln
C84B        ffh = kein Fehler bei Sector lesen
C84C
C850

******
C851
C853

******
C854        Sector lesen OK-Flag
C857
C858
C859        HS/US-Buffer
C85C
C85F        Drive-Nr nach e
C860        Akku mit alter Drivenummer laden
```

```
C861        sind die beiden gleich?
C862        wenn nicht, dann RET
C863        Tracknummer alt
C864        Tracknummer neu
C865
C866        sind die beiden gleich?
C867        wenn nicht, dann zurück
C868        Überlauf beim Record schreiben?
C86B
C86C        wenn Überlauf, dann zurück
C86D        alles OK
C86E


******
C86F        Sector lesen OK-Flag löschen
C872
C874
C875        (hl)=> Write Sector Flag
C876
C877
C878        => Flag ist 0, nicht schreiben
C879
C87A        tatsächliche Sectornr. berechnen
C87D        ^85 Sector schreiben

******
C880
C883        HS/US-Buffer
C886
C887
C888        Head Select/Unit Select-Wert nach e
C889
C88C
C88D        bestimmt Sectornummer, prüft auf overflow
C890
C891

****** bestimmt Sectornr., prüft auf Record-Overflow
C892        (Record-Nummer)
C893
C894
C896        Akku mit Anzahl Records/Sector laden
C899
C89A        Akku mit Record-Nummer laden
C89B
C8A0

****** errechnet tatsächliche Sectornummer
C8A2        Drivenr. und Track nach d und e
C8A6        Akku mit Disc-Param.-Block 0fh laden
C8A8        erste Sectornummer eines Tracks
C8AB        (gewünschte Sectornummer 0-8)
```

```
C8AE    ergibt Nr. des zu lesenden Sectors
C8AF    Sectornummer nach c
C8B0    Adresse Sectorbuffer bestimmen
C8B3    hl=hl+iy

****** Record in Sectorbuffer zum Schreiben übertragen
C8B6    alle Register retten
C8B7
C8B9
C8BA    Kennzeichen Record schreiben
C8BC    Read/Write Sector-Flag
C8BF    bc, de und hl versorgen
C8C2    KL LDIR, Record in Sector übertragen
C8C5    alle Register restaurieren

****** Record aus Sectorbuffer in Record-Buffer übertragen
C8C7    alle Register retten
C8C8
C8CA
C8CB    bc, de und hl versorgen
C8CE
C8CF    aus Sector in Record-Buffer übertragen
C8D1    alle Register restaurieren
C8D2
C8D5

****** de := Recordanfang in Sector-Buffer
C8D6    HS/US-Buffer
C8D9    Head Select/Unit Select nach e
C8DA    Akku mit Disc-Param.-Block-Wert 15h laden
C8DC    Anzahl der Records pro Sector
C8DF
C8E1
C8E2    (Record-Nummer)
C8E3    Recordlänge 128 Bytes
C8E6    Offset zum Record-Buffer
C8E9    nötige Korrektur
C8EA    errechnet die Adresse des nächsten
C8EB    Records im Sector-Buffer
C8EC
C8EE    Ergebniss nach hl
C8EF    de=de+iy, de => nächster Record
C8F2    Record-Bufferadresse nach hl
C8F5    Record-Größe 128 Bytes nach bc
C8F8
```

```
****** FDC-Statusregister auslesen, prüft ob DRIVE READY
C8F9        Result Phase FDC auslesen
C8FC        => kein Fehler aufgetreten
C8FD        FDC-Statusregister 0
C900        Drive Ready?
C902        => Drive ist READY
C903        Fehlermeldung 13, Disc is missing
C905

****** FDC-Statusregister auslesen, Diskette schreibgeschützt?
C907        Result Phase FDC, Drive READY?
C90A        => alles OK
C90B
C90C        FDC-Statusregister 1
C90F        Diskette schreibgeschützt?
C911        => ist nicht geschützt
C912        Fehlermeldung 12, Disc is write prot.
C914        Drive nach c, Meldung ausgeben, CIR
C917        'Ignore'
C918        'Cancel' => Abschluß
C91B        'Retry'

****** liest Bytes aus Result Phase des FDC in Buffer
C91C
C91D
C91E        Anzahl der gelesenen Bytes
C920        Result Phase Buffer Adresse
C923
C924        Status-Register FDC in bc
C926        warten bis
C928        Status-Byte ready
C92A        FDC-Datenregister-Adresse in bc
C92B        Interrupt-Statusregister
C92D        FDC-Statusregister-Adresse
C92E        nach (hl) abspeichern
C92F
C930        Zähler Anzahl geholter Bytes
C931
C933        kurze Warteschleife
C934
C936        Status-Byte lesen
C938        FDC-Busy-Bit
C93A        Kommando noch nicht beendet
C93C
C93D
C93E        Interrupt Code isolieren, Bit 6&7 von SR0
C940
C941        Anzahl Bytes Result Phase merken
C942
C943
```

```
C944        Fehler aufgetreten!
C945
C946        fehlerfreier Abschluß

****** Sense Interupt Status FDC
C947
C948        Statusregister FDC
C94B        OP-Code Sense Interrupt Status
C94D        Akku an FDC ausgeben
C950        Result Phase FDC auslesen
C953        Meldung 'INVALID COMMAND'?
C955        sonst noch mal versuchen
C957
C958

****** Akku mit Byte aus Disc-Parameter-Block laden und ausgeben
C959        Akku mit DPB-Wert (Akku) laden

****** prüft FDC, gibt ggf. Akku an FDC aus
C95C        auszugebenden Wert zwei mal auf Stack
C95D
C95E        Bit 7,Request for Master prüfen
C960
C961        warten, bis neues Byte gefordert wird
C963        Bit 6,Datenrichtung zum FDC
C964        Test auf Ein- oder Ausgabe
C966        kein Byte an Controller schicken,
C967        FDC will Byte an Prozessor schicken
C968

****** Akku wird an FDC übergeben
C969        auszugebenden Wert vom Stack
C96A        Datenregister FDC in bc
C96B        Byte an FDC ausgeben
C96D        Status-Reg. FDC in bc
C96E
C970        kurze Warteschleife
C971        Akku von 5 auf Null zählen
C972
C974        Wert im Akku restaurieren
C975

****** Motor ein, Stack manipulieren, (hl) auf I/O-Buffer
C976        Zwischenspeicher
C979        RETURN-Adresse vom Stack holen
C97A        bc und de retten
C97B
C97C        Stack-Pointer merken, sicher ist sicher
C980        RETURN-Adresse wieder auf den Stack
C981
C984        RETURN wieder nach hl, C9AD auf Stack
C985        endgültig RETURN-Adresse auf Stack
C986        de, bc, af retten, ein RET nach
```

```
C987        dieser Routine führt nach C9AD!
C988        nicht ganz einfach, nicht war?
C989        Del Ticker
C98C        Motor Flag
C98F        testen
C990        Motor läuft bereits
C992        Portadresse Motorsteuerung
C995
C997        Motor einschalten
C999        Anzahl der Ticks
C99D        add Ticker aufrufen, Motorhochlaufzeit
C9A0        Motor Flag
C9A3        testen
C9A4        warten bis Motor-Flag <> Null
C9A6        af, bc, de wieder vom Stack
C9A7
C9A8
C9A9        (hl) zeigt auf I/O-Buffer
C9AC        der nächste RET geht nach C9AD!

******
C9AD        bei C97C hier vermerkt
C9B1
C9B2        Anzahl der Ticks
C9B6        Add Ticker aufrufen
C9B9
C9BA        bc, de und hl wurden bei C979 bis
C9BB        C97B gePUSHt
C9BC
C9CA
C9CB        (hl) => 0BE4B
C9CC

******
C9CD        Adresse Tick Block
C9D0        Reload Count
C9D3        KL ADD TICKER

****** Tick  Routine
C9D6        Motor Flag
C9D9        ist 0 oder ffh
C9DA        wird ffh oder 0
C9DB        abspeichern
C9DC        wenn Motor Flag Null
C9DD        => Motor läuft
C9DF        sonst Adresse Tick Block
C9E2        KL DEL TICKER, Motorticker ausklinken
```

```
******
C9E5
C9E8
C9EA    Motor-Port
C9ED    Motor-Flip-Flop rücksetzen, Motor aus
C9EF
C9F0    Motor Flag löschen
C9F3

****** Disc Parameter Header und DP Blöcke anlegen
C9F4    Offset für DPH Drive B
C9F7    Offset für DPB Drive B
C9FA    DPH und DPB Drive B initialisieren
C9FD    Offset für DPH Drive A
CA00    Offset für DPB Drive A
CA03    de=de+iy, Anfang des DPB
CA06    Adresse merken
CA0A    und auf Stack merken
CA0B    hl=hl+iy, Anfang des DPH
CA0E    ebenfalls merken
CA11    und auf den Stack
CA12    Adresse des Standard-DPBs
CA15    25 Bytes
CA18    an die richtige Adresse übertragen
CA1A    Beginn der Checksum-Area in bc merken
CA1B
CA1C    Anfang des DPH, das ist XLT
CA1D    eventuelle Umsetzung des SKEW-Faktors
CA1F    wird nicht verwendet, also 0
CA20
CA22
CA25    (hl)=> DIRBUF
CA26    Offset der 128 Byte für DIR-Buffer
CA29    de=de+iy, Adresse des DIR-Buffers
CA2C    in DIRBUF eintragen
CA2D
CA2F
CA30    Anfang der DPBs
CA31    im Header eintragen
CA32
CA33
CA34    (hl)=> CSV, Checksum Vector
CA35    (bc)=> adresse der Checksum Area
CA36    im Header eintragen
CA37
CA38    (hl)=> ALV, Allocation Vector
CA39    in de zwischenspeichern
CA3A    Offset zwischen Checksum Area und
```

```
CA3D        Allocation Area
CA3E        (hl)=> ALV
CA3F        Allocation Area im Header eintragen
CA40
CA42

****** Standard DPB, Disc Parameter Block
CA43        SPT Records per Track
CA45        BSH Block-Shift
CA46        BLM Block Mask
CA47        EXM Extent Mask
CA48        DSM Anzahl freie Blöcke -1
CA49        DRM Anzahl Direinträge -1
CA4B        AL0 Directory-Belegung
CA4D        CKS Anzahl zu prüfender Einträge
CA4F        OFF Spuroffset für System-Spuren

****** Erweiterungen der DPBs, nicht im Standard-CP/M vorgesehen
CA4B        Sector-Offset für Formaterkennung
CA53        Anzahl Sectoren/Track
CA53        Länge GAP3 bei Read/Write
CA53        Länge GAP3 bei Formatieren
CA53        Fillerbyte beim Formatieren
CA53        Bytes/Sector für FDC, entspricht 512 Bytes
CA53        Anzahl Records/Sector
CA53        drei Zwischen-Speicher
CA53
CA53

****** Akku mit DPB-Wert (A890h+(Drive*40h)+Akku) laden
CA5C
CA5D
CA60        Akku =(Tabelle+(drive*40h)+Akku)
CA61
CA62

****** hl = Pointer auf aktuelle DPB+Akku
CA63
CA64        Zeiger auf DPB Drive 0
CA67        Head Select/Unit Select -1
CA68        Offset der DPBs Drive 0 und 1
CA6B        Sprung wenn Drive 0 aktuell
CA6D        Anfang Tabelle Drive 1 bestimmen
CA6E        Akku zeigt auf gewünschtes Byte
CA6F        d=0, e=gewünschtes Byte
CA70
CA71

****** ^81h Fehlermeldungen Disc Controller on/off
CA72        alten Wert in L merken
CA75        neuen Wert eintragen
CA78        alten Wert in Akku übergeben
CA79
```

```
****** wenn enabled Fehlermeldung ausgeben
CA7A        Fehlernummer im Akku
CA7B        Flag für Fehlermeldungen
CA7E        prüfen
CA7F        => keine Ausgabe erlaubt
CA81        Fehlernummer wiederholen
CA82        Drivenummer für Ausgabe nach C
CA83        System-Meldung ausgeben, C, I or R

****** Abschluß keine Ausgabe erlaubt
CA86
CA87        Akku und Flags löschen
CA88

****** bis CA90 unbenutzt
CA89
CA8F

****** BC = BC + IY
CA90
CA92        iy nach hl, hl auf Stack
CA93        bc und hl addieren
CA94        Ergebnis nach bc
CA95
CA96        hl restaurieren
CA97

****** DE = DE + IY
CA98
CA9A        iy nach hl, hl auf Stack
CA9B        de und hl addieren
CA9C        Ergebnis nach de
CA9D        hl restaurieren
CA9E

****** HL = HL + IY
CA9F
CAA0
CAA2        iy nach de
CAA3        hl und de addieren
CAA4        de restaurieren
CAA5
****** Klein- in Groß-Buchstaben wandeln
CAA6        'a' oder größer?
CAA8
CAA9        'z' oder kleiner?
CAAB
CAAC        in Großbuchstaben wandeln
CAAE
```

```
****** Speicher löschen (de) bis (de)+(bc)
CAAF      Akku löschen
CAB0      Speicher löschen
CAB1      nächste Adresse
CAB2      Anzahl erniedrigen
CAB3      Test auf bc = 0
CAB4
CAB5      löscht (de) bis (de+bc)
CAB7

****** Fehler in A ausgeben, dann 'IGN, RET, CHAN' prüfen
CAB8      Systemmeldung in a ausgeben
CABB      Systemmeldung 20
CABD      suchen und ausgeben
CAC0      KM READ CHAR
CAC3
CAC5      TXT CUR ON
CAC8      KM WAIT CHAR
CACB      wandelt in Großbuchstaben
CACE      'C' = Chancel
CAD0      => Z=1,C=0
CAD2      'I' = Ignore
CAD4      => Z=1,C=1
CAD5
CAD7      'R' =Retry
CAD9      => Z=0,C=0
CADB      Zeichen 'BELL'
CADD      TXT OUTPUT, einmal piepen
CAE0      neue Eingabe abwarten

****** Retry-Entry, Flags löschen
CAE2


****** Chancel- und Ignore-Entry, Zeichen im Akku ausgeben
CAE3      TXT OUTPUT
CAE6      TXT CUR OFF
CAE9      'CR/LF' ausgeben

****** SYSTEM-MESSAGE, Systemmeldung suchen und ausgeben
CAEB
CAED
CAEE      Bit 7 der Fehlernummer löschen
CAF0      hl zeigt auf System- und Fehlermeldungen
CAF3      Fehlernummer nach b
CAF4      eins erhöhen, wird sofort
CAF5      beim DJNZ wieder erniedrigt
```

```
****** Meldungen bis zur gewünschten überlesen
CAF7        ein Zeichen der Meldung in Akku
CAF8        Zeiger erhöhen
CAF9        testet auf Ende einer Meldung
CAFA        wenn nicht Ende, weiter suchen
CAFC        b<>0 nächste Meldung überlesen

****** gewünschte Meldung ausgeben
CAFE        ein Zeichen der Meldung in Akku
CAFF        Zeiger erhöhen
CB00        Ende der Meldung erreicht?
CB02        wenn ja, dann Sprung
CB04
CB06
CB07        Zeichen in A weiter prüfen und ausgeben
CB0A
CB0C
CB0D        nächstes Zeichen holen

****** Meldung ausgegeben, fertig
CB0F
CB12

****** Zeichen weiter prüfen und ausgeben
CB13        Zeichen im Akku testen
CB14        kleiner 80h, dann zur Ausgabe
CB17        String für Drive-Nummer
CB19        Drive-Nummer in A/B umrechnen und ausgeben
CB1B        String für numerische Variable
CB1D        Variable bestimmen und ausgeben
CB1F        String für Filenamen
CB21        Expansionstring Systemmeldung ausgeben
CB23        Filename ist acht Zeichen lang
CB25        im Speicher localisieren und ausgeben
CB28        "."
CB2A        ausgeben
CB2D        Extension ist drei Bytes lang
CB2F        de zeigt auf Position im Speicher
CB30        Zeichen in Akku
CB31        Bit sieben löschen
CB33        ausgeben
CB36        Anzahl - 1, nächstes Zeichen
CB38

******
CB39        de zeigt auf Filennamen
CB3A        " ", Leerzeichen
CB3C
CB48
CB49        Offset für ASCII-Ziffer
CB4B        ausgeben
```

```
******
CB4D
CB56

******
CB58
CB59        Null im Akku?
CB5A        dann Blank ausgeben
CB5C        Offset für ASCII-Ziffer
CB5E
CB5F        ausgeben

******
CB61        Drivenummer 0 oder 1
CB62        'A' addieren
CB64        'A' oder 'B' ausgeben


****** Zeichen im Window an Curs-Pos ausgeben
CB66        Zeichen zwischenspeichern
CB67        Leerzeichen?
CB69        dann direkt ausgeben
CB6B
CB6C
CB6D        TXT GET WINDOW
CB70        TXT GET CURSOR
CB73        d enthält rechte Window-Spalte
CB74
CB77
CB79        h enthält Cursor-Spalte
CB7A
CB7E
CB7F        'CR/LF' ausgeben

******
CB82        Zeichen wieder in Akku
CB83        TXT OUTPUT

****** Fehler-Meldungen / Systemmeldungen

****** Systemmeldung 0
CB86        CR/LF

****** Systemmeldung 1
CB89          drei Blanks ausgeben

****** Systemmeldung 2
CB8D          'num. Variable'K

****** Systemmeldung 3
CB90          'CR/LF''CR/LF' 'num. Variable K'  free
```

```
****** Systemmeldung 4
CB99        'CR/LF'Bad command'CR/LF'

****** Systemmeldung 5
CBA7        'CR/LF Filename' already exists


****** Systemmeldung 6
CBB8        'CR/LF Filename' not found

****** Systemmeldung 7
CBC4        'CR/LF Drive A/B' directory 'full'

****** Systemmeldung 8
CBD1        'CR/LF Drive A/B Disc' 'full CR/LF'

****** Systemmeldung 9
CBD4        'CR/LF Drive A/B Disc' changed,
CBDC        closing 'Filename' 'CR/LF'

****** Systemmeldung A
CBE9        'CR/LF Filename' is 'read' only

****** Systemmeldung B
CBF5        'Filename'

****** Systemmeldung C
CBF7        'CR/LF Drive A/B' user 'num. Variable'

****** Systemmeldung D
CBFF        ...^C

****** Systemmeldung E
CC05        'CR/LF failed to load' CP/M 'CR/LF'

****** Systemmeldung F
CC0C        'CR/LF failed to load' boot sector 'CR/LF'

****** Systemmeldung 10
CC1A        'CR/LF Drive A/B' 'read' ' fail CR/LF'

****** Systemmeldung 11
CC1E        'CR/LF Drive A/B' 'write' ' fail CR/LF'

****** Systemmeldung 12
CC22        'CR/LF Drive A/B disc is 'write'
CC2A        protected 'CR/LF'


****** Systemmeldung 13
CC33        'CR/LF Drive A/B disc 'missing 'CR/LF'
```

```
****** Systemmeldung 14
CC3D        'CR/LF' Retry, Ignore or Chancel?

****** Systemmeldung 15
CC58        'CR/LF' Drive 'A/B':

****** Systemmeldung 16
CC63        'CR/LF' Failed to load

****** Systemmeldung 17
CC74        'CR/LF' 'CR/LF'

****** Systemmeldung 18
CC77        'CR/LF Drive A/B' disc

****** Systemmeldung 19
CC7E        fail 'CR/LF'

****** Systemmeldung 1A
CC85        full 'CR/LF'

****** Systemmeldung 1B
CC8B        'CR/LF' 'Filename'

****** Systemmeldung 1C
CC8F        write

****** Systemmeldung 1D
CC95        read

****** unbenutzt
CC9A FF       RST   38H
CC9F FF       RST   38H

****** Alle Tape-Vektoren patchen für Disc
CCA0
CCA1        Drive und User auf Default A0
CCA4
CCA7        Akku = ffh
CCA8        OPENIN-aktiv Flag auf inaktiv ffh
CCAB        OPENOUT-aktiv Flag auf inaktiv ffh
CCAE
CCB2        Tape-Vektoren
CCB5        in Sicherheit bringen
CCB8        de=de+iy, mempool + 164h
CCBB        13*3 Bytes = 13 Vectoren
CCBE
CCC0
CCC1        0CD30H ist die Einsprungadresse für
CCC3        alle gepatchten CAS-Entries
CCC4
CCC6
CCC7        KL ASC CURR SELECTION, Nummer des
```

```
CCCA      Floppyrom als drittes Byte für RST
CCCB      Code für Return
CCCD
CCD0

****** DISC
CCD1      Disc Out
CCD4      Fehler aufgetreten dann RET

****** DISC IN
CCD5      Cass In Open
CCD8      Cass In Open und die nächsten
CCDA      sechs Entries patchen
CCDD
CCDE      Catalog
CCE1
CCE2      Catalog patchen

****** DISC OUT
CCE4      Cass Out Open und die folgenden
CCE7      vier Entries

******
CCE9      folgen Parameter?
CCEA      wenn ja dann Sprung
CCEC      sonst gewünschte Vektoren patchen
CCEF      de=de+iy, discmem + 18bh
CCF2      Restart 3
CCF4
CCF5      die Entries zeigen alle nach A88Bh
CCF6
CCFC

****** TAPE, Tape-Vectoren restaurieren
CCFD      Tape Out restaurieren
CD00      Fehler aufgetreten, dann RET

****** TAPE IN
CD01
CD04
CD07      7 Vectoren alle Tape In
CD0A
CD0D      Fehler aufgetreten
CD0E
CD11
CD14      ein Vector Cass Catalog
CD16
```

```
****** TAPE OUT
CD18
CD1B
CD1E        5 Vectoren alle Tape Out
CD21        folgen Parameter?
CD22        dann Fehler ausgeben
CD24        hl=hl+iy
CD27
CD29        Carry setzen als Kennzeichen alles OK
CD2A

******
CD2B        Systemmeldung 4, 'BAD COMMAND'
CD2D        suchen und ausgeben


****** wird von allen CAS-Entries mittels RST3 angesprungen
CD30        Start des Memory für Disc nach iy
CD34        nötig für Benutzung des
CD35        alternativen Registersets
CD36
CD37        Inhalt von c ist variabel
CD38        Rücksprungadresse nach de
CD39        zwei weitere RETs POPen
CD3A
CD3B        dieser RET muß hochrücken
CD3C        bc und originaler RET wieder auf Stack
CD3D
CD3E        c und b restaurieren
CD3F
CD41        RET-Adresse um 10D2h erhöhen
CD44        und als neue RET-Adresse auf den Stack
CD45        RET zeigt dann in folgende Tabelle
CD46        originalen Registerset wiederholen
CD47
CD48        jetzt dürfen die INTs wieder kommen
CD49        da steht ein RET!

****** Jumpblock für die gepatchten CAS/DISC-Entries
CD4C        DISC IN OPEN
CD4F        DISC IN CLOSE
CD52        DISC IN ABANDON
CD55        DISC IN CHAR
CD58        DISC IN DIRECT
CD5B        DISC RETURN
CD5E        DISC TEST EOF

CD61        DISC OUT OPEN
CD64        DISC OUT CLOSE
CD67        DISC OUT ABANDON
CD6A        DISC OUT CHAR
CD6D        DISC OUT DIRECT
```

```
CD70        DISC CATALOG

****** Stackpointer erhöhen und nach discmem+6
CD73        Wird zur Korrektur des Stacks
CD76        benötigt

****** Stackpointer nach discmem+6
CD77
CD78        zwei CALLs und der PUSH HL = 6 Bytes
CD7B        Stack entsprechend korrigiert nach hl
CD7C        und in discmem+6
CD7F        und discmem+7 merken
CD82
CD83

****** Stack sichern, wenn OPENIN nicht aktiv, dann Abbruch
CD84        Stackpointer sichern
CD87
CD88        OPENIN-aktiv Flag
CD8B

****** Stack sichern, wenn OPENOUT nicht aktiv, dann Abbruch
CD8D        Stackpointer sichern
CD90
CD91        OPENOUT-aktiv Flag
CD94        ist File nicht offen?
CD96        => Abbruch
CD98        bei Bedarf Login, Format ermitteln
CD9B
CD9C

****** Abbruch, wenn OPENIN aktiv
CD9D        OPENIN-aktiv Flag
CDA0

****** Abbruch, wenn OPENOUT aktiv
CDA2        OPENOUT-aktiv Flag
CDA5        Stackpointer sichern
CDA8        Flag ist ffh, wenn nicht aktiv
CDA9        => war nicht aktiv, alles OK
CDAA        Fehlernummer in Akku
CDAC        Carry-Flag löschen, Kennzeichen Fehler
CDAD        Stack restaurieren, Kommando abbrechen

****** Bad command ausgeben, Kommando abbrechen
CDAF        Systemmeldung 4, 'Bad command'
CDB1
CDB4        Fehlernummer in Akku
CDB6        bisher keine Fehlermeldung ausgegeben
CDB8        Carry löschen, Kennzeichen Fehler
```

```
****** Kommando abbrechen
CDB9      Stackpointer restaurieren
CDBC
CDBF
CDC0      und zurück

****** prüft Akku auf 2, wenn nicht, Abbruch und 'Bad Command'
CDC1

****** prüft Akku auf 1, wenn nicht, Abbruch und 'Bad Command'
CDC2
CDC3      => Akku ist Null
CDC4      Bad command ausgeben, Kommando abbrechen

****** holt Länge String nach b, Adresse String nach hl
CDC7      einen Parameter nach hl holen
CDCA      Länge des Strings
CDCB      (hl) => Adresse des Strings
CDCC      LD HL,(HL)

****** einen Parameter von Befehlserweiterung nach hl holen
CDCF      Lo-Byte und
CDD2      Hi-Byte des Übergabe-Parameters nach hl
CDD5      ix auf evtl. nächsten Parameter
CDD7
CDD9

****** |A:
CDDA      Akku auf 0, Wert für Drive A
CDDB

****** |B:
CDDD      Akku auf 1, Wert für Drive B
CDDF      Stackpointer sichern
CDE2      Wert an DOS übergeben

****** |DRIVE
CDE4      Stack sichern
CDE7      ein Folge-Parameter, sonst 'Bad command'
CDEA      Parameter holen
CDED
CDEE      Bad command ausgeben, Kommando abbrechen
CDF1      gew. Drive-Kennzeichen (A/B) in Akku
CDF2      wandelt in Großbuchstaben
CDF5      ergibt 0 oder 1
CDF7      Login
CDFA      Drive-Nummer an DOS übergeben
CDFD

****** |USER
CDFE      Stack sichern
CE01      ein Folgeparameter, sonst 'Bad command'
CE04      Parameter nach hl holen
```

```
CE07        maximale Usernummer + 1
CE0A        de = hl?, Usernummer legal?
CE0D        zu hoch, 'Bad command', Abbruch
CE10        Usernummer an DOS übergeben
CE13

******
CE14        erstes Zeichen Expand. Filename, Drivenr.
CE15
CE19
CE1A        Drive-Nr. angem. Drive
CE1B        ffh = OPEN auf angemeldetem Drive aktiv
CE1D        OPENIN-aktiv Flag
CE20        auf diesem Drive?
CE21        => OPENIN aktiv
CE23        OPENOUT-aktiv Flag
CE26        auf diesem Drive?
CE27        => OPENOUT aktiv
CE29        00h = kein OPEN auf angem. Drive aktiv
CE2B
CE2C
CE2D        prüft Drivenummer, ermittelt Disc-Format
CE30
CE31
CE32        prüft hl auf 0000
CE33        wenn hl = 0000, dann Drive nicht READY
CE34        Bad command ausgeben, Kommando abbrechen
CE37        (hl) => Disk-Parameter-Header (a910/a920)
CE3A        nach iy+3/iy+4
CE3D        Flag, ob OPEN auf angem. Drive aktiv
CE40        Drivenummer (HS/US)
CE43
CE47

****** kopiert expand. Filename in OPENIN-Header-Block
CE48        Offset zu OPENIN-Header-Block
CE4B
CE56

****** kopiert expand. Filename in OPENOUT-Header-Block
CE57        Offset zu OPENOUT-Header-Block
CE5A
CE5B
CE5C        hl=hl+iy, hl=> Header-Block
CE5F
CE61
CE62        (de) => Adresse des User-Buffers
CE63
CE64        nach Pointer Anfang User-Buffer
CE65
CE66        Vector in User-Buffer
CE67        auf Anfang setzen
CE68
```

```
CE69        (hl)=> Anfang Filename im Header
CE6A
CE6B        (bc) => EFN ab Usernummer
CE6C        Anzahl der zu löschenden Bytes
CE6F
CE70        löscht (de) bis (de+bc), Rest des Header
CE73        (bc) => EFN ab Usernummer
CE74        nach hl übertragen
CE75
CE76        Adresse Filename im Header nach de
CE77
CE78        Länge des Filenamens incl Drive/User
CE7B        in Header-Block übertragen
CE7D        Adresse Filename im Header nach hl
CE7E        Adresse User-Buffer nach de
CE7F
CE83
CE84        Kennzeichen Filetyp 'unprot. ASCII'
CE86
CE88
CE89        Adresse User-Buffer
CE8A
CE91

****** Zwei-Byte Prüfsumme über Header (43h Bytes)
CE92        hl => Headeranfang
CE93        Prüfsumme auf 0 setzen
CE96        Hi-Byte von de auf 0
CE97        Anzahl Bytes
CE99        hl = Zeiger in Header, Prüfsumme auf Stack
CE9A        ein Zeichen in Akku
CE9B        nächstes Zeichen im Header
CE9C        hl = Prüfsumme, Zeiger in Header auf Stack
CE9D        Akku nach e, de = Zeichen in 16 Bit
CE9E        zu Prüfsumme addieren
CE9F        noch ein Zeichen? =>
CEA1        Prüfsumme nach de
CEA2        hl => Headeranfang
CEA3

******
CEA4
CEAC

****** DISC IN OPEN
CEAF        Stackpointer merken
CEB2        2K-Bufferadresse
CEB3        Filenamen auf Gültigkeit prüfen
CEB6        Disk-Parameter-Header nach hl, ggf Login
CEB9
CEBC        (bc)=> expand.Filename
CEBD        (hl)=> erstes Zeichen Extension
CEBE        erstes Zeichen Extension =ffh?
```

```
CEBF      => Filename ohne Extension eingegeben
CEC1      Filename im Dir suchen
CEC4      => File nicht gefunden, Abbruch
CEC7

****** Filename ohne Extension angegeben
CEC9      Extension 3 Blanks eintragen
CECC      Filename ohne Extension auf Disc?
CECF      => gefunden
CED1      Extension 'BAS' eintragen
CED4      als Basic-File gespeichert?
CED7      => gefunden
CED9      Extension 'BIN' eintragen
CEDC      als Binär-File gespeichert?
CEDF
CEE0      File nicht gefunden, Extension 3 Blanks
CEE3
CEE4      => Abbruch, 'File not found'

****** Filename im Directory gefunden
CEE7      2K-Buffer-Adresse
CEE8      Filename in OPENIN-Header kopieren
CEEB      Adresse Header-Anfang
CEEC
CEEF      de=de+iy, Adresse OPENIN-FCB
CEF2
CEF3      Drivenummer
CEF4      in OPENIN-FCB
CEF5      Anzahl Character im Input-File auf 0
CEF8
CEFB      hl=hl+iy, Adresse Record-Buffer
CEFE      Record in Record-Buffer
CF01      => Feher aufgetreten
CF03      (hl)=> Anfang Record-Buffer
CF04      (de)=> Filename im OPENIN-FCB
CF05      Prüfsumme 43h Bytes des Record nach de
CF08      ld hl,(hl), evtl. gespeicherte Prüfsumme
CF0B      hl = de? wenn ja, gel. Record ist Header
CF0E
CF0F
CF10      => Prüfsumme <>, ASCII-File!
CF12
CF15      de=de+iy, OPENIN-Header-Block+5
CF18      Anzahl Headerbytes
CF1B      in OPENIN-Header-Block übertragen
CF1D

****** kein ASCII-File als Input-File
CF1F      Anzahl Character im Input-File auf 0
CF22
CF27
CF28      Adresse, von wo File ursprünglich
CF29      geschrieben wurde, nach de
```

```
CF2A
CF2C
CF2D      Länge des Files nach bc
CF2E
CF30
CF31      Kennzeichen OPENIN OK
CF32      an Betriebssystem CPC kein Fehler
CF33      Filetyp des eröffneten Files
CF36

****** DISC OUT OPEN
CF37      Stackpointer merken
CF3A      2K-OPENOUT-Buffer-Adresse
CF3B      prüft Filenamen, ob gültig, legt EFN an
CF3E      Disk Parameter Header nach hl, ggf. Login
CF41      2K-OPENOUT-Buffer
CF42      EFN in OPENOUT-Header-Block kopieren
CF45      (hl) => OPENOUT-Header +5, Usernummer
CF46      Extension '$$$' in OHB eintragen
CF49      File schon auf der Diskette?
CF4C      Adresse OPENOUT-Header nach hl
CF4D
CF4E      (hl) => Drive-Nummer
CF4F
CF52      de=de+iy, OPENOUT-Filecontrol-Block
CF55      Länge Filename mit Drive und User
CF58      nach OPENOUT-FCB übertragen
CF5A      Länge Blockbelegung im Dir(10h)+Pointer(7)
CF5D      löscht Rest hinter Filename im FCB
CF60      Header-Block +5
CF61      Kennzeichen OPENOUT OK
CF62      an Betriebssystem CPC OPENOUT OK
CF63

****** DISC IN CHAR
CF64      alle Register retten
CF65
CF66
CF67      ein Zeichen aus OPENIN-Buffer holen
CF6A      Regsiter restaurieren
CF6B
CF6C
CF6D      Kennzeichen Fehler aufgetreten
CF6E      EOF-Kennzeichen gelesen?
CF70      Kennzeichen alles OK
CF71      => nicht EOF
CF72      sonst Carry löschen
CF73      und zurück
```

```
****** holt ein Zeichen aus geöffnetem OPENIN-File
CF74        Stack sichern, OPENIN-Flag prüfen
CF77        Disc-Mempool
CF79        nach de
CF7A
CF7D        (hl)=> Kennzeichen In Char(1)/In Direkt(2)
CF7E        Kennzeichen in Akku
CF7F        War bisher Disc In Direkt aktiv?
CF81        dann Fehler, Abbruch des Kommandos
CF84        Kennzeichen Disc In Char eintragen
CF86        Prüfen, ob Zeichen im Buffer
CF89
CF8A        Drei Byte File-Characterzähler
CF8B
CF8E
CF8F        => kein Zeichen im Buffer
CF91
CF98
CF99        2K-OPENIN-Buffer füllen
CF9C        Anzahl in OPENIN-Buffer gelesene Zeichen
CF9D        auf 0 prüfen
CF9E
CF9F        => Fehler, keine Zeichen im Buffer
CFA1        Anzahl gelesene Bytes nach bc
CFA2
CFA3
CFA4        ein Zeichen wird gelesen, also abziehen
CFA5        im Buffer verbleibende Anzahl
CFA6        Zeichen merken
CFA7
CFBA
CFBB        Pointer in OPENIN-Buffer nach de
CFBC
CFBE
CFBF        LD A,(HL), Zeichen aus Buffer holen
CFC0
CFC1        OPENIN-Buffer-Pointer erhöhen
CFC2        und merken
CFC3
CFC4
CFC5        Kennzeichen, ein Zeichen geholt
CFC6

****** Fehlerabschluß
CFC7        Fehler-Code
CFC9        Carry löschen
CFCA
```

```
******
CFCB
CFD1
CFD2    ld hl,(hl), Bufferadresse
CFD5    auf Stack merken
CFD6    16 Records
CFD9    in OPENIN-Buffer laden, wenn vorhanden
CFDC
CFDE    Anzahl gelesene Records bestimmen
CFDF    Anzahl nach b
CFE0
CFE2    Anzahl gelesene Bytes bestimmen
CFE4    Ergebnis nach bc
CFE6    OPENIN-Buffer-Pointer
CFE7    (hl)=> Anzahl gelesene Bytes
CFE8    merken
CFE9
CFEE
CFEF    OPENIN-Buffer-Pointer merken
CFF0
CFF4

****** DISC IN DIRECT
CFF5    Stack sichern, OPENIN-Flag prüfen
CFF8    Ladeadresse
CFF9
CFFC    hl=hl+iy
CFFF    Kennzeichen In Char(1)/In Direct prüfen
D000    wenn Disc in Char,
D002    Fehler, Abbruch des Kommandos
D005    Kennzeichen Disc In Direct eintragen
D007
D00A
D00B    Anzahl Character nach de
D00C
D00F
D010    Ladeadresse nach hl
D011    und austauschen
D012    2^7=128
D014    teilt Anzahl Character/128
D017    Ergebnis Anzahl Records nach bc
D018
D019    Ladeadresse nach hl
D01A    errechnete Anzahl records laden
D01D
D01E    => Fehler
D020
D027
D02A    hl=hl+iy
D02D
D045
D046    ld hl,(hl)
```

```
******
D049

****** Records in OPENIN-Buffer, Anzahl Records in bc
D04B        Record, evtl von Disc, in Buffer
D04E        => Fileende oder sonstiger Fehler
D04F
D052        de=de+iy, Adresse Filetyp im OPENIN-Header
D055        Filetyp prüfen
D056
D057        => Carry nur bei 'Protected File'
D05A        Recordlänge
D05D        Buffer-Pointer erhöhen
D05E        Anzahl noch zu lesender Records
D05F        Anzahl auf 0 prüfen
D060
D061        => noch nicht alle gelesen
D063        alles OK, alle Records gelesen
D064

****** DISC TEST EOF
D065        Disc In Char aufrufen
D068        RET bei EOF, sonst Char. zurück in Buffer

****** DISC RETURN
D069
D06C
D06F        hl=hl+iy
D072
D08E

****** DISC OUT CHAR
D08F        Stack merken, OPENOUT-Aktiv-Flag prüfen
D092
D094
D095        Akku enthält das Zeichen
D096        Disc-Mempool nach de
D098
D099
D09C        Disc-Out-Mode Flag (Char/Direkt)
D09D        aus Header in Akku
D09E        bisher Disc-Out-Direkt?
D0A0        => Abschluß des Kommandos
D0A3        Disc-Out-Char-Kennzeichen eintragen
D0A5
D0A8        (hl):= Anzahl Character im OPENOUT-Buffer
D0A9
D0AA        ld hl,(hl), Anzahl der Zeichen nach hl
D0AD
D0B1
D0B2        => mehr als 2K, Daten auf Disc schreiben
D0B5
D0B6
```

D0B7 Anzahl der Zeichen im User Buffer erhöhen
D0B8 zwei-Byte-Wert
D0B9 Hi-Byte nur bei Bedarf
D0BB erhöhen
D0BC
D0BF
D0C0 File Character Counter erhöhen
D0C3
D0C6 (hl) => Vector auf Pointer in User Buffer
D0C7
D0C8 bc => Pointer in User Buffer
D0C9
D0CB
D0CC Zeichen im Akku in User Buffer ablegen
D0CD Pointer in User Buffer erhöhen
D0CE bei Bedarf auch Hi-Byte
D0D0
D0D4
D0D5 Kennzeichen alles OK
D0D6 Meldung an Betriebssystem CPC alles OK
D0D7

****** DISC OUT DIRECT
D0D8 Stack merken, OPENOUT-Aktiv-Flag prüfen
D0DB Akku = Filetyp
D0DC Adresse, ab wo geschrieben werden soll
D0DD Länge des Datenblocks
D0DE
D0E1 hl=hl+iy, Disc-Out-Mode (Char/Direkt)
D0E4 in Akku
D0E5 war bisher Disc-Out-Char aktiv?
D0E7 dann Fehler, Abschluß des Kommandos
D0EA Direkt-Kennzeichen eintragen
D0EC
D0EF
D0F0 bc = Entry-Adresse
D0F1
D0F2
D0F3 bc = Länge Datenblock
D0F4
D111
D112 Filetyp in Akku
D113
D116
D117 Filetyp in OPENOUT-Header-Block

****** 2K (User Buffer bei OUT CHAR) auf Diskette schreiben
D118 Disc Mempool
D11A nach de
D11B
D11E
D11F Header Byte 'First Block'

```
D120
D121    => nicht der erste Block
D123
D126    (hl):= Filetyp
D127    Filetyp in Akku,
D128    Hi-Nibble ist unwichtig
D12A    Kennzeichen 'unprotected ASCII'?
D12C    => ist der Fall
D12E
D131    ,hl=> OPENOUT-File-Control-Block
D132    Disc-Mempool
D133
D134    Anzahl Records +1 für Header-Record
D137    Anzahl Records im FCB prüfen
D13A
D13F
D140    Block Char Counter,
D141    Anzahl der Zeichen nach de
D142
D143
D146    hl => User Buffer Vector
D147    ld hl,(hl), hl => User Buffer
D14A
D14B    Buffer in Records, dann auf Disk
D14E    bc => User Buffer
D14F    hl => Character Count
D150    auf 0 setzen
D152
D160
D161    Kennzeichen OK
D162
D163

******
D164    Anzahl Character im Block
D165    Anzahl Records/Block
D167    Anzahl nach de
D168    teilt Anzahl Char.s /128
D16B    Ergebnis benötigte Records nach de
D16C    und bc
D16D
D16E    Records übertragen
D171    Anzahl Character
D172    Lo-Byte in Akku
D173    auf einen Record begrenzen
D175    wenn Null, dann keine weiteren Bytes
D176    sonst Rest des Files in neuen Record
D177    dann Hi-Byte = Null
D179
D17C    de=de+iy
D17F    Record Buffer
D180    KL LDIR, überträgt letzten Rec. in Buffer
D183    EOF-Kennzeichen
```

```
D185        anhängen
D186        hl => Record Buffer
D187        bc = Record-Counter erhöhen
D188

****** Records (Anzahl in bc) in File schreiben
D18A
D18B
D18E        de=de+iy
D191        Filetyp in Akku
D192        Bit 0 gesetzt?
D193        => nicht gesetzt, nicht 'Protected'
D195
D196
D199        de=de+iy
D19C        de = Recordbuffer
D19D        Recordlänge
D1A0        KL LDIR, Header-Block in Record-Buffer
D1A3
D1A4
D1A5        Record 'schützen'
D1A8        Record in Sector Buffer, evtl auf Disc
D1AB        hl => Record Buffer
D1AC        Recordlänge
D1AF
D1B0        Anzahl der Records erniedrigen
D1B1        alle Records geschrieben?
D1B2
D1B3        => noch Records zu schreiben
D1B5

****** DISC IN CLOSE
D1B6        Stack sichern, OPENIN-Flag prüfen
D1B9        Motor aus, Event ausklinken

****** DISC IN ABANDON
D1BC        FOPENIN-aktiv Flag auf inaktiv setzen
D1C0        Abschluß

****** DISC OUT ABANDON
D1C2        Stack sichern, OPENOUT-Flag prüfen
D1C5
D1C8        de=de+iy
D1CB
D1CC        Blocks in Alloc.-Tab. wieder freigeben
D1CF
D1D2
D1D3        Track 0 suchen
D1D6        Abschluß
```

```
****** DISC OUT CLOSE
D1D8        File Character Count
D1DB        hl=hl+iy
D1DE        prüft ob überhaupt Zeichen übertragen
D1DF        wurden,
D1E0
D1E2
D1E3        sonst Disc Out Abandon, kein Dir-Eintrag
D1E5        Stackpointer sichern, Openout-Flag testen
D1E8        letzten Record in Buffer übertragen
D1EB
D1EE        de=de+iy, Filename im OPENOUT-FCB
D1F1
D1F2        Filename und Blockbeleg. im Dir eintragen
D1F5
D1F8        bc=bc+iy, bc:= OPENOUT-Header-Block
D1FB
D1FE        (hl):= Filetyp
D1FF
D203
D204        erstes Zeichen Extension in Akku
D205        prüfen auf ffh
D206        => Extension angegeben
D208        Filetyp in Akku
D209
D20B
D20D        Extension 'BAS' eintragen
D210

******
D212        Filetyp 2?
D214
D216        Extension 'BIN' eintragen
D219

******
D21B        Extension 3 Blanks eintragen
D21E        Adresse OPENOUT-Header-Block nach hl
D21F
D220        Filetyp in Akku
D221        Hi-Nibble ist unwichtig
D223        'Unprotected ASCII'?
D225        => ist 'protected'
D228        OPENOUT-FCB nach bc
D229
D22B        Kennzeichen, OPENOUT nicht aktiv
D22C
D22D        überschreibt '$$$' mit Original-Extension
D230        Kennzeichen DISC OUT CLOSE OK
D231        Meldung an Betriebssystem CPC alles OK
D232
```

```
******
D233
D236        H kein OPEN auf angem. Drive aktiv
D23A
D240
D243        de=de+iy
D246
D24F

******
D252
D25A

****** 'Protected File', Schutz durch XOR
D25C        OPENIN-Buffer-Pointer nach hl
D25D        RAM LAM, Zeichen aus Buffer in Akku
D25E
D263
D264        Byte in OPENIN-Buffer zurück
D265        Buffer-Pointer erhöhen
D266        und wieder auf Stack
D267        nächstes XOR-Byte
D269        Nächstes XOR-Byte
D26A        Zähler Bytes für ix-Tabelle
D26B        => Tabelle noch nicht zu Ende
D26D        11 Byte-Tabelle für ix
D26F        Tabellenanfang nach ix
D273        Zähler Bytes für hl-Tabelle
D274        => Tabelle noch nicht zu Ende
D276        13 Byte-Tabelle für hl
D278        Tabellenanfang
D27B        b:= Anzahl der zu verschlüsselnden Bytes
D27D
D280

****** Tabelle für ix
D281
D287

****** Tabelle für hl
D28C
D298

****** Extensions
D299
D2A5



******
D2A8        drei Blanks
D2A9
```

```
******
D2AB        '$$$' für temporären Filenamen
D2AD

******
D2AF        'BAK' für Backup-File
D2B1

******
D2B3        'BAS' für Basic-Files
D2B5

******
D2B7        'BIN' für Binär-Files
D2B9
D2BA        Lo-Byte des Tabellenanfangs addieren
D2BC        Ergebnis nach e
D2BD        mit Carry Hi-Byte addieren
D2BF        Lo-Byte subtrahieren
D2C0        Ergebnis Hi-Byte nach d
D2C1

****** Extension in Filenamen eintragen
D2C3
D2C4
D2C7        de=de+iy
D2CA
D2CB
D2CC        Länge Filename mit User, ohne Extension
D2CF
D2D0        Länge Extension
D2D3
D2D4        gewünschte Extension in Filename eintragen
D2D6
D2D9



******
D2DA        Länge Filename + Extension
D2DD
D2E2
D2E4        Anz. Dir-Einträge & Alloc.-Tab. auf 0
D2E7
D2EB
D2EC        nächsten DIR-Eintrag nach (de)
D2EF
D2F0        => kein weiterer Eintrag im DIR
D2F2        Extension 'BAK' anhängen
D2F5        sucht angegebenen Filenamen auf Disc
D2F8        => BAK nicht vorhanden
D2FA
D2FC        prüft, ob File READ ONLY
```

```
D2FF        => ist READ ONLY
D301
D302        Extension in (de) eintragen
D305        sucht angegebenen Namen auf Disc
D308        => File nicht vorhanden
D30A
D30C        prüft, ob File READ ONLY
D30F        => File ist READ ONLY
D311
D327
D329        Anz. Dir-Einträge & Alloc.-Tab. auf 0
D32C
D333

******
D335        Extension 'BAK' eintragen
D338        sucht angegebenen Filenamen auf Disc
D33B        => gefunden
D33E        '$$$'-File in Dir suchen
D341        => gefunden
D342        Original-Extension anhängen
D345        sucht angegebenen Filenamen auf Disc
D348        => nicht gefunden
D349
D34B
D34C        Extension 'BAK' eintragen
D34F
****** prüft, ob Filename mit '$$$' bereits auf Diskette ist
D351        Extension '$$$' eintragen
D354        sucht angegebenen Filenamen im Dir
D357        => nicht gefunden
D358        (bc) => Filename
D359        Adresse des identischen Dir-Eintrags
D35A        nach bc
D35B        Original-Extension in Dir-Record schreiben
D35E
D35F

******
D362        Anz. Dir-Einträge & Alloc.-Tab. auf 0
D365        nächsten Dir-Eintrag ermitteln, de= Anzahl
D368        => kein weiterer Dir-Eintrag
D369        sucht temp.($$$) und Orig. Filenamen
D36C        weiter suchen

******
D36E        Anz. Dir-Einträge & Alloc.-Tab auf 0
D371        nächsten Dir-Eintrag ermitteln, de= Anzahl
D374        => kein weiterer Dir-Eintrag
D375        wenn temp.File gefunden, Orig.Ext. in Rec.
D378        sonst weiter suchen
```

```
******
D37A
D37D
D37E        Extension in (de) eintragen
D381        sucht angegebenen Filenamen auf Disc
D384
D387

****** READ ONLY-File, Kommando abbrechen
D388        Extension in (de) eintragen
D38B
D38C
D38D        Systemmeldung 10, file is read only
D38F        ausgeben, Kommando beenden


******
D392
D396
D399        de=de+iy, Adresse OPENIN-FCB
D39C        prüfen, ob letzten Record gelesen
D39F        => kein weiterer Record im File
D3A1        de:= Recordnr., hl:= Filename im FCB
D3A2        (hl):= Record-Buffer
D3A3        Record in Record Buffer holen
D3A6
D3A7

******
D3A9
D3AE

****** Records in Sector Buffer, ggf. auf Disc
D3AF        Record Buffer
D3B0
D3B1        Anzahl Records
D3B2        und noch man Record Buffer
D3B3
D3B6        de=de+iy, OPENOUT-FCB
D3B9        prüft auf DISK FULL
D3BC        => noch Platz
D3BE        Systemmeldung 8, disc full
D3C0        ausgeben, Kommando abbrechen
D3C3        sucht freien Dir-Eintrag
D3C6
D3D1
D3D2        freien Block suchen und belegen
D3D5
D3D6        Systemmeldung 8, disc full ausgeben und
D3D8        Kommando abbrechen, wenn kein Block frei
D3DB        belegten Block merken
D3DC
```

```
D3E9
D3EA      Record in Sector-Buffer, ggf. auf Disc
D3ED
D3EE
D3F1      Anzahl Records +1
D3F4
D3F6
D3F7      Kennzeichen alles OK
D3F8

******
D3F9
D408
D40A      Record in Sector Buffer, ggf. auf Disc
D40D      Bad command ausgeben, Kommando abbrechen

****** prüft, ob letzter Record gelesen wurde
D410
D422
D423      32 Bytes von (hl) nach (de)
D426
D42D

****** !DIR
D42E      Stack sichern
D431
D433      folgen Parameter?
D434      => keine weiteren Parameter
D436      ein Parameter ist OK, sonst 'Bad command'
D439      b:= Länge String, hl:= Stringadresse
D43C      in korrekten Filenamen wandeln
D43F      Disk Parameter Header nach hl, ggf. Login
D442      'Drive #: user #' ausgeben
D445      Länge eines Filenamens incl '.'
D447      Anzahl Einträge/Bildschirmzeile bestimmen
D44A
D44B
D44C      Anz. Dir-Einträge & Alloc.-Tab. auf 0
D44F      Filename suchen und Belegung ermitteln
D452      => kein weiterer Eintrag
D454      prüft, ob Eintrag SYS-Attribut trägt
D457      => zweites Zeichen Ext. >7f, SYS-Attribut
D459      gezählte Einträge auf Stack
D45A
D45B      h:= Einträge/Zeile
D45C      l:= noch auszugebende Einträge in Zeile
D45D      => drei Blanks ausgeben
D460      => 'CR/LF' ausgeben
D463      Filenamen ausgeben, de zeigt auf Filenamen
D466      Angezeigte Einträge in aktueller Zeile
D467      => Zeile noch nicht voll
D469      Zeile voll, Anzahl auf Startwert
D46A
```

```
D46B        gezählte Einträge nach hl
D46C        nächsten Eintrag

****** Abschluß !DIR
D46E
D46F        Anzahl freie Blocks bestimmen, ausgeben

****** Anzahl Einträge/Zeile für DIR und CAT bestimmen
D472        Länge Eintrag plus drei Blanks
D474
D476
D477        TXT GET WINDOW
D47A        rechte Spalte des aktuellen Window
D47B
D47D
D47F        Einträge/Zeile-Zähler initialisieren
D481        Anzahl Einträge/Zeile
D482        Länge eines DIR-Eintrages
D483
D485        Anzahl DIR-Einträge/Zeile, Korrektur
D486        wenn Anzahl=0, dann keine
D487        Formatierung, Anzahl auf 1 setzen
D489

****** ERA
D48A        Stack sichern
D48D        ein Folgeparameter, sonst 'Bad command'
D490        Adresse Name zu löschendes File nach hl
D493        Filename für DOS anlegen
D496        Disk Parameter Header nach hl, ggf. Login
D499        Anz. Dir-Einträge & Alloc.-Tab. auf 0
D49C        Filename suchen und Belegung ermitteln
D49F        => File nicht gefunden melden, Abbruch
D4A1
D4A4        Filename suchen und Belegung ermitteln
D4A7
D4A9
******
D4AA
D4B0

******
D4B1        prüft, ob File READ ONLY
D4B4
D4B5        Systemmeldung '"Filename" is read only'
D4B7        => Meldung ausgeben, Abbruch
D4BA
D4BB        Blöcke in Alloc.-Tab. freigeben
D4BE        Kennzeichen File gelöscht
D4C0        in Filenamen eintragen
D4C1
```

```
****** REN
D4C4    Stack sichern
D4C7    zwei Folge-Parameter, sonst Abbruch
D4CA    erster Parameter, neuer Name, nach hl
D4CD    legt Filenamen für DOS an
D4D0
D4D1    zweiter Parameter, alter Name, nach hl
D4D4    Filenamen für DOS anlegen
D4D7
D4D9
D4DA    Bad command ausgeben, Kommando abbrechen
D4DD    Disk Parameter Header nach hl, ggf Login
D4E0
D4E1
D4E2    prüfen, ob neuer Filename schon existiert
D4E5
D4E8
D4E9    Anz. Dir-Einträge & Alloc.-Tab auf 0
D4EC    alten Filenamen suchen, Belegung ermitteln
D4EF    => File nicht gefunden
D4F1    prüft, ob File READ ONLY
D4F4    => File ist READ ONLY, keine Umbenennung
D4F7
D4F9
D4FA    Länge neuer Filename
D4FD    alten Namen überschreiben
D4FF
D502
D505    Filename suchen und Belegung ermitteln
D508
D50B

****** File nicht gefunden, Abbruch
D50C    Adresse des Filenamens für Ausgabe nach de
D50D
D50E    Systemmeldung 6, file not found
D510    ausgeben, Kommando abbrechen

****** CATALOG
D513    Stackpointer merken
D516    User Buffer-Adresse
D517    nach ix übertragen
D519    Länge des User-Buffers
D51C    löscht User-Buffer (de) bis (de+bc)
D51F
D522    Disk Parameter Header nach hl, ggf Login
D525    Ausgabe 'Drive #: user #'
D528
D529
D52A    Anz. Dir-Einträge & Alloc.-Tab auf 0
D52D    Filename suchen und Belegung ermitteln
D530
D532    prüft, ob File SYS-Attribut trägt
```

```
D535        => ist SYS-File, nicht ausgeben
D537
D538        einen Eintrag in User-Buffer
D53B
D53C        nächsten Eintrag, so noch einer vorhanden
D53E        Länge eines DIR-Eintrags auf Monitor
D540        bestimmt Anzahl der Einträge/Zeile
D543        Anzahl nach d
D544
D54C
D54E        Anzahl der Zeilen
D54F        User-Buffer
D551        nach hl
D552        Zeilen nach c
D553        Einträge/Zeile nach b
D554
D555        Ausgabe eines Eintrags
D558
D55A
D55C        hl:= hl * 14 für alphabetische Ausgabe
D55F        in mehreren Spalten
D560
D561        und nächsten Eintrag ausgeben
D563
D56F
D571        ermittelt Anzahl belegte Blocks
D574
D576        Systemmeldung 3, xxxK free
D577        ausgeben

****** Ausgabe eines Eintrages aus User-Buffer
D57A        RAM LAM, LD A,(HL) aus User-Buffer
D57B        Zeichen eine 0?
D57C        dann kein Eintrag und =>
D57D        Register retten
D57E
D57F
D580        Einträge/Zeile
D581        gleich ausgegebener Einträge?
D582        wenn nicht gleich, dann 3 Blanks ausgeben
D585        sonst 'CR/LF' ausgeben
D588
D589        Filenamen ausgeben
D58C        prüft, ob File READ ONLY
D58F        '*'
D591        wenn R/O, '*' ausgeben
D593        'Blank'
D595        TXT OUTPUT
D598
D59B
D59C        RAM LAM, LD A,(HL) aus Ram
D59D
D59E
```

```
D59F    RAM LAM, LD A,(HL) aus Ram
D5A0
D5A1    Systemmeldung 2, drei Blanks
D5A3    ausgeben
D5A6
D5A9

******
D5AA
D5AC
D5AE    User-Buffer
D5B0    nach hl
D5B1    RAM LAM, LD A,(HL) aus Ram
D5B2    Prüft ob Zeichen 0
D5B3
D5C8

******
D5CA
D5CB    bestimmt Anzahl bel. Blocks des Files
D5CE
D5D2
D5D3    RAM LAM, LD A,(HL) aus Ram
D5D4
D5D5
D5D6    RAM LAM, LD A,(HL) aus Ram
D5D7
D5E1

******
D5E3
D5E4
D5E6
D604

******
D605
D607
D608    Ende-Kennzeichen im User-Buffer
D60A
D60B    de := Record-Pointer
D60C
D60D    Länge Filename+Extension
D610    KL LDIR, aus Record in User-Buffer
D613    Record-Pointer nach hl
D614    Record-Anfang nach hl
D615    Record-Anfang nach de
D616    Bestimmt Anzahl belegte Blocks des File
D619    Anzahl Blocks nach de
```

```
D61A
D61B        in User-Buffer eintragen
D61C
D622

******
D623
D62D
D62E        RAM LAM, LD A,(HL) aus Ram
D62F
D639

****** hl:=hl*14
D63A        merken
D63B
D63C
D63D        hl=alter Wert in hl*2
D63E        hl=alter Wert in hl*3
D63F        hl=alter Wert in hl*6
D640        hl=alter Wert in hl*7
D641        hl=alter Wert in hl*14
D642
D643

******
D644        Anz. Dir-Einträge & Alloc.-Tab. auf 0
D647        Filename suchen und Belegung ermitteln
D64A
D64C        Systemmeldung 5, file already exists
D64E        ausgeben, Kommando abbrechen

****** Filename im Directory suchen
D651        Anz. Dir-Einträge & Alloc.-Tab. auf 0
D654        Filename suchen und Belegung ermitteln
D657        => nicht gefunden
D659        Nummer des gef. Direintrags
D65A
D65D        hl=hl+iy, Adresse OPENIN-FCB
D660        nach de
D661        32 Bytes, DIR-Eintrag in OPENIN-FCB
D664        Nummer des gef. DIR-Eintrags
D665        File auf diesem Drive aktiv?
D668
D669
D66A        => File ist aktiv
D66B        Directory zu Ende lesen, Anz. Files und
D66E        Anz. belegte Blocks ermitteln
D670        Directory gelesen
D671        FFile auf diesem Drive als aktiv markieren
D675
```

```
****** sucht Filenamen im Directory, ermittelt Block-Belegung
D676        Anz. Dir-Einträge & Alloc.-Tab. auf 0
D679        Filenamen suchen und Belegung ermitteln
D67C        => nicht gefunden
D67E
D681

****** bestimmt Block-Belegung Disc
D683        => Filename
D684        Track 0 suchen
D687
D688
D68B        File auf diesem Drive aktiv?
D68E
D68F        => File ist eröffnet
D690
D691        Allocation-Tab. auf 0, Dir-Blocks belegen
D694
D695

****** Filenamen suchen, Block-Belegung + Anz. Files
D698        Anz.Blocks in Alloc.-Tab, Files zählen
D69B        => kein weiterer Eintrag
D69C        angegebebe Filename= Direintrag?
D69F        => ungleich, suchen und Belegung ermitteln
D6A1        gleichen Filenamen gefunden

****** Anzahl Files auf Disc ermitteln
D6A2        Zähler Einträge
D6A3        OPEN auf diesem Drive aktiv?
D6A6
D6A7        => Open ist aktiv
D6A9        Rec. in Buffer, Rec.-Pointer=> Dir-Eintrag
D6AC        => kein weiterer Eintrag
D6AD        de Zeiger auf Dir-Eintrag
D6AE        Kennzeichen gelöschtes File
D6B0
D6B1        => wenn Eintrag gelöscht
D6B2        sonst Anzahl Einträge erhöhen
D6B5
D6B7        bestimmt Anzahl der belegten Blocks

****** Einsprung D6A2, wenn Open aktiv
D6BA        prüft, ob Pos. Dir-Eintrag ok
D6BD
D6BE

****** Anzahl Character im File nach hl
D6C1
D6C4
D6C5        ld hl,(hl)
```

```
******
D6C8
D6E8
D6EA        Extent Maske in Akku
D6ED
D6F9

******
D6FA
D704
D707        löscht (de) bis (de+bc)
D70A
D70B

******
D70C
D720
D721
D728
D729        hl = de? Carry wenn gleich
D72C
D72E

******
D72F
D732
D734        Blockmaske in Akku
D737
D739
D73B        Block-Shift in Akku
D73E
D741
D743        max. Blocknummer, Hi-Byte in Akku
D746
D757
D758        ld hl,(hl)
D75B

******
D75D
D76D
D76F        Block-Shift in Akku
D772
D77A

******
D77B
D77C

****** Anzahl Records im FCB bestimmen
D77D
D78B
```

```
****** belegt freien Directory-Eintrag
D78C
D78D
D78E        freien Eintrag suchen
D791        Adr. Eintrag auf Stack, hl= Adr. Filename
D792        Drive-Nr wird nicht benötigt
D793        Filename und Blockbel. in Record schreiben
D796
D797        Filename und Blockbel. in Dir-Record
D79A
D79B

****** Anzahl Character im File auf 0 setzen
D79C
D7A6

****** Anzahl Character im File um 1 erhöhen
D7A7
D7B2

******
D7B3        Anz. Dir-Einträge & Alloc.-Tab auf 0
D7B6        Filename suchen und Belegung ermitteln
D7B9

****** freien Directory-Eintrag suchen
D7BB
D7BE
D7BF        prüft auf Platz im Directory
D7C2        Systemmeldung 7, directory full ausgeben
D7C4        Kommando abbrechen, wenn kein Platz
D7C7        (de)= Record-Pointer in Dir-Record
D7C8        Eintrag gelöscht?
D7CA        => nicht frei, nächsten Eintrag
D7CC
D7CD        OPEN auf diesem Drive aktiv?
D7D0
D7D1
D7D3        Fehler, Kommando abbrechen
D7D6
D7D7

****** sucht den in bc angegeben Filenamen im Directory
D7D8        (bc) => Filename des gew. Files
D7D9        (de) => Recordpointer
D7DA
D7DB        Adresse angegebener Filename nach hl
D7DC
D7DD        erstes Zeichen DIR-Eintrag, User-Nummer
D7DE        mit angegebener User-Nummer gleich?
D7DF        wenn nicht, dann =>
D7E1
D7E2
```

```
D7E3        Länge Filename + Extension
D7E5
D7E6        '?' Joker in angegebenem Filenamen?
D7E8        => Zeichen im Dir-Eintrag überlesen
D7EA        Zeichen aus Dir-Eintrag
D7EB        mit Zeichen im angegeb. Namen vergleichen
D7EC
D7EE        => Namen sind verschieden
D7F0        sonst nächstes Zeichen angegeb. Name
D7F1        nächstes Zeichen Dir-Eintrag
D7F2        prüfen
D7F4
D7F8
D7FA        Extent-Maske in Akku
D7FD
D810
D811        File nicht gefunden
D812        Kennzeichen OK
D813

****** löscht Allocation-Tabelle, trägt Dir-Blocks ein
D814
D816        max. Blocknummer nach hl
D819
D81B        teilt Blocknummer durch 8
D81E        Korrektur, 22 Bytes für Allocation-Tabelle
D81F
D820
D822        (hl) => Anfang aktuelle Allocation-Tabelle
D825        8 Blöcke = 1 Byte als frei kennzeichnen
D827        nächstes Byte
D828        Anzahl verringern
D829        alle 22 Bytes gelöscht?
D82A
D82B        => noch nicht alle gelöscht
D82D
D82F        Größe des Directory in Blocks nach hl
D832
D833
D835        (hl) => Anfang aktuelle Allocation-Tabelle
D838        Dir-Blocks als belegt eintragen
D839
D83B

****** belegt Blocks dieses Eintrags in Allocation Tab.
D83C        Anzahl geprüfte Dir-Einträge
D83D        Zeiger auf Anfang Eintrag in Record
D83E
D83F
D840        Offset zur Blockbelegung
D843
D844        Anzahl der Belegungseinträge/Dir-Eintrag
D846
```

```
D848
D84A        max. Blocknummer, Hi-Byte in Akku
D84D        wenn 0 dann 1-Byte-Einträge
D84E        => 1-Byte-Einträge
D850
D854
D855        keine weiteren Blocks belegt
D857
D858
D85A        max. Blocknummer nach hl
D85D
D85E        Blocknummer gültig?
D85F
D860
D861        gültig => in Alloc.-Tab belegen/freigeben
D864
D86B

****** Blocknr. in Bitposition Alloc.-Tab belegen/freigeben
D86C
D86D        belegte Blocknummer
D86E
D870
D872        Blocknummer durch 8 teilen
D875        Ergebnis nach de, Byte in Alloc.-Tab.
D876
D878        (hl) => Anfang aktuelle Allocation-Tabelle
D87B        erforderliche Anzahl Bytes überspringen
D87C        belegte Blocknummer nach de
D87D
D883
D884        Bit-Position in Alloc.-Tab. bestimmen
D885
D88D
D88E        bestehendes Bitmuster auf Blockbit aufORen
D88F        und in Allocation Tabelle abspeichern
D890

****** sucht freien Block in Allocation-Tabelle
D893
D895
D897        max. Blocknummer nach hl
D89A
D89B
D89D        (hl) => Anfang aktuelle Allocation-Tabelle
D8A0        b:= Bitzähler, c:= Bitmuster
D8A3        Eintrag aus Alloc.-Tab. in Akku
D8A4        Bitmuster für Block
D8A5        => freien Block gefunden
D8A7        nächstes Bit prüfen
D8A8
D8A9        de:= noch zu prüfende Blockzahl
D8AA        keine weiteren Blocks mehr?
```

```
D8AB    => kein freier Block gefunden, Disc Full
D8AD    zu prüfende Blockzahl verringern
D8AE    Bitschleife über 8 Bits
D8B0    nächstes Byte in Allocation-Tabelle
D8B1    weiter prüfen

****** ermittelt Blocknummer aus Bit-Position, belegt in Alloc.-Tab.
D8B3    Bitmuster aus Alloc.-Tab. in Akku
D8B4    gefundenen freien Block belegen
D8B5    und in Alloc.-Tab. ablegen
D8B6
D8B8    max. Blocknummer nach hl
D8BB
D8BC    Blocknummer errechnen, nach hl
D8BE    Kennzeichen freien Block gefunden
D8BF
D8C1


****** Anzahl belegte Blocks aus Alloc.-Tab. für Directory-Anzeige
D8C2
D8C3
D8C4    hl ist Blockzähler
D8C7    auf Stack
D8C8
D8CA    max. Blocknummer nach hl
D8CD    und nach de
D8CE
D8D0    (hl) => Anfang aktuelle ALV
D8D3    b:= Bitzähler, c:= Bitmuster
D8D6    Byte aus Alloc.-Tab. in Akku
D8D7    Bitposition als belegt gekennzeichnet?
D8D8    => nicht belegt
D8DA    Zähler nach hl
D8DB    Zähler erhöhen und
D8DC    wieder auf Stack
D8DD    Bitmuster in Akku
D8DE    nächstes Bitmuster durch rotieren holen
D8DF    und wieder in c merken
D8E0    de:= Anzahl noch zu prüfende Blocks
D8E1    Anzahl schon null?
D8E2    => kein weiterer Block zu prüfen
D8E4    Anzahl zu prüfender Blocks verringern
D8E5    Bitschleife
D8E7    nächstes Byte aus Allocation-Tabelle
D8E8    und weiter prüfen
```

```
****** ermittelt tatsächliche Anzahl belegter Blocks
D8EA    Zähler belegte Blocks von Stack holen
D8EB    ermittelt aus BSH und hl tats. Belegung
D8EE    Ergebnis nach de
D8EF
D8F1

****** bestimmt Anzahl der belegten Blocks des Files
D8F2    Anfang eines Dir-Eintrages im Dir-Record
D8F3    Offset zu Blockbelegungseinträgen
D8F6
D8F7    d:=Bytes Alloc.-Tab., e:= Zähler bel.Blks
D8FA
D8FC    max. Block-Nummer, Hi-Byte in Akku
D8FF    wenn 0, dann nur 1-Byt-Werte in Alloc.-Tab
D900    ein Byte aus Allocation-Tabelle
D901    Pointer in Alloc.-Tab. erhöhen
D902    max. Blocknr Hi-Byt =0?, dann nur ein Byte
D904    sonst Lo-Byte prüfen
D905
D906
D907    Byte Alloc.-Tab. prüfen, wenn 0
D908    kein Block belegt
D90A    Zähler belegte Blocks erhöhen
D90B    Anzahl verringern
D90C    noch Bytes in der Tabelle?
D90E    Anzahl nach hl
D90F
D910
D912    Block-Shift in Akku
D915
D91A

******
D91C
D91D    bc => temporärer Filename
D91E    Anzahl geprüfter Eintr. im Record
D91F    4 sind möglich, 0-3
D921
D923    Anzahl Einträge nach de
D924
D926    max DIR-Einträge nach hl
D929    hl = de? alle möglichen Einträge geprüft?
D92C
D92D
D92E    Alle gelesen =>
D930    neuen Record in Record-Buffer
D933
```

```
D935
D937    (hl) => Recordbuffer, wieder auf Anfang
D93A    Länge eines Eintrags im Buffer (32 Byt)
D93D
D93E



******
D940    Recordpointer auf nächsten Dir-Anfang
D941
D943    Recordpointer nach de
D944
D947

******
D948
D94A    Anzahl geprüfte Einträge/4
D94D    Ergebnis Anz. gepr. Records nach de
D94E
D950    (hl) => Recordnummer
D953    holt Record, Nr in hl, in Record Buffer
D956
D958    Anz. zu prüfende DIR-Einträge nach hl
D95B
D95C    hl = de, alle möglichen Einträge geprüft?
D95F
D960    => alle Einträge gelesen
D961
D963    (hl) => Prüfsummenblock
D966
D967    Prüfsumme über Record errechnen
D96A    gleich mit gespeicherter Prüfsumme?
D96B    => sind gleich
D96C
D974
D975    Fehler
D978
D979

******
D97A    Nummer des belegten Dir-Eintrages
D97B    Buffer-Adresse
D97C
D97E    teilt hl/4
D981    Ergebnis Recordnummer nach de
D982
D984    (hl) => Recordbuffer
D987
D989    Dir-Record in Sector Buffer, ggf auf Disc
D98C
D98E    Anz. zu prüfende DIR-Records nach hl
D991
```

```
D992        hl = de? alle möglichen Records geprüft?
D995
D998
D99A        (hl) => Prüfsummenbuffer
D99D
D99E        Prüfsumme über Record
D9A1        und in Prüfsummenbuffer eintragen
D9A2
D9A3
D9A4        prüft, ob Filename in richtiger Position
D9A7        Fehler
D9A8
D9AA
D9AB        Anzahl belegter Einträge erhöhen
D9AC
D9AE        (hl) => Anzahl Directoryeinträge
D9B1        neue Anzahl merken
D9B2
D9B7

****** prüft, ob Position des Dir-Eintrags ok
D9B8
D9B9        Anzahl Dir-Einträge
D9BA
D9BC        (hl) => Anzahl belegte Dir-Einträge
D9BF        Anzahl nach de
D9C0
D9C2
D9C3        hl = de, errechneter Eintrag ok?
D9C6
D9C7

****** Prüfsumme über Record, Ergebnis in Akku
D9C8
D9C9
D9CA        Record-Länge
D9CC
D9CE        (hl) => Recordbuffer
D9D1        Akku löschen
D9D2        Prüfsumme über Record
D9D3
D9D8

****** prüfen, ob READ ONLY-Attribut, Bit 7 erstes Zeichen Extension
D9D9
D9DA        erstes Zeichen Extension
D9DD

****** prüfen, ob SYS-Attribut, Bit 7 zweites Zeichen Extension
D9DF
D9E0        zweites Zeichen Extension
D9E3
D9E4        Zeichen in Akku
```

```
D9E5        Carry gesetzt, wenn Attribut gesetzt
D9E6
D9E7

****** Record, Nr in de, in Record Buffer holen
D9E8
D9EA
D9EB        Track und Sector aus Recordnr. bestimmen
D9EE        Record in Record Buffer lesen
D9F1        Fehlermeldung im Akku

****** Record, Nr in de, in Sector Buffer schreiben
D9F3
D9F6
D9F7        Track und Sector aus Recordnr. bestimmen
D9FA
D9FB        Record in Sector Buffer schreiben
D9FE        Fehlermeldung im Akku
D9FF        => Fehler, Kommando abbrechen
DA02
DA05

****** errechnet aus Recordnr. Track und Sector
DA06        Recordnummer
DA07        Buffer-Adresse für einen Record
DA08        nach bc und
DA09        an Controller-Routinen übergeben
DA0C
DA0D
DA0F        Spur-Offset nach hl
DA12        und bc
DA13
DA14
DA15        Anz. Records/Track nach hl
DA18        Korrektur
DA19
DA23
DA24        Track in c nach be54 für Controller
DA27
DA28
DA29        (hl) => (a910/a920)
DA2C
DA2D        Recordnummer übersetzen
DA30
DA31
DA32        Sector nach be55 für Controller
```

```
****** lade hl mit Disk Parameter Header + Akku
DA35        addiert Lo-Byte aktuelle DPH
DA38        Ergebnis nach l
DA39        addiert Hi-Byte, evtl mit Carry
DA3C
DA3D        Hi-Byte nach h
DA3E

****** lade hl mit Inhalt (Disk Parameter Header + Akku)
DA3F        (hl) => DPH + Akku
DA42        ld hl,(hl)

****** lade hl mit Inhalt (DPB+Akku)
DA45
DA46
DA48        (hl) => Anfang Disk Parameter Block
DA4B        gewünschter Offset
DA4C
DA4D
DA4E        errechnet benötigte Adresse
DA4F
DA50
DA51        LD HL,(HL)
****** lade Akku mit Inhalt (Disk Parameter Block + Akku)
DA54
DA55        lade hl mit Inhalt (DPB+Akku)
DA58        gewünschtes Byte in Akku
DA59
DA5A

******
DA5B
DA5E

******
DA60
DA68

******
DA6A        legt EFN an, prüft Filenamen
DA6D

****** legt expanded Filename an, prüft auf Blank und '?'
DA6F
DA71
DA74        Filename im Buffer anlegen
DA77
DA78        Länge des Filenamens incl. Extension
DA7A
DA7B        bc => Filename
DA7C        erstes Zeichen des Namens in Akku
DA7D        '?' Joker
4A7F        wenn ja => Bad command ausgeben
```

```
DA81
DA82        nächstes Zeichen auf '?' prüfen
DA84        kein Fragezeichen gefunden
DA85        scheint ok

******
DA86
DA8B

******
DA8D        Blank
DA8F        Adresse Record-Buffer
DA92        Filenamen anlegen
DA95
DA9F

****** legt Expand. Filename an, prüft, ob Filename ok
DAA0        legt EFN an
DAA3        => Leerzeichen im Filename, Bad Command
DAA5

******
DAA6        Blank
DAA8        Offset zu Record-Buffer
DAAB
DAB5

******
DAB6
DAB7        de=de+iy, Buffer für Expand. Filename
DABA        merken
DABB        Nummer des aktiven Drive
DABE        in Filenamen eintragen
DABF
DAC0        aktive User-Nummer
DAC3        in Filenamen eintragen
DAC4
DAC6
DAC7        Filename ist 8 Zeichen lang
DAC9        8 Blanks nach (de) bis (de+7)
DACC
DACD        Extension ist 3 Zeichen lang
DACF        3 mal ffh nach (de+8h) bis (de+0ah)
DAD2
DAD5        löscht (de+0bh) bis (de+0dh)
DAD8        b => Länge Inp. Filename, c => 0ffh
DAD9        de => Anfang des Expanded Filename (EFN)
DADA        hl => Adresse Inp. Filename (IFN)
DADB
DADC        legt EFN an, wenn IFN korrekt
DADF
DAE0        bad command, Input Filename nicht ok
DAE2        Adresse EFN nach bc
```

```
DAE3
DAE4        de => Filename
DAE5
DAE6        erstes Zeichen des EFN
DAE7        Blank?
DAE9

******
DAEA        Bad command ausgeben, Kommando abbrechen

****** Prüft IFN, legt EFN mit Drive und USER an
DAED        Dummy
DAEE        Länge IFN testen
DAF1
DAF2        Fehler aufgetreten! Kein Filename?
DAF3
DAF5
DAF6        ist Zeichen ein ':'?
DAF8        ja, dann =>
DAFA        ein Zeichen aus IFN, convert. Uppercase
DAFD        prüfen ob ':'
DAFF        kein ':' im Namen gefunden

****** wenn ':' prüfen auf gültige USER-Nummer
DB00
DB02
DB03        Sprung, wenn kein ':' gefunden
DB05
DB06        '0'
DB08        Zeichen kleiner '0', dann keine Ziffer
DB0A        ':'
DB0C        Zeichen größer/gleich ':', keine Ziffer
DB0E        ist Ziffer, minus 30h ergibt Binärzahl
DB10
DB11        Ziffer in EFN eintragen
DB12        ein Zeichen aus IFN, convert. Uppercase
DB15        '0'
DB17        Zeichen kleiner '0', dann keine Ziffer
DB19        ':'
DB1B        Zeichen größer/gleich ':', keine Ziffer
DB1D        Zeichen ist eine Ziffer
DB1E        war erste Ziffer eine 1?
DB1F        nein, Fehler im IFN, falsche USER-Nummer
DB20        zweite Ziffer prüfen, ob 0 bis 6
DB22
DB24        zweite Ziffer >6, falsche USER-Nummer
DB25        Ziffer in EFN eintragen
DB26        ein Zeichen aus IFN, convert. Uppercase

****** wenn keine Ziffer, prüfen auf (logisch) gültige Drive-Nummer
DB29
DB2A        'Q'
DB2C        Zeichen ist 'Q','R'... Fehler im IFN
```

```
DB2E      'A'
DB30      Zeichen ist kleiner 'A', Fehler im IFN
DB32      'A' abziehen, ergibt 0 bis 15
DB34      das sind die (logisch) möglichen Drives
DB35      ein Zeichen aus IFN, convert. Uppercase

****** wenn USER und/oder Drive, dann muß ':' folgen, sonst Fehler
DB38      Zeichen holen, Blank überlesen
DB3B      jetzt muß ein ':' kommen
DB3D      wenn nicht, dann Fehler im IFN
DB3E      Zeichen holen, Blank überlesen
DB41
DB42      Fehler, kein Zeichen nach ':' vorhanden

****** bestimmt aus IFN den eigentlichen Filenamen und Extension
DB43
DB44
DB45      '.'
DB47      Filename fehlt, nur Extension angegeben
DB48      Länge des EFN
DB4A
DB4D
DB4E      jetzt muß ein '.' kommen
DB50      Fehler, IFN zu lang
DB51      Zeichen holen, Blank überlesen
DB54      Länge der Extension
DB56      keine Zeichen mehr, dann Ext. = 3 Blanks

****** prüft Zeichen für Filename und Ext., trägt in EFN ein
DB58      ' '
DB5A      wenn kleiner Blank, dann mit Blanks füllen
DB5C
DB5D
DB5E      zwischenspeichern
DB5F      Tabelle der 'verbotenen' Zeichen im Namen
DB62      Tabellenwert holen
DB63      Zeiger erhöhen
DB64
DB65      Tabelle zu Ende?
DB67      Tabellenwert mit Zeichen vergleichen
DB68      wenn ungleich dann nächsten Vergleich
DB6A      Kennzeichen 'verbotenes' Zeichen
DB6B      Zeichen in Akku
DB6C
DB6D
DB6E      'verbotenes' Zeichen, Rest mit Blanks
DB70      Zähler Länge Filename oder Extension
DB71      Alle Zeichen durch? dann =>
DB72      '*', Joker für den Rest
DB74      wenn Joker dann Rest mit '?' füllen
DB77      Zeichen ok. In expand. Filename eintragen
DB79      ein Zeichen aus filename, convert. Upper
DB7C      kein Zeichen mehr, Rest mit Blanks füllen
```

```
DB7E      Leerzeichen?
DB80      wenn nicht, dann prüfen und eintragen

****** Rest nach Blank im IFN wegwerfen
DB82      prüft auf Blank, wenn ja, nächstes Zeichen

****** füllt Speicher (de) bis (de+c) mit Leerzeichen
DB85
DB86      Wert für Leerzeichen
DB88
DB8D

****** füllt Speicher (de) bis (de+c) mit '?'
DB8E      '?', Joker für ein Zeichen

****** füllt Speicher (de) bis (de+c) mit Zeichen im Akku
DB90      nötige Korrektur
DB91      nächstes Speicherstelle
DB92      oder evtl c=0, dann Ende
DB93      Zeichen in adressierten Speicher
DB94      Zeiger erhöhen
DB95      und noch einmal

****** holt ein Zeichen, prüft, ob Länge Filename >0
DB97      ein Zeichen, prüft Länge Filename
DB9A      Länge ist 0, Fehler!

****** prüft Akku auf Blank, wenn Blank, dann nächstes Zeichen holen
DB9B      Blank?
DB9D
DB9E      war was anderes
DB9F      nächstes Zeichen holen
DBA2      prüft auf Blank, wenn ja, nächstes Zeichen
DBA4

****** holt ein Zeichen aus Filename, convertiert in Großbuchstaben
DBA5      (verbleibende) Länge Filename
DBA6      Länge <> 0?
DBA7      wenn 0, dann Return
DBA8      (hl) => Zeiger auf inp. filename
DBA9
DBAA      RAM LAM, LD A,(HL) aus Ram
DBAB      nur ASCII-Zeichen, bitte
DBAD      wandelt in Großbuchstaben
DBB0      Kennzeichen Byte geholt
DBB1
```

```
****** in Filenamen verbotene Zeichen
DBB2        '<'
DBB3        '>'
DBB4        '.'
DBB5        ','
DBB6        ''
DBB7        ':'
DBB8        '='
DBB8        eckige Klammer auf
DBB8        eckige Klammer zu
DBB8        Underline
DBB8        '%'
DBB8        Shift Klammeraffe
DBBE        '('
DBBE        ')'
DBBE        '/'
DBBE        Backslash
DBBE        Delete
DBBE        Kennzeichen Ende der Tabelle

****** gibt drei Blanks aus, aktuell. Drivenr. nach c
DBC4        Systemmeldung 1
DBC6

****** gibt 'Filename' aus, aktuell. Drivenr. nach c
DBC8        Systemmeldung 11
DBCA
DBCB        Drive-Nr nach c
DBCE

****** gibt 'Drive #: user #' aus, e=Drive, c=Usernr.
DBD0
DBD1        Drive-Nr
DBD2        nach e
DBD3
DBD5
DBD6        User-Nr
DBD7        nach c
DBD8        Systemmeldung 12
DBDA        Systemmeldung ausgeben
DBDD
DBDE

****** überträgt 32 Bytes mit LDIR von hl nach de
DBDF
DBEA

****** teilt hl durch (Akku^2)
DBEB
DBF2
```

****** vergleicht hl mit de
DBF3
DBF8

****** lade hl mit (hl)
DBF9
DBFF
****** dc00 bis dfff werden nicht genutzt
DC00

Kapitel 5: Programme und Tricks für die DDI-1

5.1 Fehler bei MERGE und CHAIN MERGE

Dieses Kapitel ist für all die Leute interessant, die ihre Floppy in der ersten Zeit gekauft haben. Leider hat sich im Betriebssystem des CPC 464 ein Fehler eingeschlichen, so daß beim Nachladen von Programmen durch die Befehle MERGE und CHAIN MERGE unberechtigterweise die Fehlermeldung EOF met ausgegeben wird.

Da der Firma Schneider der Fehler bekannt ist, wird er in späteren Versionen des Betriebssystemes wahrscheinlich nicht mehr enthalten sein, so beim CPC 664 und dem CPC 6128. Diejenigen unter Ihnen, die überhaupt nicht wissen, ob ihre Floppy den im folgenden beschriebenen Fehler nun auch hat (oder auch nicht), können sich anhand dieses Kapitels Klarheit verschaffen.

Wie bereits erwähnt, und wie Sie vielleicht auch schon am eigenem Leibe erfahren haben, hat AMSDOS einen kleinen Fehler im Betriebssystem. Seien Sie nicht allzu verärgert darüber, denn die teuersten und edelsten Anlagen haben noch unzählige Fehler im Betriebssystem.

Auch kein Grund zur Panik: Ihr Problem wird gelöst, denn Sie haben ja das Floppy-Buch in den Händen. Wir geben Ihnen nicht nur einen Lösungsvorschlag, sondern gleich zwei. Doch wann und warum entsteht dieser Fehler? Vielleicht hatten Sie bis jetzt bei der Benutzung der Befehle MERGE und CHAIN MERGE immer Erfolg. Das kann durchaus vorkommen. Besonders bei kleinen Routinen ist es sogar relativ unwahrscheinlich, daß ein Fehler auftritt.

Probieren Sie es anhand unseres Beispieles einmal aus:

```
NEW
```

```
10 PRINT x;"Es hat geklappt."
20 GOTO 20
```

```
SAVE "Zwei"
```

```
NEW
```

```
10 PRINT "Wir versuchen es ..."
20 x=1
30 CHAIN MERGE "Zwei"
```

```
RUN
```

Danach können Sie sich das Programm LISTen lassen. Alles wie gewollt und erwartet. (Siehe auch entsprechendes Kapitel im CPC-BASIC-Handbuch.) Geben Sie nun folgendes ein:

```
NEW
```

```
26 PRINT x;"Es hat geklappt."
27 END
```

```
SAVE "Zwei"
```

```
NEW
```

```
10 PRINT "Wir versuchen es ..."
20 x=1
25 CHAIN MERGE "Zwei"
```

```
RUN
```

Achten Sie aber genau auf die Zeilennummern, sie sind wichtig in unserem Beispiel.

Nach dem Starten des Programmes werden Sie ein

```
EOF met in 25
```

erhalten. Sie wissen, daß EOF End of File bedeutet, und Sie werden sich zu Recht fragen, wieso stößt der Rechner auf ein End of File? Das genau ist der Fehler im Betriebssystem. Wir haben in unserem Beispiel diesen Fehler heraufbeschworen, indem wir die Zeilennummer 26 verwendet haben. Sobald beim Nachladen ein &1A (hexadezimal) = 26 (dezimal) gefunden wird, wird Ihnen die Fehlermeldung EOF met ausgegeben. Als ASCII-Zeichen wird der Code 26 kaum in einem Programm vorkommen können, jedoch bei der Kodierung von Programmen.

Beim Abspeichern eines Programmes auf Diskette wird für jede Zeile des Programmes auch die Länge und Startadresse im Speicher dieser Zeile auf Diskette geschrieben. Dies sind drei Zeichen pro Zeile, wo die Gefahr sehr groß ist, daß eins dieser drei Bytes &1A wird. Weiterhin wird die Zeilennummer als 16-Bit-Integerzahl abgespeichert. Hier kann auch ein &1A vorkommen (wie in unserem Beispiel). Die Wahrscheinlichkeit ist somit 1:(256/5)=51.2 pro Zeile, das bedeutet, daß statistisch gesehen bei spätestens 52 Zeilen ein "EOF met" kommen muß. In der Praxis reichen aber meistens schon weniger Zeilen.

Ein Lösungsweg ist es nun, die Programme als sogenannte ASCII-Datei abzuspeichern. Die Programme werden dann nicht mehr kodiert, sondern wie auf dem Bildschirm Zeichen für Zeichen auf Diskette geschrieben. Die Programmfiles werden dabei unwesentlich länger, bei einem 30 KByte langen Programm wurden daraus 32 KByte. In unserem Beispiel hätten Sie dann eingeben müssen:

```
SAVE "Zwei",a
```

Das *,a* steht für ASCII-Datei. Diese erste Lösung ist auch die einfachere Lösung. Wenn Sie allerdings geschützte Programme haben, die Sie nachladen wollen (als Beispiel), so nützt Ihnen diese Lösung jedoch sehr wenig.

Als zweite Lösung haben wir dann ein kleines Programm anzubieten, das den CHAIN-MERGE-Befehl wie selbstverständlich, das bedeutet, wie im Handbuch beschrieben und vom Kassetten-BASIC her bekannt, ausführt.

```
1 '
2 ' Firmware-Patch für CHAIN-MERGE
3 ' Schneider CPC & DDI-1
4 '
5 '
6 '
10 MEMORY HIMEM-41
20 DEF FNmsb(a)=&FF AND INT(a/256)
30 DEF FNlsb(a)=&FF AND UNT(a)
40 FOR i=HIMEM+1 TO HIMEM+38
50 READ byte
60 POKE i,byte
70 NEXT i
80 POKE HIMEM+3, FNlsb(HIMEM+39)
90 POKE HIMEM+4, FNmsb(HIMEM+39)
100 POKE HIMEM+9, FNlsb(HIMEM+41)
110 POKE HIMEM+10,FNmsb(HIMEM+41)
120 POKE HIMEM+18,FNlsb(HIMEM+1)
130 POKE HIMEM+19,FNmsb(HIMEM+1)
140 'CAS IN CHAR
150 POKE HIMEM+39, PEEK(&BC80+0)
160 POKE HIMEM+40, PEEK(&BC80+1)
170 POKE HIMEM+41, PEEK(&BC80+2)
180 POKE &BC80+0, &C3
190 POKE &BC80+1, FNlsb(HIMEM+1)
200 POKE &BC80+2, FNmsb(HIMEM+1)
210 DATA &e5,&2A,&00,&00,&22,&80,&bc
220 DATA &3a,&00,&00,&32,&82,&bc
230 DATA &cd,&80,&bc,&21,&00,&00
240 DATA &22,&81,&bc,&21,&80,&bc
250 DATA &36,&c3,&e1,&d8,&c8,&fe,&1a
260 DATA &37,&3f,&c0,&b7,&37,&c9
```

Ist diese Routine einmal eingegeben, sollten Sie sie auf Diskette speichern. Wenn Sie nun ein Programm haben, das das Kommando CHAIN MERGE benutzt, so sollten Sie diese Routine an den Anfang des Programmes setzen. Nach dem Ausführen der Routine können Sie die entsprechenden Zeilen mit DELETE wieder löschen (auch im Programm möglich). Auch hier kann man wieder nur das sagen, was man bei allen Unterschieden der verschiedenen "kompatiblen" Schneider-Rechner sagen muß: Sind Sie Besitzer eines CPC 664 oder eines CPC 6128, so haben Sie zwar das Glück, diesen Fehler nicht mehr in Ihrem Betriebssystem vorzufinden, jedoch: Wollen Sie Software schreiben, die möglichst von mehreren Leuten benutzt wird, so müssen Sie aus Kompatibilitätsgründen immer vom ungünstigsten Fall ausgehen, und das ist in diesem Fall der CPC 464 mit seinen Schwächen.

5.2 Fehlermeldungen

Dieses Kapitel soll sich mit den Diskettenfehlermeldungen beschäftigen, die der Schneider-Rechner von Zeit zu Zeit zu melden hat. Da sowohl der CPC 664 als auch der CPC 6128 hier schon eine Möglichkeit bietet, die Fehlermeldungen (wenn auch recht unkomfortabel) abzufangen, wollen wir in diesem Kapitel nur ein Programmbeispiel für den CPC 464 anbringen. Im Kapitel 5.3 ist allerdings auch eine Fehlerabfangroutine für den CPC 664 und den CPC 6128 eingebunden. Da das Schema dieser Fehlerabfangroutine aber für alle drei Rechner identisch ist, sei es jedem empfohlen, dieses Kapitel dennoch zu studieren.

Dem einen oder anderen unter Ihnen treibt das Wort Fehlermeldung wahrscheinlich Zornesfalten auf die Stirn, wird er doch an viele vergebliche Versuche erinnert, die er unternommen hat, um sein Programm vor Fehlermeldungen vom Diskettenlaufwerk zu bewahren.

Fehlermeldungen sind allerdings etwas Selbstverständliches, wenn man mit Computern arbeitet. Jeder macht mal Fehler, die der Rechner dann Bildschirm anzeigt. Auch die Floppy nimmt nicht alles an, was man ihr übermittelt. Will man beispielsweise ein File laden, das es gar nicht gibt, so lautet ihre lapidare Anmerkung:

```
Filename.Ext not found
```

Sie wissen dann, daß Sie entweder den Filenamen falsch eingegeben haben, oder daß die falsche Diskette eingelegt wurde. Dabei gibt sich die Floppy alle Mühe, Ihnen zu dienen, bevor die Fehlermeldung ausgegeben wird. Wenn keine Extension (Dateityp) angegeben wird, so macht die DDI-1-Floppy drei Versuche, das gewünschte File zu finden.

1) Unter Filename.
2) Unter Filename.BAS
3) Unter Filename.BIN

Erst wenn alle diese Versuche fehlgeschlagen sind, gibt Ihnen die Floppy oben angegebene Fehlermeldung aus. Wenn Sie allerdings die Extension mit angeben, so wird nur ein Ladeversuch unternommen. Beispiele sind LOAD "Ballspie.len" oder OPENIN "Daten.dat" etc.

Wie immerhin einige Fehler zu vermeiden sind, ist im Kapitel 1.5 beschrieben. Jedoch kann man diese Programmierkniffe nicht immer erfolgreich anwenden.

Leider gibt es Fehler wie Disc missing u.ä., die man per Programm überhaupt nicht abfangen kann.

Dazu muß das Diskettenlaufwerk nicht unbedingt leer sein, es kann auch ausreichen, daß die Diskette nicht vollkommen eingerastet ist, und schon erhalten Sie die Fehlermeldung:

```
Drive A: disc missing

Retry, Ignore or Cancel?
```

Ein weiteres Beispiel: wenn Ihre Diskette nicht mehr die beste ist und ein *Read Fail* als Fehlermeldung erscheint. Die Liste der Möglichkeiten ließe sich beliebig verlängern. Das wäre ja alles noch gar nicht einmal so schlimm und vielmehr sogar ein Vorzug der Floppy, wenn es nicht einen entscheidenden Haken gäbe! Sollte während eines Programmablaufes ein Fehler von der Floppy erkannt werden, ganz gleich welcher, so wird Ihnen dieser Fehlertext auch auf dem Bildschirm angezeigt und der Programmablauf auf der Stelle unterbrochen. Nicht schlimm sagen Sie, das passiert mir ja auch, wenn ich einen Fehler im Programm habe. Dies ist zwar richtig, aber:

Programme kann man nahezu fehlerfrei machen. Es können beispielsweise Syntax-Fehler vollkommen ausgeschlossen, Fehleingaben vom Benutzer überprüft und abgefangen werden. Auf diese Art und Weise sind praktisch alle denkbaren Fehler während eines Programmablaufes abzufangen.

Des weiteren ist es möglich, das BASIC-Kommando ON ERROR GOTO zu verwenden, damit bei einer Fehlermeldung während des Programmablaufes dieses nicht abbricht, sondern lediglich unterbricht.

Bei einem Fehler wird der normale Programmablauf also unterbrochen, die Stelle zwischengespeichert, und der Computer springt in eine Unterroutine, die dann auf diesen Fehler eventuell reagieren kann. Eine solche Programmierung kann eine Unterbrechnung des Programmablaufes 100%ig abfangen.

Was wollen Sie aber machen, um im Programm abzufragen, ob nun eine Diskette im Laufwerk ist oder nicht? Sie können den Bediener des Programmes anweisen, die Diskette einzulegen, dies schließt aber eine Fülle weiterer möglicher Fehlermeldungen immer noch nicht aus. Nun gut, werden Sie sich sagen, dann kann ich mich ja des Kommandos ON ERROR GOTO bedienen, ist ja auch wirklich praktisch.

Leider hat ON ERROR GOTO bei Floppyfehlermeldungen keinerlei Wirkung. Die Fehlermeldungen werden trotzdem ausgegeben und das Programm abgebrochen.

(Denjenigen unter Ihnen, denen das Kommando noch nicht hinreichend bekannt ist, sei empfohlen, den Befehl im Benutzerhandbuch zu studieren.)

An dieser Stelle muß man allerdings folgendes bemerken: Der Schneider CPC 664 und der Schneider CPC 6128 bietet Ihnen die Möglichkeit, Fehlermeldungen durch das ON-ERROR-GOTO-Kommando abzufangen. Die Fehlervariable ERR enthält dann den Wert 32. Ferner beinhaltet die Variable DERR die vom DOS übergebene Fehlernummer. Das Programm wird also nicht abgebrochen, allerdings erscheint der Fehlertext auf dem

Bildschirm. Ein weiterer Nachteil ist, daß man nicht klar abfragen kann, um welchen Fehler es sich handelt, da ERR immer 32 enthält und die Variable DERR einem auch nicht allzuviel Auskunft darüber gibt.

Da ist guter Rat sicherlich teuer. Irgendwann wird auch der zäheste Programmierer aufgeben müssen. Sie können sich vorstellen, wie ärgerlich es ist, wenn Sie gerade 200 Datensätze eingegeben haben und diese auf Diskette abspeichern wollen, die Diskette aber ein

```
Disc full
```

ausgibt. Das Programm wird abgebrochen, und Sie können Ihre 200 Datensätze noch einmal eingeben. Ein solches Programm gibt natürlich nicht gerade ein professionelles Bild ab. Programme vor Fehlbedienung zu schützen, ist erstes Gebot bei professionellen Programmierern. Gerade aber die Floppy ist sehr fehleranfällig.

In diesem Kapitel finden Sie nun eine Maschinenspracheroutine, die dieses Problem Vergangenheit werden läßt. Fehler werden nicht mehr auf dem Bildschirm ausgegeben und Programme nicht mehr unterbrochen. Man hätte es so machen können, daß der Programmierer überhaupt nicht merkt, daß ein Fehler von der Floppy gemeldet wurde. Dies ist aber auch wieder nicht wünschenswert, denn den Fehler kann man ja ruhig erfahren, wichtig ist nur, daß das Programm nicht dabei abgebrochen wird. Bei dieser Routine ist dieses Problem so gelöst worden, daß bei einem Fehler von der Floppy die Fehlermeldung

```
Unknown user function
```

ausgegeben wird. Diese Fehlermeldung kommt eigentlich sehr selten vor und wurde deswegen auch ausgewählt, damit man Floppyfehler von anderen Fehlern im Programm unterscheiden kann. Alle Fehlermeldungen haben Fehlercodes, die man in der Standardvariablen ERR abfragen kann. Ebenso kann man die

Zeile abfragen, in der der letzte Fehler aufgetreten ist. Hierzu nutzt man die Standardvariable ERL.

Wenn also eine Fehlermeldung von der Floppy kommt, so wird diese abgefangen und in einem speziellen Puffer zwischengespeichert. Das Programm wird nicht abgebrochen und es wird der Fehler *UNKNOWN USER FUNCTION* mit Errorcode 18 generiert. Diesen kann man nun mit dem Befehl ON ERROR GOTO sehr wohl abfangen. In der entsprechenden Fehlerroutine ist dann abzufragen, welcher Fehler gemeldet wurde. Ist die Fehlercodevariable ERR gleich 18, so liegt ein Fehler von der Floppy vor. Man kann sich von der Routine den Text in eine Stringvariable übergeben lassen, um den genauen Fehler festzustellen. Im Programm kann man dann entsprechend auf den gemeldeten Fehler reagieren. Es ist möglich, diese Schutzroutine jederzeit ein- oder auszuschalten.

Ein Beispielprogramm

```
10 ON ERROR GOTO 1000
20 CALL ein : 'Fehlerroutine ein
30 OPENIN "zahlen"
40 WHILE NOT EOF
50   INPUT #9,a
60   PRINT a,
70 WEND
80 CLOSEIN
90 CALL aus : END
1000 IF ERR<>18 THEN RESUME NEXT
1010 ds$="+" : CALL msg,@ds$
1020 PRINT "Disk:";ds$
1030 RESUME 90
```

Sie erkennen, daß die Fehlerroutinen mit dem CALL-Befehl aufgerufen werden. Es gibt hier drei verschiedene Einsprungadressen:

Call ein — Mit Call ein schalten Sie die Fehlerabfangroutine ein. Es werden nun eine Menge von Vektoren des Betriebssystemes verbogen. Anders läßt sich dieses Problem nicht lösen. Ab sofort werden Fehlermeldungen nicht mehr auf dem Bildschirm ausgegeben, sondern abgefangen und gepuffert. Nach eine von der Floppy gemeldeten Fehler wird ein Unknown User function generiert. Diesen Fehler kann man dann mit ON-ERROR-GOTO abfangen.

Call msg,@a$ — Es wird die letzte gemeldete Fehlermeldung in die Stringvariable a$ (jede andere Stringvariable ist natürlich ebenso möglich) übergeben. Achten Sie darauf, daß vor dem Aufruf mit @a$ die Stringvariable a$ einmal eingerichtet worden ist, es reicht ein a$="", Sie erhalten ansonsten ein IMPROPER ARGUMENT.

Call aus — Die Fehlerabfangroutine ist wieder ausgeschaltet. Der Normalzustand ist wieder hergestellt. Ab sofort erscheinen Fehlermeldungen wieder auf dem Bildschirm. Es empfiehlt sich, in jedem Programm als letztes die Routinen wieder abzuschalten, da Sie sich sonst wohl wundern werden, wenn ein FILE NOT FOUND nicht mehr wie gewohnt gemeldet wird.

Es wird vorausgesetzt, daß bei der Benutzung dieses CALL-Kommandos sich die Fehlerroutine im Speicher befindet!

Erläuterungen zum Beispielprogramm:

Zeile Erläuterung

10 Es wird definiert, daß nach einer Fehlermeldung das Programm bei Zeile 1000 fortgeführt wird.

20 Die Fehlerroutine wird eingeschaltet. Ab sofort werden Fehlermeldungen von der Floppy nicht mehr auf dem Bildschirm angezeigt sondern abgefangen und gebuffert.

30 Das sequentielle File "zahlen" wird eröffnet. Sollte das File nicht existieren, so wird ein "File not found" generiert.

40-80 Wenn das File gefunden werden konnte, werden die Zahlen eingelesen und ausgegeben.

90 Es wird die Fehlerabfangroutine wieder ausgeschaltet und das Programm beendet.

1000 Hier wird abgefragt, ob es sich um einen Diskettenfehler handelt. Wenn nicht, so wird das Programm unmittelbar nach dem Fehler fortgeführt.

1010 Es wird die Variable ds$ angelegt. Dies ist nötig, um bei @ds$ kein "Improper argument" zu erhalten. Dann wird der gelieferte Fehlertext an die Variable ds$ übergeben.

1020 Der übergebene Text wird auf dem Bildschirm ausgegeben. Dies ist nur ein Verwendungszweck, es gibt Tausende.

1030 Das Programm wird in Zeile 90 fortgeführt.

Dies als ein Anwendungsbeispiel. Wie man mit der Fehlerroutine sehr nützlich arbeiten kann, können Sie auch dem Dateiverwaltungsprogramm in diesem Kapitel entnehmen.

Es hat sich als sehr praktisch und nützlich erwiesen, im Programm sogenannte *Flags* zu setzen und abzufragen. Man kann dann innerhalb des Programmes erkennen, ob ein Fehler aufgetreten ist oder nicht und dementsprechend darauf reagieren.

Dies könnte im Beispiel wie folgt aussehen:

```
10 ON ERROR GOTO 1000
20 CALL ein : 'Fehlerroutine ein
30 errflg=0 : OPENIN "zahlen" : IF errflg THEN 100
40 WHILE NOT EOF
50   INPUT #9,a
60   PRINT a,
70 WEND
80 CLOSEIN
90 CALL aus : END
100 IF RIGHT$(ds$,9)="not found" THEN REM Reagiere auf Meldung
110 PRINT "Disk -- ";ds$
120 END
1000 IF ERR<>18 THEN RESUME NEXT
1010 ds$="+" : CALL msg,@ds$
1020 errflg=1
1030 RESUME NEXT
```

In diesem Beispiel können Sie sehen, daß nach der OPENIN-Anweisung abgefragt wird, ob ein Fehler vorliegt. Liegt kein Fehler vor, so wird ganz normal im Programmablauf fortgefahren.

Sollte allerdings ein Fehler vorliegen, so wird abgefragt, ob es sich um einen "File not found" handelt. Wenn es ein File not found ist, so kann man entsprechend darauf reagieren (beispielsweise den Filenamen verändern). Anderenfalls wird die Fehlermeldung ausgegeben. Natürlich könnte man hier noch weitere Unterscheidungen machen. So könnte man bei einem *DRIVE A: DISC MISSING* den Benutzer im Klartext veranlassen, eine Diskette einzulegen.

Durch diese Art der Fehlerbehandlung kann man beispielsweise durch probeweises Öffnen eines Files feststellen, ob das File schon existiert oder nicht und dieses dann nötigenfalls anlegen.

Sie sehen, die Möglichkeiten sind sehr vielfältig, die Bedienung dennoch entsprechend einfach.

Sie können die Fehlerabfangroutine in jeden beliebigen Speicherbereich legen. Für die Maschinenspracheprogrammierer unter Ihnen ist ein Assembler-isting abgedruckt. Für alle anderen ist ein BASIC-Loader vorhanden, um die Maschinenroutine in jeden beliebigen Bereich zu verlegen. Die Routine ist 229 Bytes lang (man sollte hier besser kurz sagen). Sicherlich ein geringer Aufwand für den Nutzen, den man davon hat. In Zeile 70 des BASIC-Loaders können Sie die Startadresse eintragen, die das Programm haben soll. Sollten Sie ein Programm haben, in dem kein SYMBOL-AFTER-KOMMANDO vorkommt, so können Sie die Routine nach ganz oben im Speicher verlegen, um möglichst wenig Speicher für Ihr BASIC-Programm zu verlieren.

Im Kassetten-BASIC haben Sie 43533 Bytes für Ihre Programme und Daten zur Verfügung. Die obere Grenze des Speichers liegt bei &AB7F. Wenn Sie das Diskettenlaufwerk anschließen, so geht für das DOS und die Input- und Output-Buffer einiges an Speicher verloren. Nach dem Einschalten liegt HIMEM bei &A67B und Sie haben nur noch 42249 Bytes frei. Allerdings benötigt das DOS jetzt noch 4096 Bytes für die eben erwähnten Input- und Output-Buffer. Diese Buffer sind allerdings frei verschiebbar (nicht so der Systemspeicher). Wir legen diesen Speicher einmal an mit:

```
OPENOUT "dummy"
MEMORY HIMEM-1
CLOSEOUT
```

Wir erhalten jetzt für HIMEM den Wert &967A. Es sind noch 38152 Bytes frei. Die obere Grenze für BASIC liegt also im besten Falle bei &967A. Unsere Routine ist 229 Bytes lang und benötigt zusätzlich noch 40 Bytes als Buffer für den Text der Fehlermeldung. Wir rechnen also &967A-229-40=956D und lassen unsere Routine nun bei &956D beginnen. Tragen Sie diesen Wert in die Zeile 70 des Loaders ein.

Wenn Sie allerdings das Symbol After Kommando verwenden, so müssen wir die Routine ein wenig tiefer in den Speicher legen. Hierzu schalten Sie Ihren Rechner aus und wieder ein. Dann geben Sie

```
Symbol After n
```

ein. Für n setzen Sie hier den größten im Programm vorkommenden Wert ein. Dann geben Sie ein

```
OPENOUT "dummy"
MEMORY HIMEM-1
CLOSEOUT

PRINT HEX$(HIMEM-229-40)
```

Als Beispiel erhalten Sie bei Symbol After 190 als Ergebnis &93DD. Definieren Sie als Startadresse &93D0. Das ist alles.

Erstellen Sie sich auf diese Weise erst einmal zur Übung eine Routine, die bei &9650 liegt. Diese können Sie in den meisten Programmen verwenden, sofern Sie kein SYMBOL-AFTER-KOMMANDO verwenden, bzw. keine anderen Maschinenroutinen im Programm eingebunden sind. Speichern Sie dieses File dann auf Diskette ab, da es einfacher ist, in Programmen die Fehlerabfangroutine von Diskette einzulesen, als jedesmal die ganzen DATA-Zeilen in das Programm einzubinden. Wollen Sie nun die Fehlerabfangroutine in einem Programm verwenden, so beachten Sie folgende Reihenfolge:

1) Erstellen des binären Programmfiles mit dem Basic Loader Programm und Abspeichern des Programmes auf Diskette (natürlich auf die Diskette, auf der auch das Programm gespeichert ist, das diese Routine benötigt und lädt).

2) Einbinden der Routine in das Programm. Hierbei ist auch wieder die Reihenfolge wichtig, in der dies geschieht, da es ansonsten passieren kann, daß 4096 Bytes unnötig verloren gehen. Dies passiert dann, wenn zuerst die Routine geladen, dann die obere Speichergrenze herabgesetzt und danach erst der Buffer definiert wird.

Ein Programm, in dem die Fehlerabfangroutine gelesen würde, müßte etwa wie folgt aussehen:

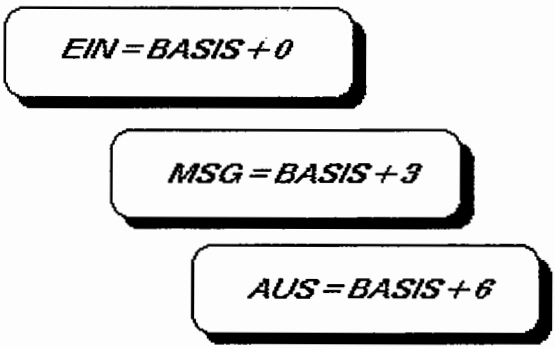
```
10 ON ERROR GOTO 20000 'Fehlerroutine
20 LOAD "ERROR.BIN"
30 MEMORY &9650-1
40 OPENOUT "dummy"
50 MEMORY HIMEM-1
60 CLOSEOUT
70  .
80  .
90  .
```

Sie erkennen die Reihenfolge. Zuerst muß einmal die Routine generiert und auf Diskette unter dem Namen "Error.bin" abgespeichert worden sein; natürlich ist jeder andere Name ebenso möglich, Sie können den Namen ja im Loader selbst auswählen.

Im Programm wird dann zuerst dieses File geladen, die Speichergrenze wird darauf folgend direkt unter unsere geladene Routine gesetzt, damit sie weder von Variablen noch von den DOS-Buffern überschrieben wird. Dann erst werden die DOS-Buffer fest angelegt und die Speichergrenze erneut heruntergesetzt. Jetzt kann die Routine nicht mehr überschrieben werden. Am besten ist es, wenn Sie den Rechner vor dem Gebrauch von solchen Programmen immer einmal aus- und wieder einschalten, da es zu Komplikationen kommen kann, wenn Sie zuvor ein Programm hatten, das den DOS-Buffer schon angelegt hat oder in igendeiner Form die obere Grenze heruntergesetzt hat. Nach-

dem dies alles geschehen ist, definieren Sie am besten noch vier Variablen, die Sie im Programm immer wieder benötigen werden.

```
EIN=&9650+0
MSG=&9650+3
AUS=&9560+6
DS$=""
```



Bei einer anderen Startadresse für die Routine müssen Sie die Werte natürlich entsprechend ändern. Es handelt sich hier um die Einsprungadressen für die verschiedenen Routinen. Wir wünschen Ihnen viel Erfolg beim Gebrauch dieser Routinen. Bevor Sie sich aber ans Eintippen des BASIC-Loaders stürzen: In diesem Kapitel befindet sich noch ein weiteres Routinenpaket, um die relative Dateispeicherung möglich zu machen. Diese Routinen wurden über die etwas komfortableren RSX-Kommandos verwirklicht. In diesem Routinenpaket ist die Fehlerabfangroutine mit eingebaut. Wenn Sie die relative Dateispeicherung also nutzen wollen, so tippen Sie lieber den BASIC-Loader der relativen Datenspeicherung ab, Sie haben sonst viel Arbeit umsonst vertan.

Besitzer vom CPC 664 und des CPC 6128, die die Fehlerabfangroutine unnötig finden, weil ihr BASIC dies unterstützt, dürfen drei wichtige Fakten nicht vergessen:

1) Die Fehlertexte erscheinen auf dem Bildschirm; dies ist für professionelle Programme sicherlich nicht sehr komfortabel.

2) Sie haben keine Kompatibilität zu den CPC 464-Rechnern.

3) Sie wissen zwar durch Abfrage der Variablen ERR, daß ein Floppyfehler vorliegt, erhalten aber nie die genaue Fehlermeldung, auch nicht durch Benutzung der Variablen DERR! So können Sie ein DISC-FULL nicht von der Fehlermeldung DISC IS MISSING unterscheiden.

5.2.1 Erläuterungen zur Fehlerabfangroutine - Wie wird es gemacht?

Um Fehlermeldungen abzufangen, sind einige Kenntnisse des Interieurs vom CPC notwendig, da ein nicht zu unterschätzender Eingriff in das Innenleben des Betriebssystemes vorgenommen werden muß. Wir greifen dazu auf die Möglichkeit des CPC zurück, bestimmte Vektoren zu verbiegen. Hiervon müssen wir starken Gebrauch machen.

Wenn die Floppy eine Fehlermeldung auf dem Bildschirm ausgibt, so wird die Betriebssystemroutine TXT OUTPUT bei &BB5A aufgerufen. Diese Routine dient dazu, ein Zeichen im Akku auf dem Bildschirm im aktuellem Fenster auszugeben. Diese Routine wird umgebogen, in der Fachsprache nennt man diesen Vorgang *patchen.*

Die Fehlerabfangroutine wird also vor die normale *TXT-OUTPUT*-Routine gesetzt und ausgeführt, und das bei jedem Zeichen, das auf dem Bildschirm erscheinen soll. In dieser Routine wird überprüft, ob das zu druckende Zeichen von der Floppy kommt. Dies macht man, indem man auf dem Stack die Rücksprungadresse überprüft. Das Floppy-ROM liegt im Bereich über &C000. Hier liegt auch das BASIC-Betriebssystem. Die Routine zur Übergabe von Fehlermeldungen liegt im Bereich &CB00-&CBFF, also brauchen wir lediglich das higher Byte der Rücksprungadresse abzufragen. Wird also TXT OUTPUT aus diesem Bereich aufgerufen, so handelt es sich um ein Zeichen aus der Floppy, das von der Routine abgefangen und zur späteren Verwendung gebuffert wird.

Gleichzeitig wird ein Flag gesetzt, um nachher überprüfen zu können, ob ein Fehler aufgetreten ist. Weiterhin dient das Flag

dazu, die BREAK-Meldung abzufangen. Die Meldung BREAK kommt nämlich nicht mehr aus dem Floppy-ROM, sondern aus dem "normalen" BASIC-ROM. All diese Dinge werden in der Routine OUTCHK (siehe Assembler-Listing) vollzogen. In dieser Routine werden die Vektoren gepatcht (verbogen), in der Routine RESET werden diese Vektoren wieder auf Normalwert gesetzt.

Wenn die Floppy die Fehlermeldung ausgegeben hat, springt sie in das BASIC-ROM, und zwar an die Stelle, an der Programme unterbrochen werden. Es wird die BREAK-Meldung ausgegeben und das Programm unterbrochen. Hier müssen wir den Vektor Ready Modus patchen, um zu verhindern, daß Programme unterbrochen werden können. In dieser Routine wird abgefragt, ob das Errorflag gesetzt ist, denn man kann ja durchaus in den Ready Modus gelangen, ohne daß eine Fehlermeldung aufgetreten ist. Ist das Fehlerflag gesetzt, so wird die Fehlermeldung #18 (Unknown User Function) generiert, die man dann abfangen kann.

Schließlich haben wir noch die Routine *GETTXT*, die dazu dient, uns die Fehlermeldung in BASIC zu übergeben, wir erhalten also die Fehlermeldung in Klartext in einer beliebigen Stringvariablen.

Die Routine KWC dient dazu, für die möglich Abfrage

```
Retry, Ignore or Cancel
```

ein *C* für *Cancel* zu senden. Auf diese Weise wird der Fehler sofort erkannt, und der Programmablauf sofort unterbrochen.

ACHTUNG! WICHTIG!!

Wenn Sie sich das Inhaltsverzeichnis einer Diskette ausgeben lassen, so werden diese Zeichen auch als Fehler interpretiert, da sie exakt aus demselben Speicherbereich kommen wie die Fehlermeldungen. Dies gilt sowohl für den IDIR- als auch für den CAT-Befehl. Wenn die Fehlerroutine eingeschaltet ist, so

wird der Befehl IDIR unwirksam! Wenn Sie dieses Kommando benutzen wollen, so müssen Sie

```
CALL aus : IDIR : CALL ein
```

verwenden. Der CAT-Befehl ist hier speziell in der Routine behandelt worden. Auch dessen Vektoren werden gepatcht, so daß das CAT-Kommando wie normal zu benutzen ist.

Der BASIC-Lader der Fehlerabfangroutine für CPC 464:

```
10 '*********************************************
20 '*** Fehlerunterdrueckungsroutine ************
30 '*** (C) 1985 by DATA BECKER GmbH JS 22/3/85 *
40 '*********************************************
50 '
60 DEFINT a-z
70 adresse = &A000 : 'Startadresse fuer Routine
80 :
90 DATA &C3,&09,&51,&C3,&83,&51,&C3,&AC,&51,&3E,&C3,&32,&01,&AC,&32,&5A
100 DATA &BB,&32,&06,&BB,&32,&9B,&BC,&21,&60,&51,&22,&02,&AC,&21,&30,&51
110 DATA &22,&5B,&BB,&21,&74,&51,&22,&07,&BB,&21,&D1,&51,&22,&9C,&BC,&C9
120 DATA &E3,&F5,&7C,&FE,&CB,&20,&1E,&3E,&01,&32,&E3,&51,&F1,&E5,&2A,&E4
130 DATA &51,&FE,&0A,&28,&0B,&77,&23,&7D,&FE,&28,&20,&01,&2B,&22,&E4,&51
140 DATA &E1,&F5,&F1,&E3,&C9,&3A,&E3,&51,&B7,&20,&F7,&F1,&E3,&CF,&00,&94
150 DATA &3A,&E3,&51,&B7,&C8,&AF,&32,&E3,&51,&21,&E6,&51,&22,&E4,&51,&1E
160 DATA &12,&C3,&94,&CA,&F5,&3A,&E3,&51,&B7,&20,&04,&F1,&CF,&3C,&9A,&F1
170 DATA &3E,&43,&C9,&1E,&02,&FE,&01,&C0,&DD,&5E,&00,&DD,&56,&01,&0E,&FF
180 DATA &21,&E6,&51,&7E,&FE,&0D,&23,&28,&FA,&2B,&E5,&2B,&0C,&23,&7E,&FE
190 DATA &0D,&20,&F9,&79,&12,&E1,&EB,&23,&73,&23,&72,&C9,&3E,&C9,&32,&01
200 DATA &AC,&3E,&CF,&32,&5A,&BB,&32,&06,&BB,&3E,&DF,&32,&9B,&BC,&21,&00
210 DATA &94,&22,&5B,&BB,&21,&3C,&9A,&22,&07,&BB,&21,&8B,&A8,&22,&9C,&BC
220 DATA &C9,&F5,&E5,&CD,&AC,&51,&E1,&F1,&CD,&9B,&BC,&E5,&F5,&CD,&09,&51
230 DATA &F1,&E1,&C9,&00,&E6,&51
```

```
240 :
260 FOR i=adresse TO adresse+&E5
270  READ a
280  POKE i,a
290  s=s+a
300 NEXT
310 IF s<>28065 THEN PRINT CHR$(7)"*** Fehler in DATAS *** " : END
320 :
330 DATA &01,&09,&04,&83,&07,&ac,&18,&60,&1e,&30,&24,&74,&2a,&d1,&3a,&e3
340 DATA &3f,&e4,&4e,&e4,&56,&e3,&61,&e3,&67,&e3,&6a,&e6,&6d,&e4,&76,&e3
350 DATA &91,&e6,&d4,&ac,&de,&09,&e4,&e6
360 :
370 FOR i=1 TO 20
380  READ of1,of2
390  ad1=adresse+of2
400  POKE adresse+of1,ad1 AND 255 : 'lower Byte
410  POKE adresse+of1+1,INT(ad1/256) AND &FF
420 NEXT
430 :
440 INPUT "Soll File gesichert werden ? (J/N) ",a$
450 IF UPPER$(a$)="J" THEN INPUT "Filename : ",b$ : SAVE
b$,b,adresse,&E8
440 INPUT "Soll File gesichert werden
```

5.3 Relative Dateiverwaltung

In Kapitel 1.5 haben Sie gelernt, wie man sequentielle Files programmiert und in Kapitel 1.6 entsprechend die Theorie der relativen Datenspeicherungstechnik - Sie sind nun sicherlich schon neugierig auf die Realisierung der relativen Dateiverwaltung, die wirklich einige bemerkenswerte Vorteile bietet. Allerdings ist diese Art der Programmierung nicht immer optimal: Insbesondere bei kleineren Anwendungen wäre ihr Einsatz oftmals unsinnig, da sie mehr Zeit und Platz in Anspruch nähme als die sequentielle Programmierung. Erst wenn häufig Korrekturen und Zugriffe auf die Datei nötig werden, macht sich die relative Dateiverwaltung schnell bezahlt. Neben den unterschiedlichen Ansprüchen, die relative und sequentielle Datenspeicherung an den Anwender stellen, gibt es noch weitere gravierende Unterschiede.

Zunächst ist es nicht mehr nötig, ein File komplett in den Speicher zu lesen, um es bearbeiten zu können. Auch komplizierte Schleifen zum Suchen oder Lesen eines bestimmten Datensatzes aus dem File können Sie vergessen. Dafür gibt es neue Regeln, an denen wir uns orientieren müssen: So ist vorher sehr sorgfältig zu überlegen, wie lang ein Datensatz maximal sein wird (wieviel Zeichen hat unser Datensatz?), und welche Anzahl von Datensätzen unsere Datei beinhaltet (wieviel Datensätze wollen wir bearbeiten?). Auf diese Vorüberlegung konnten Sie bislang bei der sequentiellen Dateiverwaltung verzichten - obschon jeder seriöse Programmierer in jedem Fall die Größe seiner Datei vor Beginn der Arbeit abschätzen wird, schon aus dem Grunde, um einen Überblick darüber zu erhalten, wie viele Datensätze auf eine Diskette passen.

Wir wollen einmal anhand unseres Beispiels aus Kapitel 1.5 eine solche Berechnung durchgehen - Sie erinnern sich sicherlich noch an unser Telefonregister, mir den drei Datenfeldern:

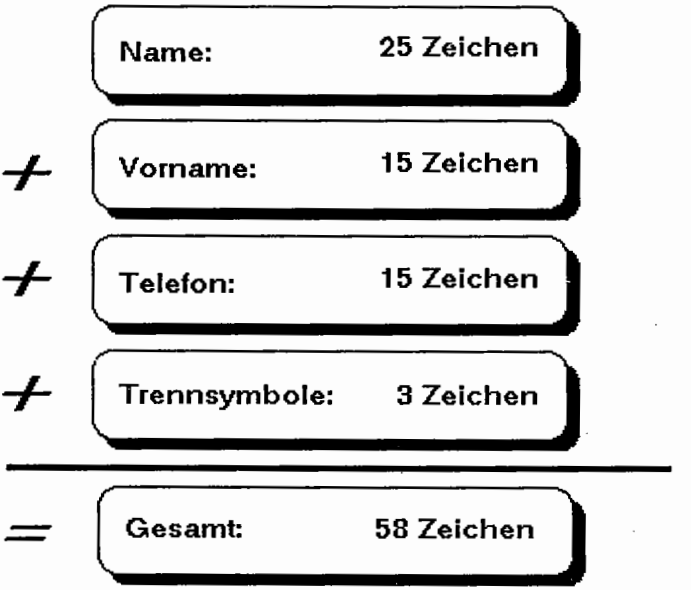
Feld 1) Name
Feld 2) Vorname
Feld 3) Telefonnummer

Nehmen wir einmal an, daß Sie 50 solcher Telefonnummern speichern wollen, dann haben wir also 50 *Datensätze.*

Um gegebenenfalls weitere Telefonnummern aufnehmen zu können, richten wir 100 Datensätze ein. Nun müssen wir noch die Länge der Datensätze bestimmen. Dies macht man, indem jedem Datenfeld eine maximale Länge zugeordnet wird, die man dann im Programm nicht mehr überschreiten darf. Sie müssen also dafür Sorge tragen, daß diese Grenzen eingehalten werden, da es sonst zu Komplikationen kommt. Das ist keine Schikane und auch nicht unkomfortabel: Sie werden diese Einschränkung bei allen relativen Dateiverwaltungen finden, selbst auf den größten Rechnern. Die Datensatzlänge ist ein wichtiger Bestandteil für den reibungslosen Ablauf der Berechnungen, wo ein bestimmter Datensatz zu finden ist.

Bei "unserer" relativen Dateiverwaltung haben Sie sogar ein wenig Spielraum, Sie können die Grenzen einzelner Datenfelder ruhig überschreiten, wenn Sie dafür ein anderes Datenfeld dementsprechend kürzer halten. Das wurde gemacht, um die Dateiverwaltung etwas flexibler zu gestalten, denn das ist nicht bei allen Systemen so! Sie müssen nur darauf achten, daß die Gesamtlänge nicht überschritten wird. Doch zurück zu unserem Beispiel:

Wir müssen jetzt die Maximallängen zuordnen, d.h. die Anzahl der Zeichen, die jedes Datenfeld maximal haben darf. Nehmen wir einmal folgende Zuordnungen als Beispiel:



Der Name darf also maximal 25 Zeichen lang sein, was wohl sogar bei Doppelnamen ausreichen dürfte, der Vorname ist auf 15 Zeichen begrenzt. Als letztes haben wir dann noch für das dritte Feld, die Telefonnummer, höchstens 15 Zeichen vorgesehen. Insgesamt ergibt dies 55 Zeichen, die wir pro Datensatz benötigen. Sie müssen zu diesen 55 Zeichen noch 3 addieren, da wir drei Datenfelder haben, die voneinander getrennt werden müssen. Das Trennsymbol wäre nicht notwendig, wenn die Datenfelder fixe Längen hätten, da dies aber nicht der Fall ist, benötigen wir auch in der relativen Dateiverwaltung ein Trennsymbol. Wir kommen also auf diese Weise auf eine Länge von

58 Zeichen.

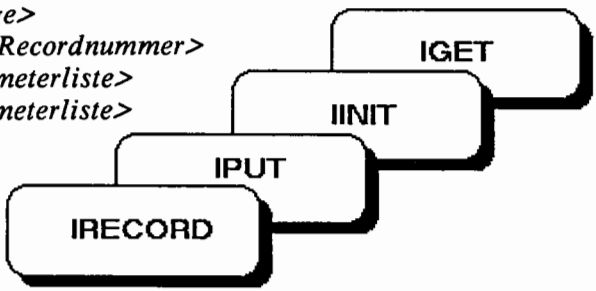
Um die abgedruckte Routine für die relative Dateiverwaltung aber nicht zu groß werden zu lassen - Sie müssen sie ja schließlich auch abtippen - wurde eine Einschränkung gemacht: Die Datensatzlänge muß auf die folgende zweier-Potenz aufgerundet werden. In unserem Beispiel bedeutet dies, daß wir die Datensatzlänge 64 angeben werden.

Maximale Datensatzlänge ist 512 Bytes, Sie haben also die Auswahl unter den Datensatzlängen:

2, 4, 8, 16, 32, 64, 128, 256 und 512.

Damit stehen Ihnen neun verschiedene, mögliche Datensatzlängen zur Verfügung, unter denen Sie wählen können. Sollte Ihre errechnete Datensatzlänge beispielsweise 32 sein, so haben Sie Glück und können auch 32 als Länge angeben. Sollte es nur ein Zeichen mehr sein, so müßten Sie entweder die Datensatzlänge 64 wählen, oder noch einmal überlegen, ob Sie nicht irgendein Datenfeld noch um ein Zeichen kürzen können. In unserem Beispiel haben wir die Länge 58 errechnet. Die Differenz zwischen 64 und 58 ist 6; Sie würden also 6 Bytes pro Datensatz zu viel haben, um nicht zu sagen "verschwenden". Sie können sich auf diese Datensatzlänge ja einstellen und das eine oder andere Datenfeld noch um ein Zeichen verlängern. Sicherlich kann man mit dieser Einschränkung leben. Die bisher verwandten Befehle für die sequentielle Dateiverwaltung können Sie bei der relativen Dateiverwaltung beruhigt vergessen, bis auf das OPENIN- und CLOSIN-Kommando, dafür kommen vier neue Befehle hinzu. Es sind die Kommandos:

IINIT,<Länge>
IRECORD,<Recordnummer>
IPUT,<Parameterliste>
IGET,<Parameterliste>



Das Kommando GET ist vergleichbar mit dem LINE-INPUT-Kommando, also zum Lesen von Daten, PUT dient dem Schreiben von Daten, also vergleichbar mit PRINT.

Wenn ein Datenfeld Zahlen beinhalten soll, so müssen diese Zahlen mit dem STR$-Kommando in einen String umgewandelt werden. Bei reellen Zahlen ist hier die Datenfeldlänge 12, bei Integerzahlen beträgt sie lediglich 9 Zeichen.

Da wir bei der DDI-1-Floppy standardmäßig keine relativen Files haben, werden diese über die sequentiellen Files simuliert. Bevor wir mit der relativen Datei arbeiten können, muß diese erst einmal "eingerichtet" werden. Wir realisieren das, indem wir ein sequentielles File erstellen, in dem praktisch nichts steht. Erinnern wir uns, unsere Beispieldatei soll 100 Datensätze mit einer Länge von 64 Zeichen beinhalten. Wir benötigen also einen effektiven Speicherplatz von 100*64=6400 Bytes. Das Einrichten einer solchen Datei ist sehr einfach:

```
10 REM ****************************
20 REM   Einrichten der REL-Datei
30 REM ****************************
40 :
50 OPENOUT "Datei.Rel"
60 FOR i=0 TO 100
70   PRINT #9, STRING$(64,13);
80 NEXT
90 CLOSEOUT
```

In der Schleife in Zeile 60 muß die Anzahl der vorgesehenen Datensätze berücksichtigt werden, in unserem Beispiel 100. In der Zeile 70 wird dann der Platz für jeden Datensatz geschaffen, wir füllen unsere (noch) sequentielle Datei also nur mit dem Trennzeichen <Carriage Return>. (Sie könnten auch jedes andere beliebige Zeichen wählen, jedoch ist das Trennzeichen <Carriage Return> zu empfehlen, da es sonst unter Umständen zu Problemen kommen kann). Es ist nicht unbedingt notwendig, daß Sie in Zeile 70 mit einem Semikolon abschließen, aber dann stimmt die Berechnung vollkommen. Nach diesen wenigen Zeilen haben wir genug getan, unsere REL-Datei ist erstellt und kann nun bearbeitet werden.

Übrigens: Sollte Ihr Datensatz länger als 255 Bytes sein, so ist es nicht möglich, diese Datensatzlänge in *einem* STRING$-Kommando auszudrücken. Teilen Sie dann diese Datensatzlänge bitte in zwei STRING$-Kommandos auf, beispielsweise für die Datensatzlänge

256 STRING$(128,13);STRING$(128,13);
512 STRING$(200,13);STRING$(200,13);STRING$(112,13);

Anders geht dies leider nicht, da das Betriebssystem dies für das STRING$-Kommando nicht zuläßt.

Weiterhin müssen Sie darauf achten, daß die Dateigröße 131072 Zeichen (das ist 2^17) nicht überschreitet, da für größere Dateien die Berechnungen fehlen.

Wir können jetzt 100 Datensätze in unsere Datei schreiben und fortan hat jeder Datensatz eine Nummer, die ihn eindeutig bestimmt. Einen Datensatz nennen wir in der Fachterminologie auch RECORD, so daß jeder Datensatz also durch seine Recordnummer bestimmbar ist. Der erste Record hat die Recordnummer 0.

Wenn Sie eine REL-Datei eröffnen, so können Sie nach wie vor das OPENIN-Kommando verwenden. Eine eröffnete REL-Datei kann zwar gleichzeitig gelesen und beschrieben werden, es heißt zum Eröffnen trotzdem immer OPENIN!!

Mit dem Kommando IINIT wird die Recordlänge bestimmt, also die Länge eines jeden Datensatzes. Dies sollte unmittelbar nach der Eröffnung des Files geschehen, OPENIN und IINIT gehören zusammen wie siamesische Zwillinge. Das Kommando IINIT hat einen Parameter, Sie können sowohl Konstanten verwenden (was wohl meistens der Fall sein dürfte) als auch Variablen. In unserem Beispiel müßten Sie

```
IINIT,64
```

eingeben. Es wäre aber auch die variable Möglichkeit denkbar:

```
Recordlaenge=64 : IINIT,Recordlaenge
```

Mit dem Kommado IRECORD bestimmen Sie den Record, der als nächstes bearbeitet werden soll; hierbei ist es egal, ob Sie diesen Record lesen oder schreiben wollen. Der erste Record, also der erste Datensatz, hat die Recordnummer 0 - wenn Sie sich nicht an diese Reihenfolge gewöhnen wollen, so können Sie den ersten Record ja "brach" liegen lassen oder als außerordentlichen Speicher verwenden. Auch das IRECORD-Kommando hat einen Parameter, der ebenfalls eine Konstante oder eine Variable sein kann. Beim IRECORD-Kommando ist wohl die Variable wahrscheinlicher. Um beispielsweise den 35. Datensatz zu positionieren, geben Sie ein:

IRECORD,35

Die Kommandos IGET und IPUT sind von der Syntax und der Anwendung her identisch. Das Kommando IGET holt Daten aus dem ausgewählten Record und das Kommando IPUT schreibt Daten in den ausgewählten Record. Die Kommandos IGET und IPUT haben eine beliebige Anzahl von Parametern, wenigstens einen, aber höchstens 32. Ohne Parameter gibt es zwar keine Fehlermeldung, das Kommando ist aber effektlos. Es dürfen ausschließlich Stringvariablen übergeben werden. Konstanten, die geschrieben werden sollen, müssen vorher einer Stringvariablen zugewiesen werden. Nehmen wir einmal an, wir wollen das Wort "Pferd" in den 23. Record schreiben, so ist folgender Befehlsablauf notwendig:

```
10 OPENIN "Datei.REL"
20 IINIT,64
30 IRECORD,23
40 a$="Pferd"
50 IPUT,@a$
60 IINIT,0
70 CLOSEIN
```

An diesem Beispiel können wir gleich zwei Eigenheiten erkennen. Bevor eine relative Datei geschlossen wird, müssen Sie das Kommando IINIT,0 geben - erst dann dürfen Sie die Datei schließen. Das hat den Sinn, daß eventuell noch im Speicher vorhandene Daten vor dem Schließen der Datei auf Diskette geschrieben werden. Eine zweite Eigenheit erkennen Sie bei dem IPUT-Befehl: Der String kann nicht einfach nach dem Komma angegeben werden, Sie müssen @ voransetzen. Dies ist keine Schikane der Autoren dieses Buches, sondern ein Manko des Betriebssystemes (siehe auch Kapitel 1), das diese Möglichkeit nämlich als einzige Schnittstelle zwischen BASIC und Maschinensprache vorsieht. Jedoch sollte Sie dieses kleine "Hindernis" nicht allzusehr stören, man hat sich sehr schnell daran gewöhnt. Wollen Sie mehr als einen Parameter in der IPUT-Anweisung übergeben, so können Sie diese durch Komma getrennt anhängen. Beispielsweise wollen wir noch das Wort "Katze" in unseren Record schreiben:

```
10 OPENIN "Datei.REL"
20 IINIT,64
30 IRECORD,23
40 a$="Pferd"
45 b$="Katze"
50 IPUT,@a$,@b$
60 IINIT,0
70 CLOSEIN
```

Es spielt allerdings keine Rolle, ob Sie nun alle Datenfelder eines Records in einer IPUT-Anweisung schreiben, oder ob Sie dies in mehreren Anweisungen erledigen, denn interne Pointer sorgen schon für einen reibungslosen Ablauf. Genauso einfach,

wie wir die Daten schreiben, können wir sie auch wieder lesen, und zwar mittels des IGET-Kommandos.

Hier können Sie eine beliebige Anzahl an Stringvariablen als Parameter übergeben. Ebenso ist es egal, ob Sie alle Datenfelder in einer Zeile einlesen, oder in mehreren. Wir wollen "unseren" Record nun wieder einlesen:

```
10 OPENIN "Datei.REL"
20 IINIT,64
30 a$="" : b$=""
40 IRECORD,23
50 IGET,@a$,@b$
60 PRINT a$,b$
70 IINIT,0
80 CLOSEIN
```

Wenn Sie der IGET-Routine Stringvariablen als Parameter übergeben, so müssen diese mindestens schon einmal im Programm vorgekommen sein - ansonsten meldet sich der BASIC-Interpreter mit der Fehlermeldung

```
Improper argument
```

Das ist eigentlich schon alles, was Sie bedenken müssen. Um die Routinen nicht unnötig zu verlängern, wurde auf die meisten Fehlerabfragen verzichtet. Sie müssen im Programm also selbst Sorge dafür tragen, daß Sie nicht zu große Recordnummern angeben (hier kann einiges Unvorsehbares geschehen), ebenso ist darauf zu achten, daß Sie die maximale Datensatzlänge nicht überschreiten, da sonst in den nächsten Datensatz hineingeschrieben wird. Wenn Sie eine REL-Datei eröffnet haben, können Sie keine OUTPUT-Datei mittels OPENOUT öffnen; Sie dürfen also "nur" die relative Datei offen haben. Dies ist aber verständlich, da Sie ja in die REL-Datei gleichzeitig lesen und schreiben können.

Achten Sie darauf, daß Sie den Input-/Outputbuffer mit dem Kommando

```
OPENOUT "Dummy" : MEMORY HIMEM-1 :
CLOSEOUT
```

am Anfang des Programmes anlegen und sichern, ansonsten kann es zu Komplikationen während des Betriebes mit den Floppyerweiterungsbefehlen kommen. Nehmen Sie das abgedruckte Beispielprogramm zur relativen Dateiverwaltung zur Hand, wenn Sie selbst mit der Routine arbeiten wollen. Sie können dem Beispielprogramm entnehmen, wie und in welcher Reihenfolge Sie die einzelnen Befehle verwenden sollten.

Verwenden Sie nicht das CAT-Kommando - es schließt die Datei, ohne eventuell notwendige Daten noch auf Diskette zu sichern, und überschreibt den Buffer.

Wenn Sie diese Regeln beachten, werden Sie viel Erfolg mit der Routine haben. Die REL-Datei kann beliebige Ausmaße annehmen und sich theoretisch über die ganze Diskette erstrecken. Die Zugriffszeiten sind sehr kurz, das Beispielprogramm beweist dies. Neben den Befehlen für die relative Dateiverwaltung wurden auch die Befehle für die Fehlerabfangroutine integriert (siehe Kap. 5.2). Die Kommandos sind genauso zu handhaben, wie in Kapitel 5.2 beschrieben. Aus CALL ein wird IAN, aus CALL aus wird IAUS und aus CALL msg wird IERR. Sie haben also noch die drei weiteren Befehle:

```
IAN
IAUS
IERR ,@<Stringvariable>
```

Mit diesen sieben neuen Befehlen sind alle Unzulänglichkeiten der Floppy ausgebügelt; Sie können nun fehlerfreie und komfortable Programme erstellen, ohne kompliziert tricksen zu müssen. Es ist sowohl ein Assemblerlisting als auch ein BASIC-Loader abgedruckt. Wie Sie die Adresse berechnen, an der die

Erweiterung stehen soll, entnehmen Sie den Beschreibungen in Kapitel 5.2. Die Länge dieser Routine beträgt &34a = 842 Bytes. Im Normalfall (d.h. bei Programmen dem Kommando SYMBOL AFTER) können Sie als Basisadresse &9300 verwenden.

Da man dem Maschinenprogramm entnehmen kann, wie das Problem genau gelöst ist, sei hier lediglich die Art und Weise erläutert, wie die relative Dateiverwaltung realisiert worden ist.

5.3.1 Erläuterungen zum Maschinenprogramm

Wenn wir eine relative Datei haben wollen, so muß diese erst einmal "angelegt" werden. Dazu erstellen wir zunächst ein sequentielles File mit entsprechender Länge. Dadurch schaffen wir den gewünschten Platz auf der Diskette und belegen die nötige Anzahl von Blocks (siehe dazu auch Kapitel 2). Die belegten Blocks werden im Disketteninhaltsverzeichnis eingetragen, jeweils 16 KByte mit einem Eintrag. Wenn wir jetzt unsere relative Datei öffnen, wird durch das Kommando IINIT,RL festgestellt, welche Blöcke von der Datei belegt werden. Es werden alle Blocknummern gespeichert (1 Block sind zwei aufeinander folgende Sektoren). Diese Tabelle ist Grundlage für die relative Dateiverwaltung, da man sonst nicht frei auf die verschiedenen Records zugreifen könnte. Zusätzlich wird noch die Recordlänge separat gespeichert, aber Vorsicht: Es wird nicht geprüft, ob die Recordlänge auch wirklich eine Zweierpotenz ist. Selbst wenn Sie eine relative Datei eröffnen, die dieselbe Recordlänge hat wie eine andere zuvor geöffnete relative Datei, so muß unbedingt nach dem OPENIN-Kommando das INIT-Kommando folgen, da sonst noch die alten Blocknummern im Speicher vorliegen!

Aus den gespeicherten Blocknummern kann man dann jeweils den entsprechend angesprochenen Sektor berechnen, und zwar im Kommando IRECORD. Die Routine ist so programmiert, daß ein Sektor nur dann nachgeladen wird, wenn es unbedingt erforderlich ist. Es kann ja sein, daß er sich schon im Speicher befindet; dies ist besonders dann der Fall, wenn Sie alle Records vom ersten bis zum letzten "durchlesen" mit dem Ergebnis, daß

sehr viel Zeit gespart wird. Zusätzlich muß, bevor ein Sektor gelesen wird, festgestellt werden, ob der aktuell im Speicher befindliche Sektor mit IPUT beschrieben worden ist. Ist dies der Fall, so wird der alte Sektor erst gesichert, bevor der neue geladen wird.

Bei dem Befehl IPUT werden alle angegebenen Stringvariablen in den Sektorbuffer kopiert, bei dem Befehl IGET werden lediglich die Zeiger der Stringvariablen auf den Sektorbuffer verbogen, ebenso wird deren Länge verändert.

Dies ist in kurzen Worten schon das Prinzip, das Sie in der Routine verwirklicht sehen. Wir haben uns für die RSX-Kommandos entschieden, weil Sie etwas leichter zu handhaben und gleichzeitig aussagekräftiger sind als irgendwelche CALLs. Außerdem müssen Sie sich (fast) keine Gedanken darüber machen, in welchem Speicherbereich die Routine liegt.

Wenn Sie die Listings nicht abtippen wollen, so sei hier kurz angemerkt, daß es eine *Diskette zum Buch* gibt, die alle hier im Buch vorkommenden Programme beinhaltet.

Das Assembler-Listing für die Erweiterungskommandos:

```
ORG       #8000
;
;
;Relative Dateiverwaltung mit den Schneider-Rechnern
;************************************************
;
;RSX-Kommandos, die hinzukommen:
;
;AN
;AUS
;ERR
;RECORD
;GET
;PUT
;INIT
;

;JS 12/2/1986


LD        A,(VER)       ;ABSTURZ BEI DOPPELTEN IAN
OR        A             ;VERHINDERN
RET       NZ
CALL      #B900         ;ROM ENABLE
PUSH      AF
LD        A,(#DE01)     ;VERSION
LD        (VER),A       ;MERKEN
POP       AF
CALL      #B90C         ;RESTORE
LD        A,(VER)
CP        #71
JR        Z,RSXON       ;464
LD        A,#DF
LD        (A1),A
LD        (A2),A
LD        A,(VER)
CP        #C9           ;6128?
JR        NZ,M02        ;664 RECHNER
```

```
LD        HL,TABU4
JR        M02+3
M02:LD    HL,TABU2
LD        (A1+1),HL
LD        (A2+1),HL
RSXON:LD  BC,RSX          ;ERWEITERUNGEN
LD        HL,KERNAL       ;4 BYTES RAM FUER SYSTEM
JP        #BCD1           ;ERWEITERUNGEN EINBINDEN
;
RSX:DEFW  TABLE           ;ADRESSE DER BEFEHLSWORTE
JP        RECORD
JP        GET
JP        PUT
JP        INIT            ;LAENGE FESTLEGEN
JP        SET             ;EINSCHALTEN
JP        RESET           ;AUSSCHALTEN
JP        GETTXT          ;HOLE TEXT
;
TABLE:DEFM   "RECOR"
DEFB      "D" + #80
DEFB      "G","E","T"+#80
DEFB      "P","U","T"+#80
DEFB      "I","N","I","T"+#80
DEFB      "A","N"+#80
DEFB      "A","U","S"+#80
DEFB      "E","R","R"+#80
DEFB      0               ;ENDE TABELLE
;
KERNAL:DEFS  4            ;SPEICHER FUER KERNAL
;
RECORD:CP  01             ;1 PARAMETER
RET       NZ              ;FALSCHE ANZAHL
LD        B,(IX+1)        ;HOLE RECORD#
LD        C,(IX)
LD        HL,(RECLEN)     ;RECORDLENGTH
A0:CALL   MULTI           ;DE=HL*BC
EX        DE,HL
LD        DE,512          ;LAENGE EINES SECTORS
A1:CALL   #BDC1           ;HL=HL/DE -- DE=REST
LD        A,(FL0)
```

```
LD        (OFFSET),DE   ;OFFSET MERKEN
LD        DE,128        ;128 BEI CARRY ADDIEREN
OR        A             ;SETZEN DER FLAGS Z=0 BEDEUTET KEINEN UEBERTRAG
JR        Z,NOC
ADD       HL,DE
NOC:LD    (SECNR),HL    ;MERKE SECTORNR
XOR       A ;A:=0
LD        (POINTER),A   ;POINTER AUF NULL
LD        (POINTER+1),A
LD        DE,(OLDSEC)
SBC       HL,DE         ;GLEICHER SECTOR?
RET       Z             ;BEI NULL IN BUFFER
CALL      SAVE          ;VORHER BUFFER SICHERN
LD        HL,(SECNR)    ;HOLE SECTORNR
LD        (OLDSEC),HL   ;SECTOR MERKEN
RR        H ;/2
RR        L ;/2
PUSH      AF            ;MERKE CARRY
LD        DE,(#A79B)    ;BLOCKTABELLE
ADD       HL,DE         ;HOLE BLOCKNR
LD        A,(HL)        ;BLOCKNR
LD        L,A
LD        H,0           ;HL:=BLOCKNR
ADD       HL,HL         ;*2
POP       AF            ;HOLE CARRY
LD        DE,0
ADC       HL,DE         ;ADDIERE CARRY
LD        DE,9
A2:CALL   #BDC1         ;HL:=HL/DE --- DE:=REST
INC       HL            ;+1
INC       HL            ;+1 = TRACK
LD        A,L           ;TRACK
LD        (TRACK),A     ;MERKEN
LD        A,E           ;SECTOR
ADD       A,#41         ;OFFSET ADDIEREN
LD        (SECTOR),A    ;SECTOR MERKEN
LD        D,L           ;D:=TRACK
LD        C,A           ;C:=SECTOR
LD        A,(#A708)     ;DRIVE FUER OPENIN
LD        E,A
```

```
LD        HL,(#A751)   ;POINTER FUER INPUT-BUFFER
DEFB      #DF
DEFW      TAB1         ;CALL #C666, LADE SECTOR
RET
;
;SICHERN DES BLOCKS
;
SAVE:LD   A,(REAWRI)   ;READ/WRITE FLAG
OR        A
RET       Z            ;SCHREIBEN NICHT NOETIG
LD        HL,(TRACK)   ;TRACK/SECTOR HOLEN
LD        D,L          ;D:=TRACK
LD        C,H          ;C:=SECTOR
LD        A,(#A708)    ;DRIVE BEI OPENIN
LD        E,A          ;E:=DRIVE
LD        HL,(#A751)   ;INPUT-BUFFER
DEFB      #DF
DEFW      TAB2         ;CALL #C64E SECTOR SCHREIBEN
XOR       A            ;AKKU:=0
LD        (REAWRI),A   ;FLAG RUECKSETZEN
RET
;
GET:LD    C,0          ;0=GET
JR        GETPUT
PUT:LD    C,1          ;1=PUT
GETPUT:OR A            ;TESTE ANZAHL PARAMETER
RET       Z            ;=0 => DANN SCHLUSS
LD        B,A          ;ANZAHL DER VARIABLEN (STRINGS!)
ADD       A,A          ;ACCU*2
PUSH      IX           ;IX NACH
POP       HL           ;HL
ADD       A,L
LD        L,A
LD        A,H
ADC       A,0          ;HL + ANZAHL*2-1 = POINTER IN PARAMETER
LD        H,A
DEC       HL
LD        A,C          ;ORDER#
LD        (ORDER),A    ;MERKEN
LO:PUSH   HL           ;MERKEN
```

```
LD        HL,(#A751)   ;POINTER FUER OPENIN
LD        DE,(OFFSET)
ADD       HL,DE        ;OFFSET ADDIEREN
LD        DE,(POINTER)
ADD       HL,DE        ;POINTER ADDIEREN
EX        DE,HL        ;DE:=POINTER IN BUFFER
POP       HL           ;ADRESSE ZURUECKHOLEN
PUSH      BC
LD        B,(HL)
DEC       HL
LD        C,(HL)       ;ADRESSE VARIABLE IN BC
DEC       HL
PUSH      HL
PUSH      DE
PUSH      BC
LD        A,(ORDER)
OR        A
JR        NZ,PUTV      ;VARIABLEN PUTen
LD        B,0          ;ZAEHLER AUF 0
L1:LD     A,(DE)       ;HOLE ZEICHEN
CP        13           ;CR=ENDE?
JR        Z,END
INC       B            ;LAENGE ERHOEHEN
JR        Z,END        ;FEHLER -- ZU LANG --> NULLSTRING !
INC       DE
JR        L1
END:POP   HL
LD        (HL),B       ;LAENGE MERKEN
POP       DE           ;DE:=STARTADRESSE
INC       HL
LD        (HL),E       ;LOW BYTE
INC       HL
LD        (HL),D       ;HIGH BYTE
LD        HL,(POINTER)
LD        D,0
LD        E,B          ;E:=LAENGE
INC       DE           ;+1 FUER CR
ADD       HL,DE
LD        (POINTER),HL  ;NEUER POINTER
LOOP:POP  HL           ;ZEIGER AUF PARAMETERTABELLE
```

```
POP       BC
DJNZ      L0            ;NAECHSTE VARIABLE
RET
;
PUTV:POP  HL            ;HOLE ADRESSE VARIABLE
LD        C,(HL)        ;ANZAHL ZEICHEN
LD        B,0
INC       HL
LD        E,(HL)        ;LOW BYTE ADRESSE
INC       HL
LD        D,(HL)        ;UND HIGH BYTE
EX        DE,HL         ;HL:=STRING
POP       DE            ;HOLE BUFFERADRESSE
LD        A,C
OR        A
PUSH      BC            ;MERKE ANZAHL
JR        Z,EN1         ;ENDE BEI NULLSTRING
LDIR                    ;VERSCHIEBE ZEICHEN IN BUFFER
EN1:EX    DE,HL
LD        (HL),13       ;CR=TRENNSYMBOL
POP       DE
INC       DE            ;+1 FUER TRENNZEICHEN
LD        HL,(POINTER)
ADD       HL,DE
LD        (POINTER),HL   ;NEUER POINTER
LD        A,1
LD        (REAWRI),A    ;MERKE, DASS GESCHRIEBEN
JR        LOOP          ;SCHLIESSE SCHLEIFE
;
;INITIALISIERT FILE
;
INIT:CP   01            ;1 PARAMETER ?
RET       NZ            ;FALSCHE PARAMETERANZAHL
LD        H,(IX+1)
LD        L,(IX)        ;LAENGE BESTIMMEN
LD        A,H
OR        L
JP        Z,SAVE        ;INIT,0 KOMMANDO
LD        (RECLEN),HL   ;MERKEN
LD        HL,#FFFF
```

```
LD        (OLDSEC),HL   ;FLAG LOESCHEN
LD        DE,(#A79B)    ;ADRESSE BUFFER OPENOUT
L10:LD    HL,#A719      ;BLOCKTABELLE
LD        B,16
L11:LD    A,(HL)
OR        A
RET       Z             ;LETZTER BLOCK
LD        (DE),A
INC       DE
INC       HL
DJNZ      L11           ;NAECHSTER BLOCK
LD        HL,(#A729)
LD        BC,#80
ADD       HL,BC
LD        (#A729),HL
LD        HL,0000
LD        (#A768),HL    ;BUFFER LEER MACHEN
DEFB      #DF
DEFW      GETCHAR       ;CALL #CF64 DISC IN CHAR
JR        L10
;
RECLEN:DEFW  64         ;RECORDLAENGE 1-512
SECNR:DEFW  0           ;SECTORNR
OFFSET:DEFW  0          ;OFFSET IM SECTOR
REAWRI:DEFB  0          ;FLAG 1=GESCHRIEBEN,0=GELESEN
OLDSEC:DEFW  #FFFF      ;LETZTER SECTOR
TRACK:DEFB  0           ;TRACK DES SECTORS
SECTOR:DEFB  0          ;SECTOR DES SECTORS
TAB1:DEFB  #66,#C6,7    ;ADRESSE #C666 IN FLOPPY-ROM
TAB2:DEFB  #4E,#C6,7    ;SECTOR SCHREIBEN
POINTER:DEFW  0         ;POINTER IN BUFFER
ORDER:DEFB  0
GETCHAR:DEFB  #64,#CF,7 ;CF64 GET CHAR AUS OPENIN BUFFER
FL0:DEFB  0
FL1:DEFB  0
ANAUS:DEFB  0
;
MULTI:XOR  A            ;AKKU LOESCHEN
LD        D,A           ;DE IST ERGEBNISREGISTER
LD        E,A
```

```
LD      (FL0),A
LD      (FL1),A
LD      A,16          ;ZAEHLER
Q1:RR   B             ;SHIFTEN VON BC
RR      C
JR      NC,Q2
PUSH    HL
ADD     HL,DE
JR      NC,Q0         ;KEIN UEBERTRAG
LD      (FL0),A       ;CARRY MERKEN
Q0:EX   DE,HL
PUSH    AF
LD      A,(FL0)
LD      HL,FL1
OR      (HL)
LD      (FL0),A       ;CARRY MERKEN
POP     AF
POP     HL
Q2:ADD  HL,HL
JR      NC,Q3
LD      (FL1),A
Q3:DEC  A             ;ZAEHLER
JR      NZ,Q1
RET
;
;
;ROUTINE ZUR FEHLERUNTERDRUECKUNG
;*********************************
;
;(C) 1985 BY DATA BECKER GMBH
;
; JS 12/2/1986
;
;
;
SET:LD  A,(ANAUS)
OR      A             ;VERHINDERE ZWEIFACHES EINSCHALTEN
RET     NZ
LD      A,(VER)
LD      (ANAUS),A
```

```
CP      #71 ;464?
JR      NZ,NO1
LD      A,#C3
LD      (#AC01),A     ;READY
NO1:LD  A,#C3
LD      (#BB5A),A     ;OUT CHAR
LD      (#BB06),A     ;KM WAIT CHAR
LD      (#BC9B),A     ;CAS CATALOG
;
LD      A,(VER)       ;VERSION
CP      #71
JR      NZ,NO3        ;664
LD      HL,READYMODE
LD      (#AC02),HL    ;READY PATCHEN
NO3:LD  HL,(#BB5B)
LD      (VEK4+1),HL
LD      HL,OUTCHK     ;OUT CHAR PATCHEN
LD      (#BB5B),HL    ;OUT CHAR PATCHEN
LD      HL,(#BB07)    ;ALTER VEKTOR MERKEN
LD      (VEK5+1),HL
LD      HL,KWC
LD      (#BB07),HL    ;KM WAIT CHAR PATCHEN
LD      HL,(#BC9C)    ;ALTER VEKTOR
LD      (VEK3),HL
LD      HL,CAT
LD      (#BC9C),HL    ;CAS CATALOG
LD      HL,BUFFER
LD      (HELP),HL
RET
;
;
;ROUTINE TESTET, OB ZEICHEN VON FLOPPY KOMMT
;
OUTCHK:EX  (SP),HL    ;HOLE RUECKSPRUNGADRESSE
PUSH    AF            ;RETTE AKKU
LD      A,H           ;TESTE HIGHER BYTE
CP      #CB           ;KOMMT AUS FLOPPY-ROM ?
JR      NZ,NEIN       ;NEIN
LD      A,1
LD      (TESTERR),A   ;FLAG SETZEN
```

```
POP       AF            ;ZEICHEN NICHT AUSGEBEN
PUSH      HL            ;RETTE HL
LD        HL,(HELP)     ;HOLE BUFFERPOINTER
CP        10            ;IST ZEICHEN LF
JR        Z,NOT
;CR WIRD ALS TRENNSYMBOL ZUGELASSEN
LD        (HL),A        ;SPEICHER ZEICHEN
INC       HL
LD        A,L
CP        40+BUFFER&#FF    ;40 ZEICHEN?
JR        NZ,N1
DEC       HL            ;40 ZEICHEN IST OBERE GRENZE
N1:LD     (HELP),HL
NOT:POP   HL
PUSH      AF
SUPRESS:POP  AF
EX        (SP),HL       ;RUECKSPRUNGADRESSE
RET                     ;KEINE AUSGABE
;
;
NEIN:LD   A,(VER)
CP        #71
JR        Z,NEINALT
LD        A,(TESTERR)
OR        A
JR        Z,N12
DEC       A
N12:LD    (TESTERR),A
JR        NZ,SUPRESS
PUSH      HL
LD        HL,BUFFER
LD        (HELP),HL
POP       HL
POP       AF            ;HOLE ZEICHEN
EX        (SP),HL
VEK4:DEFB  #CF,0,#94 ;RST 2
;
NEINALT:LD  A,(TESTERR)
OR        A
JR        NZ,SUPRESS
```

```
POP       AF            ;HOLE ZEICHEN
EX        (SP),HL
DEFB      #CF,0,#94 ;RST 2
;
;
;ROUTINE FUER READY MODUS
;
READYMODE:LD  A,(TESTERR)
OR        A             ;SETZE FLAGS
RET       Z             ;KEIN FEHLER
XOR       A
LD        (TESTERR),A
LD        HL,BUFFER
LD        (HELP),HL
LD        E,18
JP        #CA94
;
;PATCH FUER #BB06 KM WAIT CHAR
;DEFAULT-WERT FUER HARDWAREFEHLER
;IST C=CANCEL
;
KWC:PUSH  AF            ;RETTE AKKU
LD        A,(TESTERR)
OR        A             ;SETZE FLAGS
JR        NZ,DEFAULT
POP       AF            ;KEIN DEFAULT-C
VEK5:DEFB #CF,#3C,#9A
DEFAUL:LD A,4
LD        (TESTERR),A
POP       AF
LD        A,"C"         ;DEFAULT
RET
;
;
;
;
;
;ROUTINE UM FEHLERTEXT ZURUECKZUGEBEN
;
GETTXT:CP 01            ;1 PARAMETER ?
```

```
RET       NZ          ;FALSCHE PARAMETERANZAHL
LD        E,(IX)      ;LOW BYTE
LD        D,(IX+1)    ;HIGH BYTE
LD        C,#FF       ;ZAEHLER
LD        HL,BUFFER
LA:LD     A,(HL)
CP        13          ;CARRIAGE RETURN?
INC       HL
JR        Z,LA
DEC       HL          ;ERSTES TEXTZEICHEN GEFUNDEN
PUSH      HL          ;MERKEN
DEC       HL          ;MINUS EINS
SLOOP:INC C
INC       HL
LD        A,(HL)
CP        13          ;CARRIAGE RETURN?
JR        NZ,SLOOP
LD        A,C
LD        (DE),A      ;LAENGE MERKEN
POP       HL          ;TEXTADRESSE
EX        DE,HL
INC       HL
LD        (HL),E      ;LOW BYTE
INC       HL
LD        (HL),D      ;HIGH BYTE
RET
;
;
;RUECKSETZEN DER POINTER
;
RESET:LD  A,(VER)     ;VERSIONSNUMMER
CP        #71
JR        NZ,NO2
LD        A,#C9
LD        (#AC01),A
NO2:LD    A,#CF       ;RST 2
LD        (#BB5A),A   ;OUT CHAR
LD        (#BB06),A   ;KM WAIT CHAR
LD        A,#DF       ;RST 38H
LD        (#BC9B),A   ;CAS CATALOG
```

```
;
LD        HL,(VEK4+1)   ;OUT CHAR
LD        (#BB5B),HL
LD        HL,(VEK5+1)   ;KM WAIT CHAR
LD        (#BB07),HL
LD        HL,#A88B
LD        (#BC9C),HL    ;CAS CATALOG
XOR       A
LD        (ANAUS),A
RET
;
CAT:PUSH  AF            ;RETTE REGISTER
PUSH      HL
CALL      RESET         ;ORIGINALWERTE ZURUECK
POP       HL
POP       AF
CALL      #BC9B         ;CAS CATALOG
PUSH      HL
PUSH      AF
CALL      SET           ;UND WIEDER EINSCHALTEN
POP       AF
POP       HL            ;REGISTER ZURUECK
RET
;
;POINTER UND HILFSSPEICHER
;*************************
;
VEK3:DEFW  0            ;OUTCHAR VEKTOR
TESTERR:DEFB  #0
VER:DEFB  0
HELP:DEFW  BUFFER
TABU2:DEFB  #B0,#DD,#FD
TABU4:DEFB  #AE,#DD,#FD
BUFFER:DEFM  "OK"
DEFB      #0D ;CR
DEFS      40-3          ;40 ZEICHEN BUFFER
```

Der BASIC-Lader für die RSX-Kommandos:

```
10 '*******************************************
20 '*** BASIC - Loader fuer RSX-Kommandos ******
30 '*** JS 12/2/1986 (c) by DATA BECKER GmbH ***
40 '*******************************************
50 '
60 DEFINT b-z : DEFREAL s,i
70 adresse = &6000 'Startadresse der Routine
80 '
90 DATA 3A,3F,83,B7,C0,CD,00,B9,F5,3A,01,DE,32,3F,83,F1
100 DATA CD,0C,B9,3A,3F,83,FE,71,28,1D,3E,DF,32,87,80,32
110 DATA CA,80,3A,3F,83,FE,C9,20,05,21,45,83,18,03,21,42
120 DATA 83,22,88,80,22,CB,80,01,40,80,21,70,80,C3,D1,BC
130 DATA 57,80,C3,74,80,C3,FF,80,C3,03,81,C3,7D,81,C3,04
140 DATA 82,C3,FA,82,C3,D3,82,52,45,43,4F,52,C4,47,45,D4
150 DATA 50,55,D4,49,4E,49,D4,41,CE,41,55,D3,45,52,D2,00
160 DATA 00,00,00,00,FE,01,C0,DD,46,01,DD,4E,00,2A,BA,81
170 DATA CD,D4,81,EB,11,00,02,CD,C1,BD,3A,D1,81,ED,53,BE
180 DATA 81,11,80,00,B7,28,01,19,22,BC,81,AF,32,CB,81,32
190 DATA CC,81,ED,5B,C1,81,ED,52,C8,CD,E6,80,2A,BC,81,22
200 DATA C1,81,CB,1C,CB,1D,F5,ED,5B,9B,A7,19,7E,6F,26,00
210 DATA 29,F1,11,00,00,ED,5A,11,09,00,CD,C1,BD,23,23,7D
220 DATA 32,C3,81,7B,C6,41,32,C4,81,55,4F,3A,08,A7,5F,2A
230 DATA 51,A7,DF,C5,81,C9,3A,C0,81,B7,C8,2A,C3,81,55,4C
240 DATA 3A,08,A7,5F,2A,51,A7,DF,C8,81,AF,32,C0,81,C9,0E
250 DATA 00,18,02,0E,01,B7,C8,47,87,DD,E5,E1,85,6F,7C,CE
260 DATA 00,67,2B,79,32,CD,81,E5,2A,51,A7,ED,5B,BE,81,19
270 DATA ED,5B,CB,81,19,EB,E1,C5,46,2B,4E,2B,E5,D5,C5,3A
280 DATA CD,81,B7,20,24,06,00,1A,FE,0D,28,06,04,28,03,13
290 DATA 18,F5,E1,70,D1,23,73,23,72,2A,CB,81,16,00,58,13
300 DATA 19,22,CB,81,E1,C1,10,BF,C9,E1,4E,06,00,23,5E,23
310 DATA 56,EB,D1,79,B7,C5,28,02,ED,B0,EB,36,0D,D1,13,2A
320 DATA CB,81,19,22,CB,81,3E,01,32,C0,81,18,D7,FE,01,C0
330 DATA DD,66,01,DD,6E,00,7C,B5,CA,E6,80,22,BA,81,21,FF
340 DATA FF,22,C1,81,ED,5B,9B,A7,21,19,A7,06,10,7E,B7,C8
350 DATA 12,13,23,10,F8,2A,29,A7,01,80,00,09,22,29,A7,21
360 DATA 00,00,22,68,A7,DF,CE,81,18,DE,40,00,00,00,00,00
370 DATA 00,FF,FF,00,00,66,C6,07,4E,C6,07,00,00,00,64,CF
```

```
380 DATA 07,00,00,00,AF,57,5F,32,D1,81,32,D2,81,3E,10,CB
390 DATA 18,CB,19,30,15,E5,19,30,03,32,D1,81,EB,F5,3A,D1
400 DATA 81,21,D2,81,B6,32,D1,81,F1,E1,29,30,03,32,D2,81
410 DATA 3D,20,DC,C9,3A,D3,81,B7,C0,3A,3F,83,32,D3,81,FE
420 DATA 71,20,05,3E,C3,32,01,AC,3E,C3,32,5A,BB,32,06,BB
430 DATA 32,9B,BC,3A,3F,83,FE,71,20,06,21,AB,82,22,02,AC
440 DATA 2A,5B,BB,22,9E,82,21,5B,82,22,5B,BB,2A,07,BB,22
450 DATA C8,82,21,BF,82,22,07,BB,2A,9C,BC,22,3C,83,21,2A
460 DATA 83,22,9C,BC,21,48,83,22,40,83,C9,E3,F5,7C,FE,CB
470 DATA 20,1E,3E,01,32,3E,83,F1,E5,2A,40,83,FE,0A,28,0B
480 DATA 77,23,7D,FE,70,20,01,2B,22,40,83,E1,F5,F1,E3,C9
490 DATA 3A,3F,83,FE,71,28,19,3A,3E,83,B7,28,01,3D,32,3E
500 DATA 83,20,EA,E5,21,48,83,22,40,83,E1,F1,E3,CF,00,94
510 DATA 3A,3E,83,B7,20,D7,F1,E3,CF,00,94,3A,3E,83,B7,C8
520 DATA AF,32,3E,83,21,48,83,22,40,83,1E,12,C3,94,CA,F5
530 DATA 3A,3E,83,B7,20,04,F1,CF,3C,9A,3E,04,32,3E,83,F1
540 DATA 3E,43,C9,FE,01,C0,DD,5E,00,DD,56,01,0E,FF,21,48
550 DATA 83,7E,FE,0D,23,28,FA,2B,E5,2B,0C,23,7E,FE,0D,20
560 DATA F9,79,12,E1,EB,23,73,23,72,C9,3A,3F,83,FE,71,20
570 DATA 05,3E,C9,32,01,AC,3E,CF,32,5A,BB,32,06,BB,3E,DF
580 DATA 32,9B,BC,2A,9E,82,22,5B,BB,2A,C8,82,22,07,BB,21
590 DATA 8B,A8,22,9C,BC,AF,32,D3,81,C9,F5,E5,CD,FA,82,E1
600 DATA F1,CD,9B,BC,E5,F5,CD,04,82,F1,E1,C9,00,00,00,00
610 DATA 48,83,B0,DD,FD,AE,DD,FD,4F,4B,0D
620 :
630 FOR i=0 TO &34A
640   READ d$
650   POKE i+adresse, VAL("&"+D$)
660   s=s+VAL("&"+D$)
670 NEXT
680 IF s<>95703 THEN PRINT CHR$(7)"*** Fehler in Datas !! ***" : END
690 :
700 'Anpassen an Speicherbereich
710 :
720 FOR i=adresse TO adresse+&34A
730   p=PEEK(i)
740   IF p>&7F AND p<&84 THEN 800
750 NEXT
760 PRINT CHR$(7) : INPUT"Name des Files :",a$
770 SAVE a$,b,adresse,&34B
```

```
780 END
790 :
800 IF PEEK(I+1)>&7F AND PEEK(I+1)<&84 OR PEEK(i-1)=1 OR PEEK(i-1)=17
THEN 750
810 ad=PEEK(i-1) + p*256 : ad=UNT(ad) - &8000 + adresse
820 POKE i-1,&FF AND UNT(ad)   'lower Byte
830 POKE i,&FF AND INT(ad/256) 'Higer Byte
840 GOTO 750
```

5.4 Das Dateiverwaltungsprogramm

Haben Sie ein Chaos in Ihrer Plattensammlung oder wissen Sie nicht mehr, in welchem Ordner sich Ihre wertvolle Briefmarke befindet? Dann ist dieses Dateiverwaltungsprogramm Ihnen wie auf den Leib geschrieben.

Das abgedruckte Dateiverwaltungsprogramm ist zwar relativ lang, trotzdem sollten Sie sich nicht davon abhalten lassen, es abzutippen *(oder die entsprechende Diskette zum Buch zu erwerben)*, da es Ihnen ungeahnte Möglichkeiten eröffnet. Es ist zwar nicht so komfortabel wie etwa DATAMAT oder ähnliche bekannte Dateiverwaltungsprogramme, aber es reicht für viele Anwendungen von der Geschwindigkeit und der Flexibilität her vollkommen aus.

Ob Sie nun Ihre Adressen verwalten wollen oder Ihre Schallplatten, ist ganz egal, weil keine feste Eingabemaske vorgesehen ist. Sie haben bis zu 10 Datenfelder für Ihre Datensätze zur Verfügung, die Sie beliebig belegen können. Das Programm ist so ausgelegt, daß Sie bis zu 200 Datensätze verwalten können. Die Daten werden nicht relativ sondern sequentiell abgespeichert, alle Daten in den Speicher geholt. Dies wurde nicht zuletzt auch aus Rücksicht auf den Leser/Anwender gemacht, da das Programm ansonsten noch etwas länger geworden wäre.

In dieses Programm wurden die deutschen Umlaute mit eingebaut, jedoch werden diese beim Sortiervorgang nicht korrekt mitsortiert(!), da sonst der entsprechende Aufwand in keinem Verhältnis zur Wirkung gestanden hätte. Die Tasten auf der Tastatur werden entsprechend der DIN-Tastatur umbelegt. Die Tasten Z und Y werden vertauscht.

Wünschen Sie die amerikanische Tastatur, so können Sie dies erreichen, indem Sie die Zeilen 350-440 löschen. Außerdem ist es erforderlich, daß Sie die Diskette, auf der Sie das Dateiverwaltungsprogramm speichern, auch die Fehlerabfangroutine bzw. das Erweiterungspaket aus Kapitel 5.3 befindet (unter dem entsprechenden Dateinamen). Generieren Sie gemäß Kapitel 5.3

an Adresse &9F00 und speichern Sie es unter dem Namen "RSX.BIN" auf Diskette ab. Das ist schon so ziemlich alles, was Sie bedenken müssen.

Nachdem Sie das Programm abgetippt und mindestens einmal abgespeichert haben (besser noch mehrmals auf verschiedenen Disketten) und die Datei RSX.BIN existiert, können Sie das Programm starten. Sie erhalten zunächst das sogenannte *MENÜ* auf dem Bildschirm. Mit den Cursor-Tasten können Sie nun die Menüpunkte auswählen, die dann mit ENTER quittiert werden müssen; es wird dann der angewählte Menüpunkt ausgeführt.

Die Daten des Programmes werden unter User-Nummer 1 abgelegt, so daß die Daten klar vom Programm und den restlichen Daten getrennt werden. Um Ihre Dateien zu sehen, können Sie den vorhandenen Menüpunkt

Disketteninhalt zeigen

anwählen, es werden Ihnen dann nur die Dateien aus dem User-Bereich 1 auf dem Bildschirm angezeigt; im Normalfall also Ihre Dateien. Führen Sie dies zum ersten Mal durch, so werden Sie sehen, daß lediglich der verfügbare Speicherplatz auf Ihrer Diskette auf dem Bildschirm angezeigt wird. Durch betätigen einer beliebigen Taste gelangen Sie wieder ins Hauptmenü zurück.

Lassen Sie uns nun einmal eine Datei errichten, die wir auch schon in Kapitel 1 sowie Kapitel 5.3 angesprochen haben. Wir erstellen eine *Telefondatei* mit folgenden drei Datenfeldern:

Name
Vorname
Telefon

Wir können diese Datei leicht noch um drei weitere Datenfelder erweitern, indem wir beispielsweise die Datenfelder

PLZ
Ort
Straße

hinzufügen. Wählen Sie den Menüpunkt

Maske erstellen/ändern

an, wir erstellen uns dann unsere Maske mit den sechs Datenfeldern. Eingaben werden wie gehabt mit ENTER quittiert. Wenn Sie ein Datenfeld ändern wollen, so können Sie mit der *Cursor-Hoch-Taste* den Cursor hochbewegen und das entsprechende Feld überschreiben. Die Korrekturen einzelner Buchstaben ist hier aus betriebsinternen Gründen leider nicht möglich, überschreiben Sie also das entsprechende Feld bitte völlig. Der Cursor wird nach der Eingabe immer in das Feld springen, das dem letzten Feld unmittelbar folgt. Wenn Sie den Menüpunkt

Maske erstellen/ändern

verlassen wollen, so drücken Sie die CTRL- und die ENTER-Taste *gleichzeitig*. Augenblicklich befinden Sie sich wieder im Hauptmenü. Sobald Sie einen Datensatz eingegeben haben, können Sie die Maske lediglich korrigieren, aber keine weiteren Datenfelder hinzufügen.

Nachdem Sie die Maske erstellt haben, sollten Sie den Menüpunkt *Daten eingeben/ändern* anwählen. Sie erhalten nin im 80-Zeichen-Modus die Eingabemaske. Es wird Ihenn rechts oben im Bildschirm die maximale Länge Ihrer Datenfelder angegeben, damit eine Zeile nicht überschrieben wird.

Grundsätzlich wird immer das Feld invertiert angezeigt, das momentan beschrieben wird. Geben Sie zunächst einmal Ihre eigenen Daten ein. Wenn Sie fertig sind, wird die Eingabemaske wieder geleert und links unten sehen Sie nun, daß ein Datensatz gespeichert worden ist. Vielleicht haben Sie schon verwundert rechts unten im Bildschirm eine Statusanzeige entdeckt. Sie schließen nun messerscharf, daß nicht sein kann, was nicht sein darf - also muß es noch einen weiteren Status geben. Es handelt sich hierbei um den Korrekturstatus. Im mittleren Fenster sehen Sie, daß das gleichzeitige Betätigen von <CTRL> und <ENTER>-Taste zu einem Wechsel des Status führt. Versuchen Sie es einmal:

Schon lesen Sie in der rechten unteren Ecke unter "Status" nicht mehr das Wort *Eingabe*, sondern das Wort *Korrektur*. Weiterhin erscheint

** *Füllen Sie die Maske mit Suchbegriff* **

Dies hat folgende Bewandnis: Wenn Sie einen Datensatz ändern wollen, dann müssen Sie dem Programm mitteilen, um welchen Datensatz es sich hierbei handelt. Hierzu steht Ihnen die gesamte Eingabemaske zur Verfügung - aber keine Angst, Sie brauchen nicht die gesamte Eingabemaske mit Daten zu füllen. Geben Sie beispielsweise einmal in dem Feld *Vorname* Ihren Vornamen ein, die anderen Felder quittieren Sie einfach mit ENTER. Sobald Sie am unteren Ende der Eingabemaske angekommen sind, werden alle verbleibenden Felder mit Ihren Daten gefüllt, die Sie jetzt korrigieren können. Hier gilt dasselbe, wie bei der Maske: Ein Feld, das Sie korrigieren wollen, müssen Sie gesamt überschreiben; ein Feld, das Sie nicht korrigieren wollen, quittieren Sie einfach mit ENTER. Sobald Sie alle Felder mit ENTER abgeschlossen haben, befinden Sie sich automatisch wieder im Eingabemodus. Doch sollten Sie beispielsweise in Ihrer Datei

mehr als eine Person mit Ihren Vornamen haben, so kann es sein, daß ein andere Datensatz zur Korrektur angezeigt wird, da ausschließlich der erste gefundene Datensatz, der auf Ihre Beschreibung paßt, berücksichtigt wird.

Stellen Sie sich vor, unter Ihren Freunden und Bekannten seien zwei mit dem Vornamen PETER. Wollen Sie nun sicher gehen, daß der richtige PETER angezeigt wird, so müssen Sie dem Programm mehr Informationen an die Hand geben, als nur den Vornamen, beispielsweise könnten Sie die Telefonnummer oder den Nachnamen noch zusätzlich eingeben.

Geben Sie mehr als ein Datenfeld ein, so wird ein sogenannter *UND-Vergleich* durchgeführt, d.h. alle Angaben müssen zutreffen. Sollten Sie einen Datensatz suchen lassen, der garnicht existiert, so wird Ihnen die Fehlermeldung

** *Datensatz gibt es nicht* **

ausgegeben. Sie befinden sich dann immer noch im Korrekturmodus, und könnten einen weiteren Versuch starten.

Durch CTRL/ENTER kommen Sie jederzeit aus dem Korrekturmodus heraus, Sie kehren dann wieder in den Eingabemodus zurück (die Korrektur wird nicht vorgenommen!)

Aus dem Menü *Eingabe/Korrektur* kommen Sie jederzeit heraus, indem Sie in einem beliebigen Datenfeld das Wort *ENDE* eingeben.

Nun noch kurz zu den weiteren Menüpunkten:

Daten laden und sichern

Es werden die eingegebenen Daten und die Maske geladen/gesichert. Sie können einen Filenamen angeben, der allerdings nicht länger als 8 Zeichen sein darf. Durch CTRL/ENTER kommen Sie wieder ins Hauptmenü zurück.

Daten löschen

Sie können einen Datensatz löschen. Dazu können Sie ihn erst, wie im Programmteil Erstellen/Ändern suchen lassen; bevor er gelöscht wird, müssen Sie noch eine Sicherheitsabfrage bestätigen.

Daten sortieren

Sie können Ihre Daten nach einem beliebigen Datenfeld sortieren lassen, auch hier müssen Sie das Feld zunöchst quittieren.

Daten ausgeben

Sie können auswählen, ob die Ausgabe auf dem Drucker (D) oder auf dem Bildschirm (B) erfolgen soll. Nachdem Sie entweder die Taste "B" oder "D" betätigt haben, können Sie auswählen, welche Datenfelder ausgegeben werden sollen; interessant für Listen, die nicht alle Datenfelder beinhalten. Wenn ein Datenfeld nicht ausgegeben werden soll, so betätigen Sie in dem entsprechendem Feld lediglich die ENTER-Taste, ansonsten füllen Sie das Feld mit irgendeinem Zeichen, es reicht also, wenn Sie in einem Feld Leertaste/ENETER drücken, um es ausgeben zu lassen.

Diskettenінahlt zeigen

Es wird Ihnen lediglich das Inhaltsverzeichnis einer auf dem Bildschirm angezeigt.

Programm beenden

Wählen Sie diesen Programmpunkt erst dann an, wenn Sie schon Ihre Daten und evtl. deren Korrekturen auf Diskette abgespeichert haben; es erfolgt keine Sicherheitsabfrage im Programm.

Alles löschen

Es werden alle Daten im Speicher gelöscht, das Programm wird neu gestartet.

Das Dateiverwaltungsprogramm:

```
10 '************************************************
20 '***** (c) 1986 by DATA BECKER *******************
30 '***** Autor : Joerg Schieb ************************
40 '***** Dateiverwaltung ****************************
50 '************************************************
60 '
70 ON ERROR GOTO 3170
80 SYMBOL AFTER 91
90 MEMORY &9E50-1
100 LOAD "0:RSX.BIN"
110 CLEAR
120 DEFINT A-Z : ds$="" : CALL &9E50 : |AN
130 ON ERROR GOTO 3170
140 ON BREAK GOSUB 960
150 OPENOUT "dumm"
160  MEMORY HIMEM-1
170 CLOSEOUT
180 DIM ma$(20),m$(9),l(20),d$(200,9),da$(9)
190 cursorup$=CHR$(240):cursordown$=CHR$(241)
200 |USER,1
210 :
220 REM ======================================
230 REM     Definition der deutschen Umlaute
240 REM ======================================
250 :
260 DATA 135,91,136,92,137,93,132,123,133,124,134,125,129,126
270 FOR i=1 TO 7:READ a,b:KEY a,CHR$(b):NEXT
280 SYMBOL 123,198,0,120,12,124,204,118
290 SYMBOL 124,102,0,60,102,102,102,60
300 SYMBOL 92,102,60,102,102,102,102,60
310 SYMBOL 91,102,60,102,102,126,102,102
320 SYMBOL 125,102,0,102,102,102,102,62
330 SYMBOL 93,36,102,102,102,102,102,60
340 SYMBOL 126,28,35,99,108,99,99,110,96
350 KEY DEF 29,1,124,92
360 KEY DEF 28,1,123,91
370 KEY DEF 26,1,125,93,124
```

```
380 KEY DEF 24,1,126,ASC("ú")
390 KEY DEF 19,1,58,42
400 KEY DEF 17,1,59,43,64
410 KEY DEF 18,0,13,13,140
420 KEY 140,"*ESC*"+CHR$(13)
430 KEY DEF 71,1,ASC("y"),ASC("Y")
440 KEY DEF 43,1,ASC("z"),ASC("Z")
450 :
460 DATA Daten laden,Daten  sichern,Daten  eingeben/{ndern,"Daten
l|schen",Daten sortieren,Daten  geben,Disketteninhalt zeigen,Maske
erstellen/{ndern,Programm  beenden,Alles l|chen,ende
470 :
480 i=0 : WHILE ma$(i)<>"ende" : i=i+1
490 READ ma$(i) : l(i)=LEN(ma$(i))/2
500 WEND : anzmask = i-1
510 STAT=0:INK 0,0 : INK 1,13 : BORDER 0 : PAPER 0 : MODE 1 : PEN 1 :
PEN #1,0 : PAPER #1,1
520 WINDOW #1,1,40,1,3:CLS #1:LOCATE #1,16,2:PRINT #1,"M E N ]"
530 FOR i=1 TO anzmask
540  LOCATE 20-l(i),5+i*2 : PRINT ma$(i)
550 NEXT
560 feld=1 : PEN #1,1
570 PAPER #1,1 : PEN #1,0
580 WINDOW #1,20-l(feld),20+l(feld),5+feld*2,5+feld*2 : PRINT
#1,ma$(feld)
590 d$=INKEY$:IF d$="" THEN 590
600 PAPER #1,0:PEN #1,1:IF d$=cursorup$ THEN PRINT
#1,ma$(feld):feld=feld-1:IF feld=0 THEN feld=anzmask:GOTO 570 ELSE 570
610 IF d$=cursordown$ THEN PRINT #1,ma$(feld):feld=feld+1:IF
feld>anzmask THEN feld=1:GOTO 570 ELSE 570
620 IF d$<>CHR$(13) THEN 590
630 ON feld GOSUB 1680,1240,2190,2690,2810,2940,1940,2020,1880,640:GOTO
510
640 RUN 120 'Loesche alles
650 :
660 REM ======================================
670 REM    Suche das Stringarray da$
680 REM ======================================
690 :
700 IF anzahl=0 THEN found=0 : RETURN
```

```
710 FOR i=1 TO anzahl
720   FOR i9=0 TO anzahlfelder-1
730     IF da$(i9)="" THEN 750
740     IF da$(i9)<>d$(i,i9) THEN 770
750   NEXT i9
760   found=i : RETURN
770 NEXT i
780 found=0 : RETURN
790 :
800 REM ====================================
810 REM    Sortiere Feld nach Feldnr. feld
820 REM ====================================
830 :
840 IF anzahl<2 THEN RETURN
850 EVERY 30,0 GOSUB 2870
860 FOR i1=1 TO anzahl-1
870   ver=i1
880     FOR i2=i1+1 TO anzahl
890       IF d$(i2,feld)<d$(ver,feld) THEN ver=i2
900     NEXT
910   FOR i2=0 TO anzahlfelder-1
920     d$=d$(ver,i2) : d$(ver,i2)=d$(i1,i2) : d$(i1,i2)=d$
930   NEXT
940 NEXT
950 i=REMAIN(0)
960 RETURN
970 :
980 REM ===================================
990 REM  Test eines Filenamens auf Existenz
1000 REM ===================================
1010 :
1020 nf=0 : ef=0 : OPENIN file$+".dat" : IF ef THEN 1020
1030 found=1-nf : '1=gefunden, 0=nicht gefunden
1040 CLOSEIN
1050 RETURN
1060 :
1070 REM ============================
1080 REM    Loesche String Nr. Sn
1090 REM ============================
1100 :
```

```
1110 IF anzahl=1 THEN anzahl=0 : RETURN
1120 FOR i=sn TO anzahl-1
1130   FOR i1=0 TO anzahlfelder-1
1140     d$(i,i1)=d$(i+1,i1)
1150   NEXT
1160 NEXT
1170 anzahl=anzahl-1
1180 RETURN
1190 :
1200 REM ============================
1210 REM   Sichere Datei auf Disk
1220 REM ============================
1230 :
1240 GOSUB 1490 : IF file$="*ESC*" THEN RETURN ELSE GOSUB 1020 : IF
found=0 THEN 1310 ELSE WINDOW #1,1,40,25,25 : PAPER 2 : INK 2,1 : CLS #1
: PRINT #1,CHR$(7)"Soll ersetzt werden (j,n) "
1250 EVERY 20,0 GOSUB 1290
1260 d$=INKEY$ : IF d$="" THEN 1260
1270 im=REMAIN(0):IF UPPER$(d$)<>"J" THEN 1240
1280 GOTO 1310
1290 LOCATE #1,30,1 : IF im=0 THEN PRINT#1,"?";:im=1 ELSE PRINT#1,"
";:im=0
1300 RETURN
1310 ef=0 : OPENOUT file$+".dat" : IF ef THEN 1310
1320  PRINT#9,anzahlfelder
1330 FOR i=0 TO anzahlfelder-1
1340  PRINT #9,m$(i)
1350 NEXT
1360 PRINT#9,anzahl
1370  FOR i=1 TO anzahl
1380   FOR i1=0 TO anzahlfelder-1
1390    PRINT#9,d$(i,i1)
1400   NEXT
1410  NEXT
1420 CLOSEOUT
1430 RETURN
1440 :
1450 REM ============================
1460 REM   Lies Filenamen ein
1470 REM ============================
```

```
1480 :
1490 MODE 1 : WINDOW #1,1,40,1,3 : PAPER #1,2 : INK 2,1 : CLS#1 : LOCATE
#1,7,2 : PRINT #1,"Geben Sie den Filenamen ein:"
1500 WINDOW #1,1,40,15,16 : CLS #1 : PRINT#1,TAB(10)"Verwenden Sie nicht
(,.:"CHR$(34)")
1510 PRINT#1,TAB(8)"Geben Sie maximal 8 Zeichen ein
1520 WINDOW #1,1,40,6,10 : CLS #1
1530 LOCATE #1,5,3 : INK 3,3
1540 PRINT #1,"Filename :";
1550 WINDOW #1,16,24,8,8 : PAPER #1,3 : CLS #1
1560 LINE INPUT #1,"",file$ : IF file$="*ESC*" THEN RETURN
1570 IF LEN(file$)>8 THEN PEN 3:LOCATE 8,16:PRINT CHR$(7)"Geben Sie
maximal 8 Zeichen ein":FOR i=1 TO 300:NEXT:PAPER #1,2:GOTO 1500
1580 c$=",.:"+CHR$(34)
1590 FOR i=1 TO LEN(c$):IF INSTR (file$,MID$(c$,i,1))=0 THEN NEXT:GOTO
1610
1600 LOCATE 10,15 : PEN 3:PRINT CHR$(7)"Verwenden Sie nicht
(,.:"CHR$(34)")":FOR i=1 TO 300:NEXT:PAPER #1,2:GOTO 1500
1610 RETURN
1620 :
1630 :
1640 REM ======================================
1650 REM    Liest Datei ein
1660 REM ======================================
1670 :
1680 GOSUB 1490 : IF file$="*ESC*" THEN RETURN ELSE GOSUB 1020 : IF
found=0 THEN 1680
1690 ef=0 : OPENIN file$+".dat" : IF ef THEN 1690
1700 INPUT#9,anzahlfelder
1710 ERASE d$:DIM d$(200,anzahlfelder-1)
1720 FOR i=0 TO anzahlfelder-1
1730  INPUT#9,m$(i)
1740 NEXT
1750 INPUT#9,anzahl
1760 FOR i=1 TO anzahl
1770   FOR i1=0 TO anzahlfelder-1
1780     INPUT#9,d$(i,i1)
1790   NEXT i1
1800 NEXT
1810 CLOSEIN
```

```
1820 RETURN
1830 :
1840 REM ==============================
1850 REM    Programm beenden
1860 REM ==============================
1870 :
1880 |USER,0 : CALL aus : MODE 2 : PRINT "*** Programm beendet ***" :
END
1890 :
1900 REM ==============================
1910 REM    Zeige Diskettenimhalt
1920 REM ==============================
1930 :
1940 MODE 2 : CAT
1950 LOCATE 36,25 : PRINT "<< Tastendruck >>"
1960 CALL &BB06 : RETURN
1970 :
1980 REM ==============================
1990 REM   Aendern/Erstellen Maske
2000 REM ==============================
2010 :
2020 MODE 1 : WINDOW #1,1,40,1,3 : PAPER #1,2 : INK 2,1 : CLS#1 : LOCATE
#1,7,2 : PRINT #1,"Erstellen und Ändern der Maske
2030 LOCATE 1,7 : IF anzahl>0 THEN bis=anzahlfelder-1 ELSE bis=9
2040 FOR i=0 TO bis
2050  PRINT RIGHT$(STR$(i+1),2)". Feld:"m$(i)
2060 NEXT
2070 WINDOW #1,5,35,23,25 : PAPER #1,2 : PEN #1,1 : CLS#1 : LOCATE
#1,8,2 : PRINT#1,"Ctrl/Enter führt Ende
2080 WINDOW #1,10,40,7,7+bis : PAPER #1,0 : PEN #1,1
2090 CLS#1:FOR I=0 TO ANZAHLFELDER-1:PRINT #1,M$(I)
2100 NEXT:LOCATE #1,1,anzahlfelder+1+(anzahlfelder=bis+1):i2=VPOS(#1)-
2:LINE INPUT #1,"",a$ : IF a$="*ESC*" THEN RETURN ELSE IF a$="" THEN
LOCATE #1,1,anzahlfelder+1 : GOTO 2090
2110 i1=VPOS(#1)-2:IF i2=8 AND i1=8 THEN i1=9
2120 m$(i1)=a$ : IF i1>anzahlfelder-1 THEN anzahlfelder=i1+1
2130 GOTO 2090
2140 :
2150 REM ====================================
2160 REM    Daten eingeben/Ändern
```

```
2170 REM ====================================
2180 :
2190 IF anzahlfelder=0 THEN RETURN ELSE MODE 2 : WINDOW #1,1,80,1,3 :
PAPER #1,1 : PEN #1,0 : CLS #1
2200 IF stat=0 THEN a$="Eingeben und [ndern von Daten" ELSE IF stat=3
THEN a$="Selektieren des zu streichenden Datensatz" ELSE IF stat=5 THEN
a$="Selektieren der auszugebenden Datens{tze"
2210 LOCATE #1,40-LEN(a$)/2,2:PRINT #1,a$
2220 GOSUB 2580
2230 LOCATE 1,7 : FOR i=0 TO anzahlfelder-1
2240   PRINT m$(i)TAB(laenge-2)":"
2250 NEXT
2260 LOCATE 60,5 : PRINT "Datenfeldl{nge="80-laenge
2270 PEN #2,0 : PEN #3,1 : PAPER #2,1 : PAPER #3,0
2280 WINDOW #2,5,75,22,23 : CLS #2 : IF stat<5 THEN PRINT #2,TAB(7)"**
Ctrl/Entertaste=Wechsel Status (Eingabe/Korrektur) **":IF stat=0 THEN
a$="** Geben Sie in beliebigen Feld 'ende' ein f}r Ende **":GOTO 2300
2290 IF stat<>2 THEN a$="** F}llen Sie die Maske mit Suchbegriff **"
ELSE a$="**   Geben Sie nun die Korrekturen ein   **"
2300 PRINT #2,TAB(35-LEN(a$)/2)a$
2310 WINDOW #1,1,80,25,25 : PAPER #1,1 : PEN #1,0
2320 CLS#1 : LOCATE #1,5,1 : PRINT #1,"Gespeichert :";anzahl : LOCATE
#1,55,1 : PRINT #1,"Status : "; : IF stat<>1 THEN PRINT#1,"** Eingabe
**" ELSE PRINT#1,"** Korrektur **"
2330 PO=0
2340 IF stat<>2 THEN WINDOW #3,laenge,79,7,16 : CLS#3
2350 WHILE PO<anzahlfelder : WINDOW#2,1,laenge-3,7+PO,7+po : WINDOW
#3,1,laenge-3,7+po,7+po : CLS#2 : PRINT #2,m$(po)
2360 WINDOW #4, laenge, 79, 7+po, 7+po : LINE INPUT #4, "", a$ :
a$=LEFT$(a$,80 - laenge)
2370 IF a$="*ESC*" THEN 2500 ELSE IF LOWER$(a$)="ende" THEN RETURN
2380 da$(po)=a$
2390 CLS#3:PRINT#3,m$(po):po=po+1
2400 WEND
2410 IF stat=0 THEN anzahl=anzahl+1+(anzahl=200):FOR i=0 TO
anzahlfelder-1:d$(anzahl,i)=da$(i):NEXT:GOTO 2320
2420 IF stat=2 THEN 2470
2430 IF stat=5 THEN RETURN
2440 GOSUB 700 : REM ermittle gesuchten string
```

```
2450 LOCATE 25,20 : IF found=0 THEN PRINT CHR$(7)"** Datensatz gibt es
nicht **" ELSE PRINT SPACE$(30)
2460 IF found=0 THEN 2320 ELSE FOR i=0 TO anzahlfelder-1:LOCATE
laenge,7+i:PRINT d$(found,i):NEXT:IF stat=3 THEN RETURN ELSE stat=2:GOTO
2280
2470 FOR i=0 TO anzahlfelder-1:IF da$(i)="" THEN 2490
2480 d$(found,i)=da$(i)
2490 NEXT:stat=1:GOTO 2500
2500 IF stat=5 THEN RETURN ELSE IF st>1 THEN 2280 ELSE stat=1-stat:GOTO
2280
2510 END
2520 RETURN
2530 :
2540 REM ================================
2550 REM    Ermittle groessten String
2560 REM ================================
2570 :
2580 laenge=0
2590 FOR i=0 TO anzahlfelder-1
2600   IF LEN(m$(i))>laenge THEN laenge=LEN(m$(i))
2610 NEXT
2620 laenge=laenge+4
2630 RETURN
2640 :
2650 REM ======================================
2660 REM     Datensatz streichen
2670 REM ======================== Ç=====  =======
2680 :
2690  IF anzahl=0 THEN RETURN ELSE stat=3 : st1=1 : GOSUB 2190 : IF
LOWER$(a$)="ende" THEN RETURN ELSE st1=0
2700 LOCATE 1,18 : PRINT "Datensatz ok (j/n) ":EVERY 30,0 GOSUB 2740
2710 d$=INKEY$ : IF d$="" THEN 2710
2720 sn=REMAIN(0):IF LOWER$(d$)<>"j" THEN 2690
2730 sn=found : GOSUB 1110 : RETURN
2740 i=ABS(i=0):LOCATE 20,18:IF i THEN PRINT"?" ELSE PRINT" "
2750 RETURN
2760 :
2770 REM ======================================
2780 REM     Sortieren
2790 REM ======================================
```

```
2800 :
2810 MODE 2:PRINT"Nach welchem Feld soll sortiert werden (1-
"STR$(anzahlfelder)"):"; : INPUT " ",a
2820 IF a<1 OR a>anzahlfelder THEN 2810
2830 PRINT : PRINT "Feld "CHR$(34);m$(a-1);CHR$(34)" -- ok ? (j/n)
2840 d$=INKEY$ : IF d$="" THEN 2840
2850 IF LOWER$(d$)<>"j" THEN 2810
2860 feld=a-1 : GOTO 840
2870 i9=(i9=0):LOCATE 5,7:IF i9 THEN PRINT" "CHR$(224) ELSE PRINT
CHR$(224)" "
2880 RETURN
2890 :
2900 REM ====================================
2910 REM    Ausgabe auf Bildschirm und Drucker
2920 REM ====================================
2930 :
2940 MODE 2:PRINT"Ausgabe auf Drucker oder Bildschirm ? (D/B)"
2950 d$=INKEY$ : IF d$="" THEN 2950
2960 IF INSTR("bd",LOWER$(d$))=0 THEN 2950
2970 IF LOWER$(d$)="d" THEN ae=8 ELSE ae=0
2980 st1=1:stat=5:GOSUB 2190
2990 f=0
3000 FOR i=0 TO anzahlfelder-1
3010   IF da$(i)<>"" THEN f(f)=i:f=f+1
3020 NEXT
3030 laenge=laenge-4:IF ae=0 THEN MODE 2
3040 FOR i=1 TO anzahl
3050   FOR i1=0 TO f-1
3060     PRINT#ae,m$(f(i1))SPACE$(laenge-LEN(m$(f(i1))))": "d$(i,f(i1))
3070   NEXT
3080   PRINT#ae
3090 NEXT
3100 IF ae=0 THEN PRINT "<< Tastendruck >>":CALL &BB06
3110 RETURN
3120 :
3130 REM ====================================
3140 REM    Fehlerroutine
3150 REM ====================================
3160 :
3170 IF ERR<>18 AND ERR<32 THEN RESUME NEXT
```

```
3180 ef=0 : |ERR,@ds$
3190 nf=0:IF RIGHT$(ds$,9)="not found" THEN nf=1:RESUME NEXT
3200 WINDOW #6,1,40,24,25:PAPER #6,1:PEN #6,0:CLS #6
3210 PRINT #6,CHR$(7)"*** Disk:";ds$;" in";ERL:PRINT #6,"
<ENTER>=Hauptmenu, sonst nochmal
3220 d$=INKEY$ : IF d$="" THEN 3220
3230 ef=1 : IF ASC(d$)=13 THEN RESUME 510
3240 RESUME NEXT
```

5.5 DISKMON - Ein Diskettenmonitor für den CPC

Haben Sie das Programm zum Lesen und Anzeigen einzelner Sektoren auf der Diskette ausprobiert? Dann wissen Sie ja, wie langsam die Bildschirmausgabe bei diesem Programm war. Im nun folgenden Diskmonitor, der ursprünglich auf dem zuvor genannten Programm beruht, haben wir die Ausgabe der Sektorinformation in ein kurzes Maschinenprogramm gepackt, um eine akzeptable Ausgabegeschwindigkeit zu erreichen. Auch haben wir weitere Kommandos eingebaut, die sicher bei der einen oder anderen Gelegenheit recht hilfreich sind. Im Einzelnen sind die folgenden Kommandos enthalten:

- *Lesen von Sektoren*
- *Verändern des Sektorinhalts*
- *Schreiben von Sektoren*
- *Ansehen des Directory*
- *Berechnen der Track- und Sektornummer aus der Blocknummer*

Das Lesen der Sektoren ist recht komfortabel gestaltet. Mit den vier Cursortasten kann jeder beliebige Sektor der Diskette erreicht werden. Dabei sind die Tasten 'Cursor up' und 'Cursor down' für die Auswahl des nächst höheren oder tieferen Tracks gedacht, die Tasten 'Cursor left' und 'Cursor right' lesen den Sektor, der sich vor oder hinter dem aktuellen Sektor befinden. Bei diesen Funktionen ist ein 'Wrap araoud' eingebaut. Haben Sie also den Sektor 9 eines Tracks gelesen und drücken dann die Taste 'Cursor right', so wird Ihnen der Sektor 1 des nächst höheren Tracks angezeigt.

Als Besonderheit wird bei gedrückt gehaltenen Cursor-Tasten nicht der Inhalt des gerade gelesenen Sektors angezeigt. Dadurch beschleunigt sich der Zugriff deutlich. Allerdings wird dann leider auch nicht immer der letzte gelesene Sektor angezeigt, so daß Sie diesen mit 'R (ENTER)' noch einmal lesen sollten.

Natürlich können Sie auch einen Sektor gezielt auswählen. Auch dazu dient das 'R'-Kommando. Geben Sie jedoch statt des ENTER eine Laufwerksnummer in der Form 0 für Laufwerk A oder 1 für Laufwerk B ein, so fragt das Programm nach den gewünschten Werten für Track und Sektor. Danach wird der Sektor gelesen und die erste Hälfte des Inhalts wird angezeigt.

Leider lassen sich die vollständigen 512 Bytes eines Sektors als ASCII- und HEX-Dump nicht auf einem Bildschirm unterbringen. Aus diesem Grund kann man mit den Tasten 'Shift Cursor left' und 'Shift Cursor right' zwischen den beiden Bildschirmseiten quasi hin- und herblättern.

Zum Ändern der gelesenen Information ist das 'M'-Kommando vorgesehen. Dies Kommando fragt nach der Adresse der zu ändernden Bufferadresse. Beantworten Sie diese Frage einfach mit ENTER, so wird als Adresse der Anfang des Buffers gewählt.
Die Adresse wird dann zusammen mit dem Inhalt angezeigt. Danach erwartet das Programm den neuen Wert der Bufferadresse. Als Eingaben sind die Hex-Zahlen 00 bis FF erlaubt, die ohne '&' eingegeben und mit ENTER abgeschlossen werden müssen. Danach wird dann die nächste Bufferadresse nebst Inhalt angezeigt. Wollen Sie jedoch den Inhalt der angezeigten Adresse nicht ändern, so können durch Drücken der ENTER-Taste zur nächsten Adresse kommen. Den Eingabemodus verlassen Sie durch 'X (ENTER)'.

Einen solchermaßen geänderten Sektor können Sie anschließend mit dem 'W'-Kommando auf die Diskette zurückschreiben. Durch die Eingabe von 'W (ENTER)' wird der Buffer auf den Sektor geschrieben, von dem er ursprünglich gelesen wurde. Sie können jedoch auch einen anderen Sektor wählen, wenn Sie wie beim 'M'-Kommando die Drivenummer, den Track und den Sektor angeben.

Das 'B'-Kommando stellt gewissermaßen ein Hilfsmittel für das Lesen von Dateien dar. Wie Sie aus den vorigen Kapiteln wissen, werden im Directory die Nummern der von der Datei belegten Blocks gespeichert. Die Umrechnung dieser Blocknummern in

die tatsächlichen Track- und Sektornummern ist zwar nicht schwer, aber doch lästig. Diese Arbeit nimmt uns das 'B'-Kommando ab. Dazu geben Sie die im Directory gefundene Blocknummer an, der CPC errechnet daraus automatisch die entsprechenden Werte und zeigt sie in der Kopfzeile an. Mit 'R (ENTER)' können Sie sich dann den errechneten ersten Sektor des Blocks lesen und anzeigen. Den zweiten Sektor erhalten Sie durch Drücken der Taste 'Cursor right'.

Das 'C'-Kommando zeigt Ihnen ein normales Inhaltsverzeichnis der eingelegten Diskette an. Diese Funktion ist recht hilfreich, da Sie nicht das Programm verlassen müssen, um sich einen Überblick über den Disketteninhalt zu verschaffen.
Da das Programm in weiten Teilen in BASIC geschrieben ist, eignet es sich natürlich hervorragend, um es mit eigenen Routinen und Kommandos zu erweitern. Eine mögliche Erweiterung ist es, das Format der eingelegten Diskette anzuzeigen und einen Diskettenwechsel zu erkennen. Haben Sie nämlich eine Diskette im Datenformat gelesen, so müssen Sie in der vorliegenden Fassung beim Wechsel des Formates erst den 'C'-Befehl abschicken, damit das AMSDOS das geänderte Format feststellt und den DPB entsprechend ändert. Dies Problem könnten Sie z.B. so lösen, daß Sie die Fehlermeldungen abschalten und an Hand der Flags nachsehen, ob der Sektor erfolgreich gelesen wurde. Ist das nicht der Fall, so könnten Sie die AMSDOS-Routine zur Ermittlung des Formates (&C56C) anspringen und den Sektor danach noch einmal lesen.

Eine weiterer Schwachpunkt, den Sie evtl. verbessern können, stellt das 'B'-Kommando dar. es funktioniert NUR im Standard-CP/M- und Vendorformat korrekt. Bei anderen Formaten sind die errechneten Ergebnisse falsch.

Eine weitere mögliche Ergänzung wäre es, den Sektorinhalt auf einem Drucker auszugeben. Dafür könnte die Hexdump-Routine aus dem Beispiel zum Lesen eines einzelnen Sektors gute Dienste leisten.

Der Diskettenmonitor

```
1000 ' *****    DISKMONITOR   *****
1010 ' *****    RBR 17/4/85   *****
1020 '
1030 DEFINT a-l,n-z
1040 MEMORY &A000-1
1050 MODE 2
1060 LOCATE 10,10:PRINT"einen Moment bitte ..... "
1070 GOSUB 9000: 'windows definieren und ml poken
1080 CLS #1
1090 bef$=" "
1100 buffer = &A2
1110 cmd$="CRWMB"+CHR$(&F0)+CHR$(&F1)+CHR$(&F2)+CHR$(&F3)
1120 cmd$=cmd$+CHR$(&F6)+CHR$(&F7)
1130 FOR i=1 TO LEN(cmd$):cmd1$=cmd1$+MID$(cmd$,i,1)+",":NEXT
1140 cmd1$=LEFT$(cmd1$,LEN(cmd1$)-1)
1150 sector = 1:track = 0:drive = 0:befehl = &84:catflag = 0
1160 '
1170 '
2000 ' ***** hauptprogramm *****
2010 '
2020 POKE &A108,buffer:POKE &A10A,buffer
2030 LOCATE #1,1,1:CLS
2040 GOSUB 6000 : 'eine page zeigen
2050 PRINT"befehl "cmd1$" >"
2060 bef$=UPPER$(INKEY$):IF bef$="" THEN 2060 ELSE CLS
2070 ON INSTR(cmd$,bef$)GOTO
3100,3150,3200,3250,3300,3500,3550,3600,3650,3400,3450
2080 GOTO 2050
2090 '
2100 '
3000 ' ***** kommandotabelle *****
3010 '
3100 ' ***** d-kommando *****
3110 GOSUB 4000 : GOTO 2050
3120 '
3150 ' ***** r-kommando *****
3160 GOSUB 5000 : GOTO 2020
```

```
3170 '
3200 ' ***** w-kommando *****
3210 GOSUB 5100 : GOTO 2020
3220 '
3250 ' ***** m-kommando *****
3260 GOSUB 7000 : GOTO 2030
3270 '
3300 ' ***** b-kommando *****
3310 GOSUB 8000 : GOTO 2030
3320 '
3400 ' ***** erste page zeigen *****
3410 POKE &A108,buffer:POKE &A10A,buffer:GOTO 2030
3420 '
3450 ' ***** zweite page zeigen *****
3460 POKE &A108,buffer + 1:POKE &A10A,buffer + 1:GOTO 2030
3470 '
3500 ' ***** track + 1 *****
3510 track=-(track<39)+track+((track=39)*39):GOSUB 5500:GOTO 2020
3520 '
3550 ' ***** track - 1 *****
3560 track=(track>0)+track+(-(track=0)*39):GOSUB 5500:GOTO 2020
3570 '
3600 ' ***** sector - 1 *****
3610 sector=(sector>1)+sector+(-(sector=1)*(PEEK(&A8A0+(drive*64))-1))
3620 IF sector=9 THEN GOTO 3550 ELSE GOSUB 5500:GOTO 2020
3630 '
3650 ' ***** sector + 1 *****
3660 sector=-(sector<9)+sector+((sector=9)*(PEEK(&A8A0+(drive*64))-1))
3670 IF sector=1 THEN GOTO 3500 ELSE GOSUB 5500:GOTO 2020
3680 '
4000 ' ***** directory anzeigen *****
4010 WINDOW SWAP 0,1:CLS:PRINT:|DIR:WINDOW SWAP 1,0:catflag = 1
4020 RETURN
4030 '
5000 ' ***** sector lesen *****
5010 PRINT"      sektor lesen":PRINT
5020 GOSUB 5800:GOTO 5500
5030 '
5100 ' ***** sector schreiben *****
5110 befehl = &85:PRINT"      sektor schreiben":PRINT
```

```
5120 GOSUB 5800
5130 '
5500 ' ***** sektor lesen/schreiben *****
5510 POKE &A100,befehl
5520 POKE &A104,drive
5530 POKE &A105,track
5540 POKE &A106,sector-1 + PEEK(&A89F+drive*&40)
5550 CALL &A0A0:befehl = &84:RETURN
5560 '
5800 ' ***** drive,track sector holen *****
5810 INPUT"      drive (0/1) oder enter";drive$:IF drive$="" THEN RETURN
5820 drive = VAL(drive$)
5830 INPUT"      track  (0-39)          ";track
5840 INPUT"      sector (1-9)           ";sector
5850 RETURN
5860 '
6000 ' ***** anzeigen *****
6010 IF catflag = 1 THEN catflag = 0:CLS #1
6020 WINDOW SWAP 0,1
6030 PRINT USING"      | drive ###   |   track ###   |   sector ###
|";drive,track,sector
6040 PRINT
6050 IF INKEY$ <> "" GOTO 6070
6060 IF INSTR("BW",bef$) = 0 THEN CALL &BB81:CALL &A000:CALL &BB84
6070 WINDOW SWAP 1,0
6080 RETURN
6090 '
7000 ' ***** modify *****
7010 PRINT"    buffer aendern"
7020 INPUT"bufferadresse ";buadr$
7030 IF buadr$ = "" than buadr=0 ELSE buadr=VAL("&"+buadr$)
7040 madr=buadr+(buffer*256)
7050 PRINT HEX$(buadr,4);"  ";HEX$(PEEK(madr),2);"  ";
7060 INPUT newbyt$:newbyt$=UPPER$(newbyt$)
7070 IF newbyt$="X" THEN RETURN
7080 IF newbyt$="" THEN GOTO 7110
7090 newbyt=VAL("&"+newbyt$):newbyt$=""
7100 POKE madr,newbyt
7110 buadr=buadr+1:GOTO 7040
7120 '
```

```
8000 ' ***** blocknummer umrechnen *****
8010 PRINT"     blocknummer (amsdos-format)"
8020 INPUT"     blocknummer (in hex)  ";block$
8030 block$="&"+block$:a=VAL(block$)
8040 sectcnt=a*2+18
8050 track = INT((a*2+18)/9)
8060 sector = (a*2+18) MOD 9 + 1
8070 RETURN
8080 '
9000 ' ***** windows definieren *****
9010 WINDOW #0,1,80,20,25
9020 WINDOW #1,1,80,1,19
9030 '
9100 ' ***** DATAS FUER HEXDUMP/SECTOR I-O *****
9110 FOR i = &A000  TO &A0BC
9120 READ byte : POKE i,byte : s = s + byte : NEXT
9130 DATA
&C3,&2D,&A0,&7C,&CD,&08,&A0,&7D,&F5,&1F,&1F,&1F,&1F,&CD,&11,&A0
9140 DATA
&F1,&E6,&0F,&C6,&30,&FE,&3A,&38,&02,&C6,&07,&C3,&5A,&BB,&3E,&0D
9150 DATA
&CD,&5A,&BB,&3E,&0A,&C3,&5A,&BB,&3E,&20,&C3,&5A,&BB,&2A,&07,&A1
9160 DATA
&ED,&5B,&09,&A1,&FD,&21,&01,&00,&06,&10,&E5,&C5,&ED,&4B,&0B,&A1
9170 DATA
&09,&CD,&03,&A0,&C1,&E1,&CD,&28,&A0,&CD,&28,&A0,&7E,&CD,&08,&A0
9180 DATA
&FD,&2B,&CD,&28,&A0,&A7,&ED,&52,&19,&28,&2C,&23,&10,&EE,&01,&F0
9190 DATA
&FF,&09,&06,&10,&CD,&28,&A0,&7E,&E6,&7F,&FE,&20,&38,&02,&18,&02
9200 DATA
&3E,&2E,&CD,&5A,&BB,&23,&10,&EF,&CD,&1E,&A0,&E5,&37,&ED,&52,&E1
9210 DATA
&C8,&FD,&E5,&C1,&0D,&18,&B1,&05,&3E,&10,&90,&4F,&06,&00,&ED,&42
9220 DATA
&23,&41,&18,&D0,&4E,&4B,&3A,&4C,&44,&09,&41,&2C,&22,&20,&22,&0D
9230 DATA
&21,&00,&A1,&CD,&D4,&BC,&22,&01,&A1,&79,&32,&03,&A1,&21,&04,&A1
9240 DATA &5E,&23,&56,&23,&4E,&21,&00,&A2,&DF,&01,&A1,&C9,&2D
9250 IF s <> 20027  THEN PRINT "error in checksum" : END
```

```
9260 S=0
9270 FOR i = &A100  TO &A10C
9280 READ byte : POKE i,byte : s = s + byte : NEXT
9290 DATA &84,&00,&00,&00,&00,&01,&00,&A2,&FF,&A2,&00,&5E
9300 IF s <> 806 THEN PRINT "error in checksum" : END
9310 RETURN
```

5.6 Der Diskettenmanager

Der Diskettenmanager dient der Erstellung und Pflege von Arbeitsdisketten für die Schneider CPC-Rechner. Die einzelnen Funktionen werden unter verschiedenen Menüpunkten angeboten, so daß Sie beispielsweise zum Formatieren einer Diskette nicht mehr unbedingt CP/M booten müssen. Nachdem Sie den Diskettenmanager abgetippt haben, sollten Sie ihn zunächst ganz selbstverständlich auf Diskette abspeichern. Nachdem Sie das Programm dann gestartet haben, erhalten Sie folgendes Menü auf dem Bildschirm:

1) Umbenennen von Files
2) Löschen von Files
3) Formatieren einer Diskette
4) Anzeigen Disketteninhalt
5) Kopieren von Nicht-Programm-Dateien
6) APPEND - Zusammenbinden von 2 Dateien
7) Zeige Inhalt einer Datei (TYPE)

9) Beende Programm

Um einen Menüpunkt anzuwählen, brauchen Sie lediglich die entsprechende Zifferntaste zu betätigen (ein Betätigen der ENTER-Taste ist also *nicht* erforderlich). Augenblicklich gelangen Sie in die gewählte Routine.

Um die folgenden Erläuterungen zu vereinfachen wollen wir jeden Menüpunkt einzeln besprechen:

1) Umbenennen von Dateien

Dieser Menüpunkt unterstützt Sie beim Umbenennen von verschiedenen Dateien. Sie werden zunächst nach dem Namen der umzubenennenden Datei gefragt. Geben Sie den Namen ein und betätigen Sie anschließend die ENTER-Taste.

Wollen Sie einen Menüpunkt frühzeitig verlassen, so betätigen Sie bei der Eingabe eines Feldes ohne jede weitere Eingabe einfach die ENTER-Taste; Sie gelangen dann wieder ins Hauptmenü zurück.

Nachdem Sie den Namen der Datei eingegeben haben, werden Sie nach dem neuen Namen gefragt. Wollen Sie beispielsweise die Datei mit Namen "A" in "B" umtaufen, so geben Sie als ersten Namen "A" und dann "B" ein, jedesmal mit ENTER abschließen.

Sie werden aufgefordert, die Diskette einzulegen, auf der sich die umzubenennende Datei befindet. Nachdem Sie dann eine beliebige Taste auf der Tastatur betätigt haben, wird Ihr Auftrag ausgeführt.

2) Löschen von Dateien

Dieser Menüpunkt ist der mit Abstand am komfortabelsten gestaltete im Diskettenmanager. Er dient dazu, Disketten "aufzuräumen". Sie werden sich noch wundern, was sich so alles auf einer Diskette an Programmen und Daten ansammeln kann. Mit dieser Routine können Sie Ihre Disketten von alten und unwichtigen Daten und Prorgrammen ein für allemal befreien.

Nachdem das Diskettеninhaltsverzeichnis auf dem Bildschirm angezeigt wurde, können Sie mit den Cursortasten die verschiedenen Dateinamen "anfahren". Durch Betätigen der COPY-Taste können Sie eine Datei zum Löschen markieren - das entsprechende Feld blinkt dann rot/rosé - durch nochmaliges

Betätigen der COPY-Taste können Sie diese Markierung auch wieder aufheben.

Haben Sie alle Dateien markiert, die Sie auf der Diskette eliminieren wollen - so betätigen Sie bitte die ENTER-Taste. Es erfolgt dann eine Sicherheitsabfrage, ob Sie wirklich alle Dateien markiert haben und ob tatsächlich die Dateien alle gelöscht werden sollen. Haben Sie diese Abfragen bestätigt, so gibt es keinen Weg mehr zurück. Alle markierten Dateien auf der Diskette werden gelöscht.

Bei uns hat sich diese Routine zum "Putzen" von Disketten bewährt, da man bei dieser Routine nicht so schnell eine Datei gelöscht hat, wie das beim Benutzung des |ERA-Kommandos der Fall ist.

3) Formatieren einer Diskette

Dieser Menüpunkt ist besonders interessant, da es ansonsten nicht möglich ist, aus BASIC heraus eine Diskette zu formatieren und komfortable Programme sollten diesen Menüpunkt eigentlich immer anbieten.

Diese Formatierungsroutine ist sogar schneller, als die unter CP/M benutzte. Allerdings wird hier auf eine Sicherheitsabfrage verzichstet, d.h es wird nicht jeder formatierte Sektor überprüft. Allerdings hat sich bei uns noch nie ein Fehler eingeschlichen, wer aber auf Nummer sicher gehen will, sollte trotzdem die CP/M-FORMAT-Routine benutzen.

Sie können Ihre Diskette in Vendor- oder im Data-Only-Format formaieren; ein Formatieren IBM-Format ist nicht vorgesehen

Vendor-Format bedeutet, daß die Diskette als CP/M-Systemdiskette formatiert wird, auf der sich das CP/M allerdings nicht befindet - es könnte aber nachträglich aufkopiert werden.

Im Normalfall dürften Sie Ihre Disketten im Data-Only-Format formatieren, da hier die gesamte Speicherkapazität zum Spei-

chern von Daten und Programmen zur Verfügung steht. Sie werden aufgefordert, eine Diskette einzulegen und eine beliebige Taste zu betätigen.

5) Kopieren von Nicht-Programm-Dateien

Programmdateien können Sie mit diesem Menüpunkt nicht kopieren, lediglich einfache sequentielle Dateien. Sie können auswählen, ob Quell- und Zieldatei auf einer oder auf zwei verschiedenen Disketten sein soll. Sollte sich Quell- und Zieldatei auf ein und derselben Diskette befinden, so unterliegen Sie keinerlei Beschränkungen bezüglich der Dateilänge. Wird von einer Diskette auf eine andere kopiert, so sind höchstens 250 Datensätze möglich. Sollte die Datei hierfür zu lang sein, so wird Ihnen dies auf dem Bildschirm angezeigt.

6) APPEND - Zusammenbinden von 2 Dateien

Mit diesem Menüpunkt sind Sie in der Lage, zwei Dateien (wieder *keine* Programmdateien) zusammenzubinden. Geben Sie einfach die Namen der beiden zusammenzubindenden Dateien ein; der erste eingegebene Dateiname stellt auch die spätere zusammengebundene Datei dar. Die Zieldatei wird auf derselben Diskette erstellt, auf der sich auch die beiden Quelldateien befinden.

7) Zeige Inhalt einer Datei

Dieser Menüpunkt mach dasselbe, wie das CP/M-Kommando TYPE - es wird Ihnen der Inhalt einer Datei als ASCII-Zeichen-Folge auf dem Bildschirm dargestellt. Durch Betätigen einer beliebigen Taste können Sie die Ausgabe anhalten und auch wieder fortfahren.

Ich bin sicher, die eine oder andere Routine wird Ihnen gute Dienste bei der Arbeit mit dem Schneider-Laufwerk leisten.

Der Diskettenmanager

```
10 '*****************************************
20 '*** Diskettenmanager ---- JS/RB 1.5.85 ***
30 '*****************************************
40 :
50 OPENOUT "dummy" : MEMORY HIMEM-1 : CLOSEOUT
60 DATA &3e,&00,&32,&2f,&80,&3a,&2f,&80,&57,&3a,&30,&80,&5f,&3a,&31,&80
70 DATA &4f,&21,&35,&80,&df,&32,&80,&3a,&2f,&80,&fe,&27,&c8,&3c,&32,&2f
80 DATA &80,&21,&35,&80,&06,&09,&77,&23,&23,&23,&23,&10,&f9,&18,&d6,&27
90 DATA &00,&41,&52,&c6,&07
100 :
110 FOR i=&8000 TO &8034
120   READ d
130   POKE i,d
140   s=s+d
150 NEXT
160 IF s<>4258 THEN PRINT"*** Fehler in Datas ***"
170 MEMORY &7FFF
180 :
190 CLEAR : DEFINT b-z
200 MODE 1 : INK 0,11 : INK 1,16,6 : INK 2,0 : INK 3,24 : PEN 3 : PAPER
2 : CLS
210 BORDER 0
220 CLS : ORIGIN 0,0,0,640,340,400 : CLG 3
230 PAPER 3 : PEN 2 : LOCATE 14,2 : PRINT"Diskettenmanager" : PAPER 2 :
PEN 3
240 LOCATE 1,7
250 PRINT"1) Umbenennen von Files"
260 PRINT"2) Loeschen von Files"
270 PRINT"3) Formatieren einer Diskette"
280 PRINT"4) Anzeigen Disketteninhalt"
290 PRINT"5) Kopieren von NICHT-PROGRAMM-FILES
300 PRINT"6) APPEND - Zusammenbinden von 2 Files"
310 PRINT"7) Zeige Inhalt eines Files"
320 PRINT : PRINT"9) Beende Programm"
330 LOCATE 1,20 : PEN 1 : PRINT "Ihre Wahl:"
340 a$=INKEY$ : IF a$="" THEN 340
350 we=VAL(a$) : IF we<1 OR (we>7 AND we<>9) THEN 340
```

```
360 PEN 3 : ON we GOTO 1200,550,380,1120,1340,1700,1900,1900,2100
370 '
380 '================================
390 '    Formatiere Diskette
400 '================================
410 :
420 CLS : PRINT"1) Vendor-Format" : PRINT : PRINT"oder" : PRINT :
PRINT"2) Data-Only-Format"
430 a$=INKEY$ : IF a$<"1" OR a$>"2" THEN 430
440 IF a$="1" THEN f$="Vendor" : y=&41 ELSE f$="Data-Only" : y=&C1
450 x=&8035
460 FOR i=1 TO 9
470   POKE x,0 : POKE x+1,0 : POKE x+2,y : POKE x+3,2
480   x=x+4
490   y=y+2 : IF (y AND &F) = &B THEN y=y-9
500 NEXT
510 PRINT : PRINT"Bitte die Diskette einlegen" : PRINT"und eine Taste
druecken..."
520 IF INKEY$="" THEN 520
530 CALL &8000 : GOTO 190
540 '
550 '============================
560 '   Loesche Files
570 '============================
580 :
590 DIM a$(65),era(64)  :CLS
600 LOCATE 4,11 : PRINT"Directory wird gelesen ..."
610 PRINT : PRINT"    einen Moment, bitte
620 lin$=STRING$(40,154)
630 FOR i=0 TO 63
640   a$(i)=STRING$(11,32)
650 NEXT
660 a=PEEK(&BB5A) : POKE &BB5A,&C9 : CAT : POKE &BB5A,a
670 anz=PEEK(&A912) : a=PEEK(&A79C)*256 + PEEK(&A79B)+1
680 CLS
690 FOR i=0 TO anz
700   POKE @a$(i)+1,a-(INT(a/256)*256)
710   POKE @a$(i)+2,INT(a/256) : a=a+14
720 NEXT
730 FOR i=0 TO anz
```

```
740 IF ASC(LEFT$(a$(i),1)) = 0 THEN a=i : i=anz : GOTO 770
750 a$(i)=LEFT$(a$(i),8) + "." + RIGHT$(a$(i),3)
760 PRINT a$(i),
770 NEXT : anz=a
780 LOCATE 1,22 : PRINT lin$
790 txt$="Auf dieser Disc loeschen? (J/N)" : GOSUB 1090
800 GOSUB 1100 : IF LOWER$(a$)="j" THEN 840
810 txt$="ENTER = neue Disc, X = Ende" : GOSUB 1090
820 GOSUB 1100 : IF a$=CHR$(13) THEN ERASE a$,era : GOTO 550
830 IF LOWER$(a$)="x" THEN 190 ELSE 820
840 txt$="Taste 'COPY' markiert zum loeschen" : GOSUB 1090
850 x=0 : xc=1 : yc=1 : GOTO 950
860 x=temp : GOSUB 1100 : IF a$=CHR$(13) THEN 1000
870 IF ASC(a$)=&F0 THEN x=x-3 : IF x<0 THEN 860 ELSE 950
880 IF ASC(a$)=&F1 THEN x=x+3 : IF x>anz-1 THEN 860 ELSE 950
890 IF ASC(a$)=&F2 THEN x=x-1 : IF x<0 THEN 860 ELSE 950
900 IF ASC(a$)=&F3 THEN x=x+1 : IF x>anz-1 THEN 860 ELSE 950
910 IF ASC(a$)<>&E0 THEN 860
920 era(x) = era(x) XOR 1 'Invertiere Feld
930 LOCATE xc,yc : PAPER era(x)
940 PRINT a$(x); : PAPER 2 : GOTO 860
950 yco = x\3+1 : xco = (x- ((yco-1)*3))*13+1
960 LOCATE xc,yc : PAPER (era(temp)=0)*-2+era(temp)
970 PRINT a$(temp); : PAPER 2
980 LOCATE xco,yco : PAPER era(x) : PRINT a$(x); : PAPER 2
990 xc=xco : yc=yco : temp=x : GOTO 860
1000 LOCATE xc,yc : PAPER 2+(era(x)<>0) : PRINT a$(x); : PAPER 2 :
txt$="Alle Files markiert? (J/N)" : GOSUB 1090
1010 GOSUB 1100 : IF LOWER$(a$)<>"j" THEN 840
1020 txt$="Wirklich loeschen (J/N)" : GOSUB 1090
1030 GOSUB 1100 : IF LOWER$(a$)<>"j" THEN 840
1040 txt$="Files werden geloescht !" : GOSUB 1090
1050 FOR i=0 TO anz
1060 IF era(i) THEN |ERA,@a$(i)
1070 NEXT
1080 GOTO 190
1090 LOCATE 4,24 : PRINT CHR$(20);txt$ : RETURN
1100 a$=INKEY$ : IF a$="" THEN 1100 ELSE RETURN
1110 '
1120 '================================
```

```
1130 '   Zeige Disketteninhalt
1140 '================================
1150 :
1160 CLS : CAT
1170 PRINT : PRINT"Tastendruck ..."
1180 IF INKEY$="" THEN 1180 ELSE 190
1190 :
1200 '================================
1210 '    Umbennen eines Files
1220 '================================
1230 :
1240 CLS
1250 INPUT "Name des alten Files : ",falt$ : IF falt$="" THEN 190
1260 INPUT "Neuer Name des Files : ",Neu$ : IF neu$="" THEN 190
1270 PRINT : PRINT"Legen Sie bitte die Diskette ein,"
1280 PRINT : PRINT"auf der sich das File "CHR$(24);falt$;CHR$(24)
1290 PRINT : PRINT"befindet, und betaetigen Sie eine Taste"
1300 IF INKEY$="" THEN 1300
1310 |REN,@neu$,@falt$
1320 GOTO 190
1330 '
1340 '================================
1350 '     Kopiere Nicht-Prg-Files
1360 '================================
1370 :
1380 CLS : INPUT "Name des Quellfiles : ",quell$ : IF quell$="" THEN 190
1390 INPUT "Name des Zielfiles  : ",ziel$ : IF ziel$="" THEN 190
1400 INPUT "Auf andere Diskette kopieren (J/N) : ",jn$
1410 IF LOWER$(jn$)="n" THEN 1620
1420 DEFSTR a : DIM a(250)
1430 PRINT : PRINT"Bitte legen Sie die Quelldiskette ein"
1440 PRINT"und betaetigen Sie eine Taste"
1450 IF INKEY$="" THEN 1450
1460 OPENIN quell$
1470 WHILE NOT EOF
1480   LINE INPUT #9,a(xc)
1490   xc=xc+1 : IF xc>250 OR FRE(0)<2000 THEN PRINT CHR$(7)"File ist zu
gross" : FOR i=1 TO 1000:NEXT:CLOSEIN:RUN 190
1500 WEND
1510 CLOSEIN
```

```
1520 PRINT : PRINT CHR$(7)"Bitte legen Sie die Zieldiskette ein"
1530 PRINT"und betaetigen Sie eine Taste"
1540 IF INKEY$="" THEN 1540
1550 PRINT : PRINT"Es werden"xc"Datensaetze kopiert."
1560 OPENOUT ziel$
1570 FOR i=0 TO xc
1580   PRINT #9,a(i); : IF LEN(a(i))<255 THEN PRINT #9
1590 NEXT
1600 CLOSEOUT
1610 RUN 190
1620 OPENIN quell$ : OPENOUT ziel$
1630 WHILE NOT EOF
1640   LINE INPUT #9,a$
1650   PRINT #9,a$
1660 WEND
1670 CLOSEIN : CLOSEOUT
1680 GOTO 190
1690 '
1700 '================================
1710 '    Verbindet zwei Files
1720 '================================
1730 '
1740 CLS : INPUT "Name des ersten Files : ",f$(0) : IF f$(0)="" THEN 190
1750 INPUT "Name des zweiten Files: ",f$(1) : IF f$(1)="" THEN 190
1760 INPUT "Name des Zielfiles     : ",f3$ : IF f3$="" THEN 190
1770 PRINT : PRINT"ok"
1780 OPENOUT f3$
1790 FOR i=0 TO 1
1800   OPENIN f$(i)
1810   WHILE NOT EOF
1820     LINE INPUT #9,a$
1830     PRINT #9,a$
1840   WEND
1850   CLOSEIN
1860 NEXT
1870 CLOSEOUT
1880 GOTO 190
1890 '
1900 '==============================
1910 '    Zeige Inhalt eines Files
```

```
1920 '============================
1930 :
1940 ON ERROR GOTO 2090
1950 CLS : INPUT "Name des Files : ",f$ : IF f$="" THEN 190
1960 MODE 2 : PEN 1 : INK 1,1 : PRINT"Inhalt des Files "f$
1970 PRINT STRING$(80,"-")
1980 OPENIN f$
1990 WHILE NOT EOF
2000   LINE INPUT #9,a$
2010   PRINT a$
2020   IF INKEY$="" THEN 2040
2030   IF INKEY$="" THEN 2030
2040 WEND
2050 CLOSEIN
2060 PRINT : PRINT"Tastendruck"
2070 IF INKEY$="" THEN 2070
2080 GOTO 190
2090 PRINT"File Type Error - Kein ASCII File !!" : RESUME 2060
2100 CLS : PEN 1 : PRINT"Programm beendet
```

Stichwortverzeichnis

A

A 63
A: 34
ABSPEICHERN 61
ADREß-MARKE 228
AMSDOS 26,32,34,46,62,91,96
AMSDOS 125,139,151,167,235,247,329
AMSDOS-KOMMANDOS 62
AMSDOS-ROM 247
ANSCHLUß-BELEGUNG 172
ANZEIGEN 41
APPEND 108
ASCII-DATEI 139
ASSEMBLER 146
AUSGABEBUFFER 96

B

B 63
B: 34
BACKUP-DATEIEN 59
BAS 59
BASIC 21
BAUD 14
BEFEHLE, EXTERNE 62
BEFEHLSERWEITERUNG 145
BENUTZERBEREICH 53,54,69
BETRIEBSSYSTEM 96
BINÄRDATEI 61
BLOCK 25
BOOTEN 34,45
BOOTGEN 34,45
BUFFER 93,96

C

CANCEL 27
CARRIAGE RETURN 77,98
CAS CATALOG 128
CAS IN ABANDON 127
CAS IN CLOSE 127
CAS IN DIREKT 127
CAS IN OPEN 127
CAS OUT ABANDON 127
CAS OUT CHAR 127
CAS OUT CLOSE 127
CAS OUT DIREKT 128
CAS OUT OPEN 127
CAS RETURN 127
CAS TEST EOF 127
CAT 49,57,125
CHAIN MERGE 49,49,329
CHKDISC 30
CHKDISK 34
CLOAD 34
CLOSE 140
CLOSEIN 49
CLOSEOUT 49
CON 42
CONTROL-SEQUENZEN 31
COPYDISC 29,34
CP/M 2.2 22
CP/M 3.0 17,18,21,22
CP/M-KOMMANDOS 23
CPC 119
CPC-RECHNER 13
CPM-VERSIONEN 22
CSAVE 34

D

DATA-ONLY-FORMAT 33
DATEI 15,74,75,87,109,378
DATEI-TYP 219
DATEIBEFEHLE 54
DATEIKENNUNG 36
DATEINAME 51
DATEITYP 36
DATEIVERWALTUNGSPROGRAMM 378
DATENFELD 76,77,350,378
DATENFORMAT 211
DATENSATZ 75,76,77,82,109,350,378
DATENSPEICHERUNG 37,74,108
DATENSPEICHERUNG, RELATIV 350
DBASE 21
DERR 123
DIM-ANWEISUNG 86
DIR 34,39,57,63
DIRECTORY 23,39
DIRECTRORY 212
DISC CATALOG 142
DISC FORMAT PARAMETER 155
DISC IN ABANDON 132
DISC IN CHAR 106,132
DISC IN CLOSE 131
DISC IN DIREKT 133
DISC OUT ABANDON 140
DISC OUT CHAR 140,141
DISC OUT CLOSE 139
DISC OUT OPEN 136
DISC RETURN 134
DISC TEST EOF 49,63,134
DISC-CONTROLLER 125
DISC.IN 49,63
DISC.OUT 49,64
DISCCHK 30,34
DISCCOPY 28,34
DISK IN OPEN 128
DISKETTE 16,24

DISKETTE, DIE MITGELIEFERTE 20
DISKETTENFEHLER 119,122
DISKETTENFORMAT 209
DISKETTENFORMATE 198
DISKETTENLAUFWERK 13,16
DISKETTENMONITOR 395
DISC 63
DOS-VEKTOREN 125
DR. LOGO 20
DREI-ZOLL-DISKETTEN 16
DRIVE PARAMETER 49,64,151

E

ED 34
EINGABEBUFFER 96
EINRICHTEN DATEI 110
END OF FILE 132
ENTWICKLUNG 16
EOF 49,83,85,132
ERA 34,40,49,64
ERR 123
ERWEITERUNGEN 356

F

FDC 164,167
FDC 765 170
FDC-PROGRAMMIERUNG 180
FEHLER 329
FEHLERMELDUNGEN 329,336
FELDVARIABLE 85,87,88
FILECOPY 28,34,35,38
FIRMWARE-MANUAL 127
FLOPPY 13
FLOPPYBUFFER 93,96
FLOPPYCONTROLLER 167
FLOPPYINTERFACE 167
FORMAT 30,34,161

FORMATE 32
FORMATIEREN 18,24
FORMATIERUNGSFORMATE 31
FREIER ZUGRIFF 75,108,109,350
FUNKTIONSWEISEN 117

G

GAP 228
GARBAGE COLLECTION 93
GET 106

H

HANDBUCH 15
HEAD LOAD TIME 152
HEAD LOADE TIME 154
HEADER 219
HIMEM 93

I

IBM-FORMAT 33,211
ID 228
IGNOR 27
INDEXLOCH 16,188
INHALTSVERZEICHNIS 39,212
INPUT 49,103
INTERNER POINTER 108
INTERNER ZEIGER 82
ISAM 119

J

JOKER 38

K

KANAL 96
KARTEIKASTEN 109
KASSETTE 13
KASSETTENLAUFWERK 14,74
KENNUNG 219
KERNAL 152
KEY-INDEX-DATEIEN 119
KL FAR PCHL 152
KL FIND COMMAND 146
KLIRRFAKTOR 220
KOMMANDOIS 23
KOMPATIBILITÄT 123
KOPIEREN 27,35

L

LAUFWERK 51
LESEBEFEHL 82
LESEKOPF 17
LINE INPUT 49,103
LIST 49
LOAD 8,49
LOGO 17,20
LÖSCHEN 40
LST 42

M

MAGNETISMUS 220
MASCHINENPROGRAMM 61,125
MEMORY 93
MERGE 49,329
MESSAGE ON/OFF 151
MFM 220
MOVCPM 34,45

O

ON-ERROR-GOTO 122,151,336
OPENIN 49
OPENOUT 49,125

P

PASCAL 87,115
PATCHEN 143
PIP 34,42
POINTER INTERNER 82,83,108
PRINT 49,98
PROGRAMM 74
PROGRAMMDATEI 74
PROGRAMMIERUNG 79
PROZESSOR ..... --21

P

PUN 42

R

RAM 119,125,210,235
RANDOM ACCESS 108,109
RDR 42
READ SECTOR 156
READ-ONLY 43,218
READ-WRITE 43
RECALIBRATE 193
RECORD 26,115,356
RECORDNUMMER 115
RELATIV 37,108
RELATIVE DATENSPEICHERUNG 37,108,350
REN 34,41,49,68
RESIDENT 34
RESTART 152
RETRY COUNT 27,164

ROM 62,119,125,210,235,247
ROM-LISTING 249
RSX 146,356,378

S

SATENSATZ 75
SAVE 34,49,59
SCHNITTSTELLE 169
SCHREIBEN 160
SCHREIBKOPF 17
SEEK TRACK 163,193
SEKTOR 24,25,157,159,160
SEQUENTIEL 37,74
SEQUENTIELLE DATEI 79,87
SEQUENZ 74
SETUP 46
SOFTWARE 21
SPEED WRITE 14,125
SPEICHER, EXTERNER 16
SPEICHER, FLUECHTIGER 13
SPEICHERKAPAZITAET 18
SPUR 24,25,163
STAT 34,43
STATUSREGISTER 198
STRINGS 93
SYNTAX 87
SYNTAX-DIAGRAMM 88
SYSGEN 34,45
SYSTEM VERSCHIEBEN 45
SYSTEM-DISKETTE 247
SYSTEM-FORMAT 210
SYSTEM-RAM 235
SYSTEMDISKETTE 22

T

TAPE 49,69
TAPE.IN 49,69
TAPE.OUT 49,69
TECHNIK 167
TERMINAL COUNT 181
TEST DRIVE 164
TRACK 163
TRANSIENT 34
TRENNSYMBOL 77,85,98
TTL 167
TYPE 34,41

U

ÜBERPRÜFEN 30
ÜBERSCHREIBEN 17
UMBENENNEN 41
UMSCHALTEN LAUFWERK 41
UNTERSCHIEDE 119
USER 34,49,53,69
USER-BEREICH 53

V

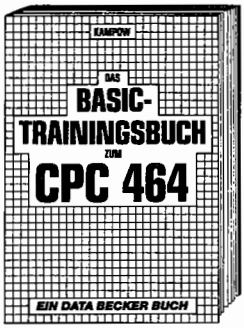
VARIABLENADRESSE 71
VEKTOR 143
VERSTECKTE KOMMANDOS 151

W

WAGENRÜCKLAUF 77,98
WILD CARDS 38
WIRTH 87
WORDSTAR 21
WRITE SEKTOR 49,98,160

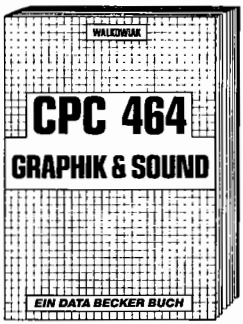
Z

Z-80 21
ZEIGER 82,83,108
ZEIGER, INTERNER 82,108



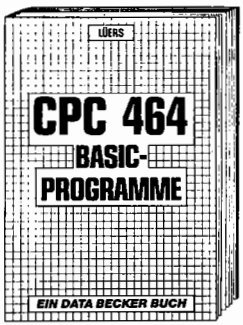
CPC 464 BASIC? Kein Problem! Mit diesem Trainingsbuch lernen Sie von Grund auf nicht nur die einzelnen Befehle und ihre Anwendungen, sondern auch einen richtig sauberen Programmierstil. Von der Problemanalyse über den Flußplan bis zum fertigen Programm. Dazu viele Übungsaufgaben mit Lösungen und zahlreichen Beispielen. Schlichtweg unentbehrlich.

Kampow
Das BASIC-Trainingsbuch zum CPC 464
285 Seiten, DM 39,— ISBN 3-89011-038-X



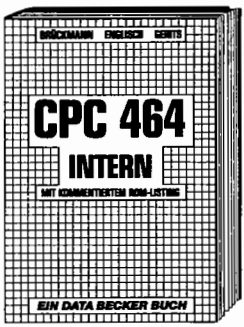
In diesem erstklassigen Buch wird gezeigt, wie man die außergewöhnlichen Grafik- und Soundmöglichkeiten des CPC 464 nutzt. Natürlich mit vielen interessanten Beispielen und nützlichen Hilfsprogrammen. Aus dem Inhalt: Grundlagen der Grafikprogrammierung, Sprites, Shapes und Strings, mehrfarbige Darstellungen, Koordinatentransformation, Verschiebungen, Drehungen, Rotation, 3-D-Funktionsplotter, CAD, Synthesizer, Miniorgel, Hüllkurven und vieles mehr.

Walkowiak
CPC 464 Graphik & Sound
220 Seiten, DM 39,— ISBN 3-89011-050-9



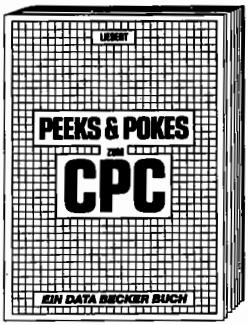
Spitzenprogramme vom Disassembler bis zum Sporttabellenprogramm – mit spannenden Superspielen und kompletten Anwendungsprogrammen: mit Hexdump, Grafik- und Soundeditor, deutsche Umlaute, Mathematikzeichensatz, ausführliche Fehlermeldungen, Variablenreferenzliste, Kalender, Disassembler, Langspielplattenverwaltung Texteditor, Codeknacker, Zahlsystemumrechner.

Lüers
CPC 464 BASIC-Programme
185 Seiten, DM 39,— ISBN 3-89011-049-5



Wirklich alle Geheimnisse des CPC 464 lüftet dieses Standardwerk, das für den Fortgeschrittenen BASIC-Programmierer unentbehrlich, für den Assembler-Programmierer ein absolutes Muß ist. Neben dem ausführlich dokumentierten und kommentierten BASIC-ROM-Listing enthält es umfangreiche Kapitel zu Speicheraufteilung, Prozessor, Besonderheiten des Z 80, Gate Array, Video-Controller und Video-Ram, Soundchip, Schnittstellen, Betriebssystem, Routinennutzung, Character-Generator, BASIC-Interpreter und mehr.

Brückmann/Englisch/Gerits — CPC 464 Intern mit kommentiertem ROM-Listing
548 Seiten, DM 69,— ISBN 3-89011-080-0



Wer die wichtigen Peeks und Pokes zum CPC 464 kennen und anwenden will, der findet hier umfassende Information. Sie reicht vom Adreßbereich des Prozessors über Betriebssystem und Interpreter bis hin zur Einführung in die Maschinensprache. Dazu präzise Programmierhilfen, sinnvolle Routinen sowie reichlich Material zu den Themen Grafikfunktionen, Massenspeicherung und Peripherie, Tricks und Formeln in BASIC, RAM-Pages.

Liesert/Schieb
Peeks & Pokes zum CPC 464
180 Seiten, DM 29,— ISBN 3-89011-092-4

Von den Grundlagen der Maschinenspracheprogrammierung über die Arbeitsweise des Z-80-Prozessors und einer genauen Beschreibung seiner Befehle bis zur Benutzung von Systemroutinen ist alles ausführlich und mit vielen Beispielen erklärt. Im Buch enthalten sind Assembler, Disassembler und Monitor als komplette Anwenderprogramme. So wird der Einstieg in die Maschinensprache leichtgemacht!

Dullin/Straßenburg
Das Maschinensprachebuch zum CPC 464
330 Seiten, DM 39,— ISBN 3-89011-070-3

Speziell für den Hobbyelektroniker, der mehr aus seinem CPC 464 machen möchte. Von nützlichen Tips zur Platinenherstellung über Adreßdecodierung, Adapterkarten und Interfaces bis zu EPROM-Programmierboard und – Programmiernetzteil oder Motorsteuerung für Gleich- und Schrittschaltmotoren werden machbare Erweiterungen ausführlich und praxisnah beschrieben. Am besten gleich anfangen!

Schüssler
CPC 464 Hardware-Erweiterungen
445 Seiten, DM 49,— ISBN 3-89011-083-5

Ein Führer in die phantastische Welt der Abenteuer-Spiele! Hier wird gezeigt, wie Adventures funktionieren, wie man sie erfolgreich spielt und wie man eigene Adventures auf dem CPC 464 programmiert. Der Clou des Buches ist neben vielen fertigen Adventures ein kompletter ADVENTURE-GENERATOR, mit dem das Selberprogrammieren packender Adventures zum Kinderspiel wird.

Walkowiak — Adventures – und wie man sie auf dem CPC 464 programmiert
320 Seiten, DM 39,— ISBN 3-89011-088-6