Markt&Technik

Helmut Tischer

Programmentwicklung unter CP/M 2.2 auf dem CPC 464/664

● Beschreibung des CP/M-Systems mit seinen BDOS/BIOS-Routinen ● Verbindung zum BASIC-Betriebssystem und Schnittstellen ● Anwendungsbeispiele für die selbständige Programmentwicklung

Programmentwicklung unter CP/M 2.2 auf dem CPC 464/664

Helmut Tischer

Programmentwicklung unter CP/M 2.2 auf dem CPC 464/664

Beschreibung des CP/M-Systems mit seinen BDOS/BIOS-Routinen.

Verbindung zum BASIC-Betriebssystem und Schnittstellen.

Anwendungsbeispiele für die selbständige Programmentwicklung.

Markt&Technik Verlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Tischer, Helmut:
Programmentwicklung unter CP, M 2.2 auf dem CPC 464, 664 :
Beschreibung d. CP/M-Systems mit seinen BDOS/BIOS-Routinen ;
Verbindung zum BASIC-Betriebssystem u. Schnittstellen ; Anwendungsbeispiele
für d. selbständige Programmentwicklung / Helmut Tischer. –
Haar bei München : Markt-und-Technik-Verlag, 1986.
ISBN 3-89090-209-X

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
89 88 87 86

ISBN 3-89090-209-X

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Kösel, Kempten
Printed in Germany

Inhaltsverzeichnis

Vorwort

Finden Sie nicht auch, daß man jedes CP/M-Programm viel 'professioneller' bedienen kann, als Basic- oder gewöhnliche Maschinenprogramme? Möchten Sie selbst solche Programme entwickeln? Oder möchten Sie einfach nur genauer wissen, wie das CP/M-Betriebssystem funktioniert? Wenn ja, dann sollten Sie sich dieses Buch ansehen.

Als Voraussetzung zum Verständnis sollten Sie schon etwas Erfahrung im Umgang mit CP/M haben. Sie sollten einen groben Überblick über die fundamentalen Möglichkeiten haben, die CP/M bietet: Ansprechen verschiedener Laufwerke und Userbereiche, Löschen und Umbenennen einzelner Dateien, Schreibschützen und so weiter. Nicht verlangt ist aber, daß Sie jeden exotischen CP/M-Befehl bis in alle Einzelheiten kennen. Dafür gibt's schließlich ein Handbuch zum Nachschlagen.

Ein Teil dieses Buches ist eine detaillierte Beschreibung des Grundprinzips des CP/M 2.2-Betriebssystemes. Wenn Sie nur diesen theoretischen Teil verstehen möchten, brauchen Sie nur geringe Erfahrung im Umgang mit CP/M zu haben.

In einem größeren Teil erfahren Sie alles, was Sie beim Selbsterstellen von CP/M-Programmen beachten müssen und welche Hilfen Ihnen schon standardmäßig zur Verfügung stehen. Dann können Sie wirklich professionelle Programme entwickeln.

Wenn Sie jede einzelne Information auch in der Praxis ausnützen und das letzte aus Ihrem Computer herausholen wollen, müssen Sie allerdings die Maschinensprache beherrschen. Dabei ist es gleichgültig, ob Sie Z80- oder 8080-Assembler programmiert haben: Anhand eines Vergleiches Z80/8080 können auch diejenigen die 8080-Assemblersprache erlernen, die bisher nur in Z80-Assembler programmiert haben. Zum Programmieren in Assembler benötigen Sie keinerlei zusätzliche Software: Allein die mit Ihrem Diskettenlaufwerk standardmäßig mitgelieferte Software reicht vollständig aus.

Viele Informationen können Sie aber auch ohne Maschinensprachekenntnisse ausnützen. Für einfachere Experimente oder wenn Sie Ihre Anwendungsprogramme nur ganz allgemein besser an die Eigenheiten des CP/M-Betriebssystemes anpassen wollen, genügt es auch, wenn Sie nur eine unter CP/M einsetzbare höhere Programmiersprache wie zum Beispiel Pascal beherrschen.

Da auf den Schneider-Computern das volle Standard-CP/M 2.2 zur Verfügung steht, werden die Möglichkeiten dieses CP/M 2.2-Betriebssystemes beschrieben. Nicht behandelt werden die Ergänzungen von CP/M Plus, da das den Rahmen dieses Buches sprengen würde.

Zwei gesonderte Kapitel behandeln ausführlich die zusätzlichen und normalerweise nicht ausgenutzten Möglichkeiten zur Diskettensteuerung und Systemverwaltung, die es unter CP/M nur auf den Schneider-Computern gibt. Dabei wird nicht nur auf die Schneider 3"-Floppydisk eingegangen, sondern es werden auch die Eigenheiten der Vortex-5 1/4"-Laufwerke berücksichtigt.

Im Kapitel 12 sind einige in der Praxis nützliche CP/M-Programme abgedruckt. Außer dem Assemblerlisting ist dort zu jedem Programm auch ein passender Basic-Lader zu sehen. Die Programme nützen Ihnen also auch, wenn Sie Maschinensprache nicht beherrschen.

Besonders genau sollten Sie sich das Kapitel 1 des Buches durchlesen. Hier wird kurz erklärt, was Sie beim Arbeiten mit CP/M *niemals* falsch machen dürfen, wenn Sie sich nicht in Gefahr begeben wollen, einige Ihrere schon eingegebenen Daten zu verlieren. Das ist schon für einen Nur-Anwender wichtig, also gilt es für Sie als Selbstprogrammierer im Besonderen.

Viel Spaß beim Lesen!

Helmut Tischer, Dezember 1985

1 Die Todsünden

Normalerweise bietet das CP/M 2.2-Betriebssystem einen guten Schutz gegen Fehlbedienungen. Es gibt aber bestimmte Verfahrensweisen (die der typische Benutzer im Falle des Falles 'selbstverständlich' unbewußt anwendet), die selbst den ausgeklügeltsten Schutz überlisten.

Wenn Sie als Programmierer derartige Fehler machen, stehen Sie vor der schwierigen Aufgabe, festzustellen, ob der aufgetretene Fehler wirklich im getesteten Programm zu suchen ist, oder ob er 'nur' die verspätete Folge eines schon früher gemachten schlimmen Bedienungsfehlers ist. Die einzige Möglichkeit, sich dabei einen Haufen Ärger zu ersparen ist, unnötige Bedienungsfehler möglichst zu vermeiden.

Auch wenn Sie schon etwas Erfahrung im Umgang mit CP/M haben, sollten Sie sich die folgenden Punkte durchlesen. Denn es ist sicher einer darunter, den Sie bisher in der entsprechenden Situation 'immer' gemacht haben, und der eben nur bei der reinen Anwendung eines Programms noch glimpflich abgeht.

Allgemeines zum Umgang mit Disketten

3"-Disketten sind wegen ihrer Hardcoverbox vor mechanischen Beschädigungen sehr gut geschützt. Natürlich dürfen Sie sie nicht zum Unterlegen eines wackligen Stuhlbeines verwenden, aber ansonsten vertragen sie auch mal eine etwas robustere Behandlung.

Empfindlich sind die Disketten aber gegen direktes Sonnenlicht, Staub und Feuchtigkeit sowie starke magnetische Felder.

- Der Schutz vor Sonnenlicht ist am einfachsten: Gewöhnen Sie es sich an, Ihre Disketten nach getaner Arbeit nicht mehr auf dem Schreibtisch liegen zu lassen, wenn dieser irgendwann am Tag von der Sonne beschienen wird.

- Um Ihre Disketten vor Staub und Feuchtigkeit zu schützen, empfehle ich Ihnen die Anschaffung einer sogenannten Disketten-Archiv-Box. Das ist eine einmalige Ausgabe von etwa 30 DM und Sie haben dann genug Raum um alle Disketten, die Sie jemals besitzen werden, nach allen Regeln der Kunst aufzubewahren.

- Selbstverständlich dürfen Sie niemals etwa Ihre Tasse Kaffee über eine Diskette schütten. Sollte Ihnen dieses Malheur trotzdem passieren, dann

dürfen Sie nicht versuchen, ob sich die betroffene Diskette nicht doch noch verwenden läßt! In diesem Fall würde die verschmutzte Diskette auch noch Ihr Laufwerk beschädigen! Es ist besser, Sie opfern (wenn auch schweren Herzens) die Diskette und werfen sie unbesehen weg.

- Eine große Gefahr für Disketten sind auch starke Magnetfelder, da diese Daten auf der Diskette verfälschen können. Ein starkes Magnetfeld befindet sich in der unmittelbaren Nähe Ihres Computers: die Ablenkung des Elektronenstrahls im Bildschirm wird von starken Elektromagneten erledigt. Doch auch eingeschaltete Lautsprecherboxen, Transformatoren, Elektromotoren und sogar Scheren und Schraubenzieher sind Quellen von Magnetfeldern. Sie sollten Ihre Disketten von solchen Gegenständen fernhalten.

  Auch beim Ein- und Ausschalten des Diskettenlaufwerks erzeugt der Schreib-/Lesekopf des Laufwerks kurzzeitig ein starkes Magnetfeld. Dieses ist zwar nur räumlich begrenzt, reicht aber gelegentlich trotzdem dazu aus, die direkt darunterliegenden Stellen der Diskette zu löschen. Nehmen Sie also vor dem Ein- und Ausschalten des Laufwerks immer die Diskette heraus.

5 1/4"-Disketten sind wesentlich weniger stabil als 3"-Disketten, und müssen deshalb von Ihnen wie rohe Eier behandelt werden. Außer den Vorsichtsmaßnahmen für die 3"-Disketten sollten Sie deshalb folgendes beachten, wenn Sie solche verwenden:

- Berühren Sie niemals die frei zugänglichen Stellen der Magnetscheibe. Die dadurch auf die Diskette kommende Verschmutzung überträgt sich bei der Benutzung auf den Tonkopf des Laufwerks und verschlechtert dessen Qualität. Kommt das öfters vor, können sie irgendwann Ihre Disketten nicht mehr lesen. Der beste Schutz dagegen ist, die Diskette sofort nach der Entnahme aus dem Laufwerk wieder in den Papierumschlag zu schieben.

- Vermeiden Sie es, Ihre Disketten zu verbiegen. Wird die Diskette schräg gelagert, genügt bereits Ihr Eigengewicht, daß sie im Laufe der Zeit in der Mitte 'durchhängt' und so eine dauerhafte Biegung eingeprägt wird. Lagern Sie deshalb Ihre Disketten immer *genau* senkrecht stehend oder (weniger gut) *genau* waagrecht liegend. Wenn Sie Ihre Disketten waagrecht lagern, sollten Sie außerdem darauf achten, nicht zuviele übereinander zu stapeln.

- Legen Sie niemals schwere Gegenstände auf die Diskette. Bereits ein hochkant daraufgestelltes Buch genügt, daß zumindest die Magnet-

scheibe unter der Buchkante einen Knick erhält, so daß Sie sich nicht mehr ordnungsgemäß drehen kann.

Gelegentlich Backup machen

Auch wenn Sie alle Tips über den Umgang mit Disketten beherzigen, kann es gelegentlich vorkommen, daß sich irgendeine Datei nicht mehr lesen läßt. Dabei spielen allerdings weniger Beschädigungen der Disketten eine Rolle (was, wenn Sie die Disketten ordnungsgemäß behandeln, ziemlich selten vorkommt), sondern eher Bedienungsfehler.

Besonders als Selbstprogrammierer werden Sie öfter erleben, daß ein gerade entwickeltes Programm nicht an die Position auf der Diskette schreibt, die Sie sich vorgestellt haben, sondern irgendwohin. Vor solchen Fehlern schützt nicht einmal ein Schreibschutz der Datei. Wenn Sie dann von der zufällig an der betroffenen Stelle stehenden Datei keine Sicherheitskopie gemacht haben, haben Sie Pech gehabt.

Auch wenn Sie zum Beispiel einen langen Programmtext bearbeiten, sollten Sie auch (und gerade) im größten Stress diesen gelegentlich auf Diskette abspeichern. Nur so vermeiden Sie die Gefahr, bei einem Stromausfall die Arbeit von Stunden zu verlieren.

CP/M nicht ohne eingelegte Diskette betreiben

Achten Sie darauf, daß sich im Laufwerk A immer eine Diskette mit Systemspur befindet. Es kann zwar durchaus sein, daß das verwendete Programm eine Zeitlang ohne Diskettenzugriff auskommt. Nach einiger Zeit vergessen Sie aber im Eifer des Gefechts leicht, daß noch keine Diskette im Laufwerk liegt. Läuft erst einmal der Motor des Laufwerks, ist die letzte Gelegenheit, die Diskette nachträglich einzulegen, verstrichen. Es hilft nur noch das Ausschalten des Rechners.

Wenn Sie einen sonstigen Fehler machen, gibt es immer noch Tricks, das Ergebnis Ihrer bisherigen Arbeit zu retten - wenn nur eine Diskette im Laufwerk liegt.

Neue Disketten sofort formatieren

Gewöhnen Sie es sich an, alle neugekauften Disketten sofort zu formatieren. Öfter als Sie denken werden Sie in die Situation kommen, daß Ihre Arbeitsdisketten voll sind. Und sehr leicht verwechseln Sie in solchen Situationen eine schon formatierte Leerdiskette, die Sie dann schnell einlegen wollen, mit einer noch nicht formatierten - und das ist genauso fatal, wie das Betreiben von CP/M ganz ohne Diskette.

Von der ROM-Software des 3"-Diskettenlaufwerkes werden 4 Diskettenformate unterstützt. Unter CP/M muß aber bis auf wenige Ausnahmefälle (beim Kopieren mit Filecopy und beim Formatieren) im Laufwerk A immer eine Diskette mit dem Systemformat liegen.

Ich empfehle Ihnen deshalb, alle Disketten im Systemformat zu formatieren, selbst wenn Sie vorhaben, diese nur im Basic-System einzusetzen. Denn

1. wird dadurch die Gefahr des Verwechseln einer Datendiskette mit einer Systemdiskette vermieden - was einen Systemabsturz verursachen würde;

2. werden Sie sehr schnell feststellen, wie nützlich es ist, wenn Sie aus dem Basic ganz schnell auf CP/M umschalten und mit FILECOPY Dateien kopieren oder ohne umständliche @Q$-Sequenzen umbenennen und löschen können.

Gegenüber diesen Vorteilen fällt der Nachteil, daß eine Diskette im Systemformat 9 KByte weniger Speicherplatz als eine Datendiskette bietet, gar nicht mehr ins Gewicht.

Jeden Diskettenwechsel anmelden

Um eine vernünftige Arbeitsgeschwindigkeit zu erreichen, holt sich das CP/M-Betriebssystem nicht bei jedem Diskettenzugriff alle Informationen über die belegten und freien Bereiche der betroffenen Diskette. Das hat jedoch die unangenehme Konsequenz, daß die Diskette nicht zu jedem beliebigen Zeitpunkt gewechselt werden darf: Es kann sonst vorkommen, daß auf der neuen Diskette wichtige Daten überschrieben werden, weil auf der alten Diskette an der gleichen Stelle freier Speicherplatz war.

Wenn Sie derartige Vorfälle vermeiden wollen, dann dürfen Sie die Disketten nur in den folgenden Situationen und nur zusammen mit einer entsprechenden Mitteilung an das Betriebssystem wechseln:

1. Wenn die Bereitschaftsmeldung des CP/M-Betriebssystems erscheint und Sie die Diskette im Bezugslaufwerk wechseln wollen, dann dürfen Sie das tun. Also etwa wenn die Meldung 'A>' erscheint und Sie die Diskette im Laufwerk A wechseln wollen. In diesem Fall müssen Sie aber *nachdem* Sie die Diskette gewechselt haben ^C am Anfang der Befehlszeile eingeben.

2. Wenn die Bereitschaftsmeldung des CP/M-Betriebssystem erscheint und A das Bezugslaufwerk ist, müssen Sie das Wechseln der Diskette im Laufwerk B *vor oder nach* dem Wechseln mit ^C anmelden.

Manche Programme führen eine solche Anmeldung automatisch aus, bevor sie abgebrochen werden. In den meisten Fällen erkennt man das daran, daß das Diskettenlaufwerk unmittelbar bevor die Meldung 'A>' oder 'B>' erscheint nochmal kurz anläuft. In diesem Fall können Sie sich also das ^C sparen.

3. Wenn ein gerade laufendes Programm durch eine Meldung wie zum Beispiel:

```
Bitte die Datendiskette in Laufwerk A einlegen
```

ausdrücklich einen Diskettenwechsel anfordert, dürfen Sie ohne Bedenken die Diskette wechseln. Der Programmierer wird dann schon dafür Sorge getragen haben, daß der Wechsel dem Betriebssystem richtig mitgeteilt wird.

4. Manche Programme erlauben in besonderen Fällen einen Wechsel der Disketten, ohne daß der Wechsel ausdrücklich angefordert wird. Im Programm WordStar dürfen zum Beispiel die Disketten genau dann gewechselt werden, wenn das Startmenü erscheint.

   Im Programm PIP dürfen die Disketten unmittelbar vor dem Programmabbruch durch ENTER gewechselt werden, wenn die Bereitschaftsmeldung '*' erscheint, weil bei diesem Programmabbruch automatisch ein ^C ausgeführt wird. Solche Fälle sind aber sehr selten und dürfen nur dann ausgenützt werden, wenn sie ausdrücklich im Handbuch erwähnt werden.

5. Wenn Sie sicher sind, daß auf die neue Diskette nicht geschrieben, sie also nur gelesen wird, dürfen Sie die Diskette ohne besondere Vorkehrungen wechseln. Das aber nur, wenn Sie sicher sind, daß noch keine Datei oder keine Datei der alten Diskette mehr in Bearbeitung ist.

   Wenn Sie auf Nummer sicher gehen wollen, dann sollten Sie nach jedem Diskettenwechsel ^C eingeben - das schadet nie. Wenn Sie also keine Lust haben sich alles gerade gesagte zu merken, dann sollten Sie sich zumindest das zur festen Gewohnheit machen. Es erspart Ihnen viel Ärger.

Programm immer nach Vorschrift beenden

Beenden Sie Programme niemals nur durch Ausschalten des Rechners (etwa weil Sie sowieso gerade mit dem Arbeiten fertig sind) oder durch vorzeitigen Abbruch mit ^C!

Komplexere Programme legen nämlich Ihre Eingaben üblicherweise in verschiedenene Zwischendateien ab. Erst wenn Sie ordnungsgemäß das Programmende anfordern, werden diese in die endgültige Zieldatei umgespeichert und die Zwischendateien gelöscht. Zu bestimmten Zeitpunkten stehen auf der Diskette sogar nur die neuen Daten, im Directory ist aber dann Ihre Position nicht vermerkt.

Wenn Sie ein Anwenderprogramm in solchen Situationen unterbrechen, ist Ihre ganze Arbeit, die Sie in dieser Sitzung erledigt haben, verloren.

Erlaubte Dateinamen

Nicht alle Zeichen sind in Dateinamen erlaubt. Als Nur-Anwender von CP/M-Programmen brauchen Sie aber nicht über jedes einzelne verbotene Zeichen Bescheid zu wissen: In der Regel findet in den Anwenderprogrammen eine Kontrolle der Eingabe statt.

Anders ist es jedoch, wenn Sie selbst Programme entwickeln: Dann findet keine automatische Eingabekontrolle statt und Sie können ohne weiteres eine Datei mit einem ungültigem Dateinamen erzeugen. Eine solche Datei können Sie unter normalen Umständen weder verwenden noch ändern oder löschen - es sei denn mit Hilfe eines 'Gegenprogrammes'. Bei einem REN- oder ERA-Befehl wird ja die Eingabe wieder auf Gültigkeit getestet und so die Ausführung mit einem ungültigem Dateinamen verhindert.

Folgende Zeichen dürfen in Dateinamen nicht verwendet werden:

- Bekannt dürfte sein, daß die Zeichen '*' und '?' in Dateinamen nicht verwenden werden dürfen. Wenn Sie etwa mit einem selbstentwickelten CP/M-Programm eine Datei mit dem Namen A*.* erstellen würden, würde der Befehl 'ERA A*.*' ja nicht nur die Datei mit diesem Namen löschen, sondern alle Dateien, die mit dem Buchstaben A beginnen.

- Ebenfalls nicht verwenden sollten Sie Kleinbuchstaben. Bei einer normalen Eingabe werden alle Kleinbuchstaben sofort in Großbuchstaben umgewandelt, so daß Sie niemals die Chance haben, wirklich eine Datei mit Kleinbuchstaben im Namen anzusprechen.

- Leerzeichen dürfen nicht in der Mitte von Dateinamen auftreten, da ein Leerzeichen bei einer normalen Eingabe als das Ende des Namens interpretiert würde. Beim Versuch, die Datei 'AB CD.TXT' anzusprechen, würde also immer die Datei 'AB.' (ohne Namensextension!) aufgerufen werden.

- In allen Dateinamen verboten sind Steuerzeichen mit Codes zwischen 00h und 1Fh, das DEL-Zeichen (Code 7Fh) sowie die Sonderzeichen

  ```
  < > . , ; : = [ ] _
  ```

  Bei Verwendung eines deutschen Zeichensatzes werden '[' und ']' durch 'Ä' und 'Ü' ersetzt.

- Für Dateien, die auch unter dem Basic-Betriebssystem verwendet werden sollen sind (nicht dem CP/M-Standard entsprechend) einige weitere Zeichen verboten. Wenn Sie also nicht ganz sicher sind, daß Sie die zu erstellende Datei nur unter dem CP/M-Betriebssystem benötigen, dann sollten Sie auch die Zeichen

  ```
  | \ ( ) % /
  ```

  vermeiden. Statt '|' und '\' sind natürlich die Zeichen 'ö' und 'Ö' verboten, wenn Sie einen deutschen Zeichensatz verwenden. Im CP/M-Betriebssystem sind diese Zeichen aber erlaubt.

- Bei der Vortex-Floppydisk sind im Basic-Betriebssystem auch die Zeichen '{' und '}' beziehungsweise ihre deutschen Äquivalente 'ä' und 'ü' in Dateinamen verboten.

In der Regel ist es empfehlenswert, sich an diese Beschränkungen zu halten. Wenn Sie unbedingt wollen, können Sie diese Beschränkungen aber auch gezielt durchbrechen, um zum Beispiel (für den durchschnittlichen Anwender) lese- und kopiergeschützte Dateien zu erstellen.

2 Was ist ein CP/M-Programm?

Wie Sie sicher wissen, sind die umfangreicheren CP/M-Befehle auf Ihrer Systemdiskette in Dateien mit der Namensextension .COM abgespeichert und werden nur bei Bedarf in das RAM Ihres Computers geladen.

Der Befehl STAT funktioniert zum Beispiel nur, wenn sich auf der im Laufwerk liegenden Diskette die Datei STAT.COM befindet. Trotzdem wird STAT genauso aufgerufen, als ob es sich dabei um einen gewöhnlichen Befehl wie etwa DIR handeln würde: Durch Eingabe von 'NAME PARAMETER1 PARAMETER2 ...'.

Welche Befehle das CP/M-Betriebssystem versteht, hängt also von den Namen der COM-Dateien ab, die sich auf der eingelegten Diskette befinden. Sie können sich also ein 'System nach Maß' zusammenstellen, indem Sie eine gewisse Auswahl der COM-Dateien auf der CP/M-Systemdiskette auf Ihre Arbeitsdiskette kopieren.

Genausogut ist es aber auch möglich, weitere Befehle von Softwarefirmen zusätzlich zu erwerben. Diese zusätzlich erworbenen Befehle sind die eigentlichen Programme, die es für das CP/M-Betriebssystem gibt.

Das Entwickeln von CP/M-Programmen bedeutet also, COM-Dateien zu erstellen. 'Reine' CP/M-Programme sind dabei einer Reihe von zusätzlichen Normen unterworfen, so daß diese auf jedem beliebigen CP/M-Rechner jeder x-beliebigen Firma lauffähig sind. Ein Beispiel dafür ist das STAT- oder das PIP-Programm. Weniger 'reine' CP/M-Prgramme benötigen gewisse Hardware-Voraussetzungen. Ihr FORMAT-Befehl etwa funktioniert nur zusammen mit einem ganz bestimmten Floppydisk-Typ. Trotzdem ist der FORMAT-Befehl aber noch eine COM-Datei.

Eine COM-Datei ist nichts anderes als ein 8080- oder Z80-Maschinenspracheprogramm, das in einer besonderen Art und Weise auf Diskette abgespeichert wurde. Eine andere Möglichkeit COM-Dateien zu erstellen gibt es nicht, denn das CP/M-Betriebssystem versteht nur Maschinensprache. Selbst Compiler, die es erlauben COM-Dateien scheinbar in einer Hochsprache wie Pascal zu schreiben, bilden keine Ausnahme. In Wirklichkeit nehmen diese dem Anwender nur das Übersetzen der Hochsprachenanweisungen ab, so daß am Ende wieder nur ein besonderes Maschinenspracheprogramm entsteht, das in einer COM-Datei abgespeichert wird.

Möchten Sie also nicht in einer Hochsprache programmieren und damit auf einen großen Teil der Möglichkeiten, die das CP/M-Betriebssystem bietet verzichten, ist es für Sie unerläßlich, sich mit der Z80- oder 8080-Maschinensprache auseinanderzusetzen.

Ob Sie sich dabei mehr auf die Z80- oder mehr auf die 8080-Maschinensprache konzentrieren bleibt Ihrem ganz persönlichen Geschmack überlassen. Die Z80-CPU, die in den Schneider Computern eingebaut ist, beherrscht ja auch alle Befehle des 8080-Prozessors. Umgekehrt beherrscht der 8080-Prozessor aber nicht alle Befehle des Z80-Prozessors.

Wenn Sie Maschinensprache nicht verstehen, können Sie die nächsten beiden Kapitel ohne Schaden überspringen. Die meisten Informationen in späteren Kapiteln werden Ihnen auch nützen, wenn Sie nur in einer höheren Programmiersprache wie Pascal arbeiten.

3 Die Maschinensprachen des CPC 464/664

3.1 Die Z80-Maschinensprache

Voraussetzungen zum Verständnis

Die detaillierte Erklärung der allgemeinen Funktionsweise von Maschinensprachen sowie aller 698 Z80-Befehle wäre so umfangreich, daß sie den Rahmen dieses Buches sprengen würde. Wenn Sie alle Informationen dieses Buches in der Praxis ausprobieren möchten, sollten Sie die Z80-Maschinensprache also bereits beherrschen. Zu diesem Thema gibt es aber genügend Sekundärliteratur, so daß es Ihnen nicht schwer fallen dürfte, etwas Passendes auszuwählen.

Im folgenden sind deshalb nur die Beschränkungen beschrieben, die es nur auf den Schneider-Computern gibt, und die deshalb natürlich nicht in Standardwerken zur Z80-CPU aufgeführt sind.

Verbotene Z80-Befehle auf dem CPC 464/664

Der alternative Registersatz

In 'normalen' Maschinenprogrammen darf der alternative Registersatz, also die Register af', bc', de' und hl' nicht verwendet werden. Im CP/M-Betriebssystem ist das - entgegen der weitverbreiteten Meinung - doch erlaubt. Nicht einmal den Interrupt müssen Sie vor der Verwendung der Register abschalten. Wenn Sie reine Standard-CP/M-Programme entwickeln wollen, gibt es für Sie also keinerlei Einschränkungen.

Vorsicht ist allerdings geboten, wenn Sie Firmwareroutinen Ihres Schneider-Computers verwenden möchten. Vor dem Aufruf dieser Routinen muß das bc'-Register und das Carry'-Flag auf einen definierten Wert gesetzt werden. Die genaue Bedeutung dieses Wertes können sie in weiterführender Literatur über das Innenleben Ihres Schneiders nachlesen. Es genügt aber vorläufig, wenn Sie sich merken, daß das das Carry'-Flag vor einem Firmwareaufruf gelöscht und das bc'-Register auf den Wert 7F86h gesetzt werden muß, wenn sich der Bildschirm im Modus 2 befindet. Im Modus 1 heißt der Wert 7F85h und im Modus 0 7F84h. Eine Firmwareroutine können sie also etwa so starten:

```
and  a
ex   af,af'
ld   bc,7F86h
exx
call Firmware
exx
ex   af,af'
```

Nur wenn Sie sicher sind, daß das bc'-Registerpaar nach einem Kaltstart nicht mehr verändert wurde, dürfen Sie diese Prozedur weglassen. Selbstverständlich müssen Sie zusätzlich die Register retten, die von der Routine verändert werden und deren Inhalte Sie noch benötigen.

Achten sie darauf, daß nach Abschluß einer Firmwareroutine das bc'-Register denselben Wert hat wie vor dem Aufruf. Eine Zerstörung der alten Werte bewirken die Firmwareroutinen zur ROM-Umschaltung und die Bildschirmmodusumschaltung. Die entsprechenden Firmwareroutinen sind also verboten. Die einzige Ausnahme ist der Mode-Befehl, der durch aussenden des Steuerzeichens mit dem Code 4 an den Bildschirm simuliert wird. Nach einer solchen Umschaltung muß aber der geänderte Standardwert für das bc'-Register beachtet werden.

Wenn Sie diese Vorsichtsmaßnahmen nicht beachten, können Sie fast sicher sein, daß Ihr CPC-Computer bei einem Aufruf einer Firmwareroutine abstürzt.

Interrupts

Den Interrupt dürfen Sie prinzipiell zu jeder Zeit abschalten, wenn es für bestimmte Aufgaben nötig ist. Für die Dauer der Abschaltung unterbleibt die Ausführung aller zeitabhängigen Vorgänge. Insbesondere wird die Tastatur solange nicht abgefragt, bis der Interrupt wieder eingeschaltet ist. Vergißt man das Wiedereinschalten des Interrupts, reagiert der Computer auf keinerlei Eingaben mehr und es hilft nur noch das Abschalten der Stromversorgung. Sie sollten also immer darauf achten, daß der Interrupt nicht zu lange abgeschaltet bleibt.

Erlaubte Input/Output-Befehle

Im CPC 464/664 werden zu Adressierung der Ein-/Ausgabegeräte nicht - wie allgemein üblich - die 8 niederwertigen Bits des Adreßbusses verwendet, sondern die 8 höherwertigen. Nicht alle Z80-Input/Output-Befehle legen aber die 8 höherwertigen Bits auf einen definierten und konstanten Wert. Es dürfen deshalb nur die folgenden I/O-Befehle verwendet werden:

`OUT (C),r` wobei 'r' für die Register A,B,C,D,E,H,L steht.
Die Adresse des Ausgabegerätes steht jedoch hier nicht wie normalerweise im C-Register, sondern im B-Register. Der Wert des C-Registers ist beliebig.

`IN r,(C)` wobei 'r' für die Register A,B,C,D,E,H,L steht.
Das Ausgabegerät wird genauso adressiert, wie bei 'OUT (C),r' beschrieben.

`IN A,(n)` wobei n für einen 8-Bit-Wert steht.
Die Adresse des Eingabegerätes muß bei der Verwendung im CPC 464/664 im A-Register stehen! Der Wert von 'n' ist beliebig. Beachten Sie bitte, daß die Adresse im Register A nach einem Leseversuch zerstört ist.

`INI, IND, OUTI, OUTD`
Beachten Sie, daß diese Befehle den Wert im Register B und damit die Ein-/Ausgabeadresse verändern. Vor einer erneuten Verwendung muß deshalb das B-Register restauriert werden.

Alle anderen Z80-Ein-/Ausgabebefehle sind im CPC 464/664 verboten. Der Befehl 'OUT (n),A', weil hier der Wert des höherwertigen Adreßbytes nicht definiert ist und die anderen Blockein-/ausgabebefehle, weil das B-Register ohne Korrekturmöglichkeit verändert wird.

Anwendbarkeit

Es ist nicht verboten, die Ein-/Ausgabebefehle in einem CP/M-Programm zu verwenden. Machen Sie sich aber bewußt, daß dieses Programm dann nur auf dem CPC 464/664 ablauffähig ist, da es ja direkt auf die Hardware des Rechners zugreift.

Überlegen Sie es sich sehr genau, ob der Vorteil, den der Einsatz dieser Befehle bringt, diese Einschränkung wert ist. Wenn das Programm allerdings mit Sicherheit nur für Ihren persönlichen Bedarf gedacht ist und Sie es nicht verkaufen wollen, dann brauchen Sie sich über diesen Punkt keine Gedanken zu machen.

3.2 Die 8080-Maschinensprache

Allgemeines zur 8080-CPU

Die 8080 CPU wurde Anfang der 70'er Jahre von der Firma Intel entwikkelt und wurde einige Zeit lang sehr häufig verwendet. Als 1975 bei der Firma Zilog die Z80-CPU entwickelt wurde, wollte man es ermöglichen, die schon in großen Mengen vorhandene 8080-Software weiter zu verwenden. Die Z80-CPU wurde deshalb als Nachfolger der 8080-CPU konstruiert: Alle Befehle und Register der 8080-CPU gibt es auch in der Z80-CPU. Lediglich die mnemonischen Bezeichnungen der Z80-Befehle wurden geändert. 8080- und Z80-Befehle, die dieselbe Funktion haben, werden aber trotzdem zum selben Objektcode übersetzt. Es kann also wirklich jedes 8080-Programm unverändert auf einem Z80-Computer weiterverwendet werden.

Das CP/M-Betriebssystem wurde deshalb ebenfalls vollständig in 8080-Maschinencode entwickelt, um sowohl auf den älteren 8080-Maschinen, als auch auf den neueren Z80-Maschinen verwendbar zu sein. Wenn Sie also Programme entwickeln wollen, die voll dem CP/M-Standard entsprechen, dann sollten Sie sich unbedingt an die 8080-Programmiersprache gewöhnen. Außerdem ersparen Sie sich dann die Anschaffung eines teuren Z80-Assemblers - denn auf Ihrer CP/M-Systemdiskette befindet sich bereits ein 8080-Assembler.

Im folgenden wird die 8080-CPU anhand eines Vergleichs zur Z80-CPU erklärt, so daß Sie leicht umlernen können.

Die Register der 8080-CPU

In der 8080-CPU fehlt der alternative Registersatz (die 'gestrichenen' Register), die Indexregister, das Interrupt- und das Refreshregister. Zur freien Verfügung für den Benutzer stehen nur noch die folgenden 8-Bit-Register:

`f` Das Flag-Register. Wenn es im PUSH- oder POP-Befehl zusammen mit dem A-Register angesprochen wird, nennt man dieses Paar bei der 8080-Programmierung jedoch nicht mehr 'AF-Registerpaar', sondern 'PSW-Register'.

`a` Das Universalregister.

`b, c` Auch in der 8080-CPU werden diese beiden Register von bestimmten Befehlen als ein einziges 16-Bit-Register betrachtet. In den entsprechenden mnemonischen Befehlen

wird das Paar jedoch nicht als 'BC-Register' bezeichnet, sondern weiterhin als 'B-Register'.

`d, e` Das Paar, das diese beiden Register bilden, wird in den betreffenden Befehlen ebenfalls nicht 'DE' sondern nur 'D' genannt.

`h, l` Werden diese beiden Register zu einem Paar gekoppelt, bilden sie auch bei der 8080-CPU das Universalregister für 16-Bit-Additionen. 8080-Befehle, die direkt mit dem 16-Bit-Wert operieren, nennen dieses Paar äquivalent zum BC- und DE-Registerpaar ebenfalls 'H-Registerpaar'. Befehle mit indirekter Adressierung, die das hl-Registerpaar als Zeiger auf eine Speicherzelle betrachten, nennen es aber 'M', zum Beispiel 'MOV A,M' statt 'MOV A,(HL)'.

`pc, sp` Die Bedeutungen des Stackpointers und des Programmcounters in der 8080-CPU sind analog zu denen der Z80-CPU.

Die Befehle der 8080-CPU

Die 8080-CPU versteht nur Befehle, die einen 1-Byte-Opcode besitzen. Im Vergleich zur Z80-CPU fehlen also zunächst:

- alle Operationen, die indizierte Adressierung verwenden (es gibt ja sowieso keine Indexregister),
- die Blocktransferbefehle,
- einige 16-Bit-Lade- und arithmetische Befehle,
- die Ein-/Ausgabebefehle mit indirekter Adressierung,
- Befehle zur detaillierten Steuerung der Interruptmöglichkeiten: Eine 8080-CPU befindet sich immer im Interruptmodus 1.
- Alle Einzelbit- und die meisten Schiebeoperationen. Auf einer 8080-CPU gibt es nur die Rotationsbefehle RLCA, RLA, RRCA und RRA.

Doch auch einige 1-Byte Objektcodes gibt es auf der 8080-CPU nicht:

- Selbstverständlich fehlen die Befehle "EX AF,AF'" und EXX. Diese Befehle sind auch bei der 8080-Programmierung gar nicht sinnvoll, da es ja den alternativen Registersatz nicht gibt.
- Leider ist es bei der 8080-Programmierung unmöglich, relative Sprünge zu verwenden. Die Befehle DJNZe, JRe und 'JRcc,e' fehlen nämlich. Damit ist es ohne Tricks völlig unmöglich, frei verschiebbare 8080-Programme zu erstellen. Das ist eigentlich die schlimmste Beschränkung, die die 8080-CPU dem Z80-gewohnten Programmierer auferlegt.

Objectcode: Höherwertiges Nibble

Niederwertiges Nibble	0	1	2	3	4	5	6	7
0	NOP NOP				LD B,B MOV B,B	LD D,B MOV D,B	LD H,B MOV H,B	LD (HL),B MOV M,B
1	LD BC,nn LXI B,nn	LD BC,nn LXI D,nn	LD HL,nn LXI H,nn	LD SP,nn LXI SP,nn	LD B,C MOV B,C	LD D,C MOV D,C	LD H,C MOV H,C	LD (HL),C MOV M,C
2	LD (BC),A STAX B	LD (DE),A STAX D	LD (nn),HL SHLD nn	LD (nn),A STA n	LD B,D MOV B,D	LD D,D MOV D,D	LD H,D MOV H,D	LD (HL),D MOV M,D
3	INC BC INX B	INC DE INX D	INC HL INX H	INC SP INX SP	LD B,E MOV B,E	LD D,E MOV D,E	LD H,E MOV H,E	LD (HL),E MOV M,E
4	INC B INR B	INC D INR D	INC H INR H	INC (HL) INR M	LD B,H MOV B,H	LD D,H MOV D,H	LD H,H MOV H,H	LD (HL),H MOV M,H
5	DEC B DCR B	DEC D DCR D	DEC H DCR H	DEC (HL) DCR M	LD B,L MOV B,L	LD D,L MOV D,L	LD H,L MOV H,L	LD (HL),L MOV M,L
6	LD B,n MVI B,n	LD D,n MVI D,n	LD H,n MVI H,n	LD (HL),n MVI M,n	LD B,(HL) MOV B,M	LD D,(HL) MOV D,M	LD H,(HL) MOV H,M	HALT HLT
7	RLCA RLC	RLA RAL	DAA DAA	SCF STC	LD B,A MOV B,A	LD D,A MOV D,A	LD H,A MOV H,A	LD (HL),A MOV M,A
8					LD C,B MOV C,B	LD E,B MOV E,B	LD L,B MOV L,B	LD A,B MOV A,B
9	ADD HL,BC DAD B	ADD HL,DE DAD D	ADD HL,HL DAD H	ADD HL,SP DAD SP	LD C,C MOV C,C	LD E,C MOV E,C	LD L,C MOV L,C	LD A,C MOV A,C
A	LD A,(BC) LDAX B	LD A,(DE) LDAX D	LD HL,(nn) LHLD nn	LD A,(nn) LDA nn	LD C,D MOV C,D	LD E,D MOV E,D	LD L,D MOV L,D	LD A,D MOV A,D
B	DEC BC DCX B	DEC DE DCX D	DEC HL DCX H	DEC SP DCX SP	LD C,E MOV C,E	LD E,E MOC E,E	LD L,E MOV L,E	LD A,E MOV A,E
C	INC C INR C	INC E INR E	INC L INR L	INC A INR A	LD C,H MOV C,H	LD E,H MOV E,H	LD L,H MOV L,H	LD A,H MOV A,H
D	DEC C DCR C	DEC E DCR E	DEC L DCR L	DEC A DCR A	LD C,L MOV C,L	LD E,L MOV E,L	LD L,L MOV L,L	LD A,L MOV A,L
E	LD C,n MVI C,n	LD E,n MVI E,n	LD L,n MVI L,n	LD A,n MVI A,n	LD C,(HL) MOV C,M	LD E,(HL) MOV E,M	LD L,(HL) MOV L,M	LD A,(HL) MOV A,M
F	RRCA RRC	RRA RAR	CPL CMA	CCF CMC	LD C,A MOV C,A	LD E,A MOV E,A	LD L,A MOV L,A	LD A,A MOV A,A

Niederwertiges Nibble

Bild 3-1: *Äquivalente Befehle der 8080- und der Z80-CPU. 1. Zeile eines Feldes: Z80-Mnemonic, 2. Zeile: 8080-Mnemonic.*

Bis auf wenige Ausnahmen wurden für den Z80-Assembler alle mnemonischen 8080-Befehle umbenannt, obwohl deren Bedeutung und Objektcode gleich blieb. Im Bild 3-1 sehen Sie deshalb die Zuordnung zwischen Z80- und entsprechendem 8080-Befehl. Wegen der besseren Übersichtlichkeit wurden alle Z80-Befehle, zu denen es keinen entsprechenden 8080-Befehl gibt, weggelassen.

Verwechslungsgefahren zwischen 8080 und Z80

Wenn Sie bisher viel in Z80-Assembler programmiert haben, dann wird es Ihnen früher oder später passieren, daß Sie versehentlich statt einem 8080-Befehl einen Z80-Befehl in Ihr Assemblerprogramm einfügen. Normalerweise ist das nicht weiter tragisch: Beim Übersetzen gibt der Assembler eben eine Fehlermeldung aus und Sie müssen die Zeile korrigieren.

Objektcode: Höherwertiges Nibble

	8	9	A	B	C	D	E	F
0	ADD A,B ADD B	SUB B SUB B	AND B ANA B	OR B ORA B	RET NZ RNZ	RET NC RNC	RET PO RPO	RET P RP
1	ADD A,C ADD C	SUB C SUB C	AND C ANA C	OR C ORA C	POP BC POP B	POP DE POP D	POP HL POP H	POP AF POP PSW
2	ADD A,D ADD D	SUB D SUB D	AND D ANA D	OR D ORA D	JP NZ,nn JNZ nn	JP NC,nn JNC nn	JP PO,nn JPO nn	JP P,nn JP nn
3	ADD A,E ADD E	SUB E SUB E	AND E ANA E	OR E ORA E	JP nn JMP nn	OUT (n),A OUT n	EX (SP),HL XTHL	DI DI
4	ADD A,H ADD H	SUB H SUB H	AND H ANA H	OR H ORA H	CALL NZ,nn CNZ nn	CALL NC,nn CNC nn	CALL PO,nn CPO nn	CALL P,nn CP nn
5	ADD A,L ADD L	SUB L SUB L	AND L ANA L	OR L ORA L	PUSH BC PUSH B	PUSH DE PUSH D	PUSH HL PUSH H	PUSH AF PUSH PSW
6	ADD A,(HL) ADD M	SUB (HL) SUB M	AND (HL) ANA M	OR (HL) ORA M	ADD A,n ADI n	SUB n SUI n	AND n ANI n	OR n ORI n
7	ADD A,A ADD A	SUB A SUB A	AND A ANA A	OR A ORA A	RST 00h RST 0	RST 10h RST 2	RST 20h RST 4	RST 30h RST 6
8	ADC A,B ADC B	SBC A,B SBB B	XOR B XRA B	CP B CMP B	RET Z RZ	RET C RC	RET PE RPE	RET M RM
9	ADC A,C ADC C	SBC A,C SBB C	XOR C XRA C	CP C CMP C	RET RET		JP (HL) PCHL	LD SP,HL SPHL
A	ADC A,D ADC D	SBC A,D SBB D	XOR D XRA D	CP D CMP D	JP Z,nn JZ nn	JP C,nn JC nn	JP PE,nn JPE nn	JP M,nn JM nn
B	ADC A,E ADC E	SBC A,E SBB E	XOR E XRA E	CP E CMP E		IN A,(n) IN n	EX DE,HL XCHG	EI EI
C	ADC A,H ADC H	SBC A,H SBB H	XOR H XRA H	CP H CMP H	CALL Z,nn CZ nn	CALL C,nn CC nn	CALL PE,nn CPE nn	CALL M,nn CM nn
D	ADC A,L ADC L	SBC A,L SBB L	XOR L XRA L	CP L CMP L	CALL nn CALL nn			
E	ADC A,(HL) ADC M	SBC A,(HL) SBB M	XOR (HL) XRA M	CP (HL) CMP M	ADC A,n ACI n	SBC A,n SBI n	XOR n XRI n	CP n CPI n
F	ADC A,A ADC A	SBC A,A SBB A	XOR A XRA A	CP A CMP A	RST 08h RST 1	RST 18h RST 3	RST 28h RST 5	RST 38h RST 7

Bild 3-1: *Fortsetzung*

Davon gibt es jedoch 2 Ausnahmen mit fatalen Folgen!

1. CP n: Bei der Z80-Programmierung hat dieser Befehl die Bedeutung: 'Vergleiche das A-Register mit dem Inhalt der nachfolgenden Speicherzelle'. Der 8080-Assembler übersetzt diesen Befehl aber als: 'Wenn der Wert der letzten Operation positiv war, dann rufe das Unterprogramm an der Stelle n auf'. Richtig müßte der Vergleichsbefehl heißen 'CMP n'.

2. JP n: Der Z80-Assembler liest diesen Befehl als 'Springe unbedingt an die Adresse n'. Der 8080-Assembler liest diesen Befehl jedoch als 'Springe nur, wenn das Ergebnis der letzten Operation positiv war'. Der unbedingt Sprung heißt beim 8080-Assembler 'JMP n'!

Diese beiden Verwechslungen sind sehr heimtückisch. Denn der Assembler übersetzt die Befehle, ohne einen Fehler zu bemerken. Beim manuellen Durchlesen des Listings fällt die Verwechslung ebenfalls kaum auf: Wenn einem ein so schön 'passender' Befehlsname vorgesetzt wird, ist man leicht geneigt, ihm die Aktion zuzuschreiben, die man sich wünscht.

Der beste Schutz vor solchen Fehlern ist: Wenn Sie mit den 8080-Befehlsnamen noch nicht ganz so vertraut sind, dann sehen Sie lieber einmal zu oft in der Tabelle nach, als einmal zu wenig!

4 Hilfsmittel zur Maschienprogrammerstellung

4.1 Der 8080-Assembler

4.1.1 Prinzipielle Arbeitsweise

Der mit dem CP/M 2.2-Betriebssystem mitgelieferte 8080-Assembler hat den Namen ASM.COM. Dem Anfänger erscheint er recht unscheinbar: Er läßt sich nur starten und läuft dann vollständig automatisch ab - ohne daß etwa über ein Menü weitere Eingaben vom Anwender erforderlich wären.

Trotzdem ist seine Leistungsfähigkeit nicht zu unterschätzen: Mit ihm kann alles gemacht werden, was auch mit jedem anderen Assembler möglich ist. Er beherscht symbolische Adressierung, Definition von Konstanten und bedingte Assemblierung mit IF und ENDIF. Sie brauchen sich also nicht mehr schon vor der Programmerstellung überlegen, wo sich die einzelnen Programmteile befinden. Jetzt tippen Sie einfach ein 'CALL UNTERPROGRAMM' und später 'UNTERPROGRAMM: ....'. Für Sie bedeutet das eine wesentliche Arbeitserleichterung.

Die Assembler-Quelldatei kann mit dem ASM-Programm jedoch nicht erstellt werden. Dazu muß man einen separaten Texteditor verwenden. Die Assembler-Quelldatei muß dabei als normalen Standard-Ascii-Text abgefaßt sein. Das ist jedoch kein Nachteil, sondern ein Vorteil: So kann man seine Assemblerprogramme sowohl mit Hilfe des jedem CP/M 2.2-Betriebssystem beiliegenden CP/M-Texteditors ED.COM entwickeln, als auch mit jedem hochentwickelten Textverarbeitungsprogramm. Wenn Sie etwa WordStar besitzen, dann können Sie Ihre Assemblerprogramme ruhig damit erstellen und dessen komfortable Editormöglichkeiten ausnützen.

Das eigentliche Assemblerprogramm übernimmt dann nur noch die Übersetzung dieses Quelltextes in verschiedene - durch Optionen gesteuerte - Zieldateien.

4.1.2 Erstellen einer Assembler-Quelldatei

Grundlegende Forderungen

Damit der Assembler eine Datei überhaupt lesen kann, muß sie folgenden Bedingungen genügen:

1. Außer den Codes 0Dh (^M oder CR), 0Ah (^J oder LF), 1Ah (^Z oder SUB) darf der Quelltext nur druckbare Ascii-Zeichen mit Codes zwischen 20h und 7Eh enthalten. Andere (Steuer-)Zeichen sind verboten und führen dazu, daß der Quelltext nicht korrekt verarbeitet wird.

2. Jede Zeile muß sowohl durch einen Wagenrücklauf (0Dh) als auch durch einen Zeilenvorschub (0Ah) von der vorhergehenden getrennt werden. Ein vollständiger Zeilenübergang besteht also aus 2 Zeichen!

3. Insbesondere muß auch nach der letzten Zeile ein CR/LF-Paar folgen. Zusätzlich muß das Textende durch ^Z (Code 1Ah) markiert werden. Wird diese Maßnahme nicht beachtet, erkennt der Assembler das Textende nicht richtig und verarbeitet die zufälligen, nach dem Textende stehenden Codes als weitere Assemblerzeilen. Das kann unter Umständen zu bösen Resultaten führen.

4. Groß- und Kleinschreibung dürfen Sie dagegen beliebig mischen. Intern wandelt der Assembler vor der eigentlichen Übersetzung alle Klein- in Großbuchstaben um. Selbst symbolische Namen dürfen Sie also einmal groß und das andere mal wieder klein schreiben.

Gewöhnliche Ascii-Texteditoren erfüllen aber diese Bedingungen automatisch ohne daß Sie besonders darauf achten müssen. In manchen der anderen Fälle können Sie durch Umspeichern der Datei mit 'PIP A:= NAME. TXT,EOF[FZ]' eine Endemarkierung anhängen und zumindest die Steuerzeichen zum Seitenvorschub aus dem Text entfernen und die Bits 7 aller Zeichen löschen.

Befehlsformate

Bei einem Assembler-Quelltext muß jeder 8080-Befehl in einer eigenen Zeile des Quelltextes stehen. Der CP/M-Assembler unterstützt jedoch zwei Arten von Zeilenübergängen:

1. Das physikalische Zeilenende:
   Dieses wird, wie schon weiter oben besprochen, durch eine Kombination aus CR und LF markiert.

2. Das logische Zeilenende:
   Ein Ausrufezeichen '!' wird in der Regel vom Assembler genauso verarbeitet, als ob es ein physikalisches Zeilenende wäre. Das heißt für diesen beginnt nach einem Ausrufezeichen eine neue Zeile. In einer einzigen *Bildschirm*zeile kann man also durchaus mehrere Befehle hintereinander unterbringen, wenn sie durch ein Ausrufezeichen getrennt werden. Etwa 'MOV B,H!MOV C,L'.

Eine (logische) Befehlszeile kann ebenfalls 2 verschiedene Formate annehmen.

Die reine Kommentarzeile

Wenn eine logische Zeile mit einem Stern '*' beginnt, dann wird sie vom Assembler überhaupt nicht beachtet. Der Rest der Zeile kann also einen beliebigen Text enthalten. Diese Form einer Zeile sollten Sie für besonders auffallende Kommentare verwenden.

Die richtige Befehlszeile

Eine logische Zeile, die wirklich einen Assemblerbefehl enthält, hat die folgende Form:

```
<Zeilennummer> <Marke> <Befehl> <Argumente> <Kommentar>
```

Die Zeilennummer wird vom Assembler überhaupt nicht beachtet. Sie dürfen also nicht analog zum 'GOTO Zeile' in Basic im 8080-Assembler schreiben 'JP 10000' und damit keine Speicheradresse, sondern eine Zeilennummer meinen. In solchen Fällen sollten Sie 'JP <Marke>' schreiben und in der Zeile, die das Sprungziel enthält ebenfalls den Text <Marke> vor den eigentlichen Befehl setzen. Die Zeilennummer können Sie also ohne weiteres weglassen. Tatsächlich wurde diese Möglichkeit auch nur eingefügt, um auch Quellcodes, die schon einmal durch ein externes Programm (zum Beispiel PIP mit Optionen N oder N2) verändert wurden trotzdem weiterverwenden zu können.

<Befehl> ist der Hauptteil eines 8080-Assemblerbefehles und <Argumente> seine Zusätze. Ein MOV-Befehl benötigt zum Beispiel noch die Argumente 'A,B'. Eine Zahl in einem Argument kann auch durch einen symbolischen Namen ersetzt werden. Der Wert dieses symbolischen Namens wird dann durch den Assembler berechnet.

Ein Kommentar besteht aus einem führenden Strichpunkt ';' und daran anschließend einem beliebigen Text. Dieser Text wird vom Assembler nicht beachtet. Sie können also hier Hinweise zum Programmablauf unterbringen.

Wenn es nichts zu kommentieren gibt, dürfen Sie selbstverständlich den Kommentar samt Strichpunkt weglassen. Ein Ausrufezeichen gilt aber auch in einem Kommentar als logisches Zeilenende.

Aber auch alle anderen Bestandteile einer Zeile können wahlweise weggelassen werden: Natürlich brauchen Sie für einen Befehl wie etwa XCHG nicht extra ein Argument erfinden. Als Extremfall ist so auch eine Leerzeile eine korrekte Assemblerzeile.

In den meisten Fällen braucht bei einer Marke nur der symbolische Name angegeben werden. Steht einmal in einer Zeile kein Befehl, aber eine Marke (was erlaubt ist), so muß der Marke zusätzlich ein Doppelpunkt folgen. Eine erweiterte Marke sieht also so aus: <Marke>:

Wenn zwischen zwei Gruppen einer Zeile ein spezielles Trennzeichen steht (zum Beispiel ':' oder ';') und dadurch die Grenze eindeutig bestimmt werden kann, so dürfen die beiden Teile unmittelbar aufeinanderfolgen. Im anderen Fall müssen Sie selbst die beiden Teile entweder durch ein Leerzeichen oder ein Tabulationszeichen (Code 09h) voneinander trennen. Natürlich dürfen Sie auch mehrere Leerzeichen hintereinander schreiben.

Form symbolischer Namen

Äußere Form symbolischer Namen

Ein symbolischer Name enthält in der Regel nur Zahlen und Buchstaben, wobei das erste Zeichen ein Buchstabe sein muß. Groß- und Kleinbuchstaben werden vom Assembler nicht unterschieden, so daß Sie diese beliebig mischen dürfen. Die maximale Länge eines symbolischen Namens ist 16 Zeichen. Alle Zeichen in einem Namen tragen zur Unterscheidung verschiedener Namen bei. Also nicht etwa nur die ersten 6 wie bei manchen anderen Assemblern.

Lange Namen sind sehr unübersichtlich. Deshalb wurde die Möglichkeit eingerichtet, Dollarzeichen '$' an beliebiger Stelle im Namen einzusetzen. Diese Dollarzeichen werden vom Assembler nicht beachtet und tragen nur zur leichteren Lesbarkeit bei. Der symbolische Name SEHR$LANGER$-NAME wird also intern als SEHRLANGERNAME betrachtet.

Diese Dollarzeichen zur besseren Übersichtlichkeit dürfen genauso innerhalb von Konstanten auftreten. Eine 16-stellige binäre Zahl dürfen Sie etwa als '0101$1010$0011$1100B' schreiben.

Ein besonderer symbolischer Name

Eine Ausnahme bildet das Dollarzeichen '$', wenn es völlig alleinstehend, also nicht innerhalb eines symbolischen Namens auftritt. In diesem Fall wird es als eigener symbolischer Name mit einem bestimmten Wert gewertet. Das $-Zeichen wird dann als die Adresse des nächstfolgenden Befehls gewertet.

```
        ;letzter Befehl
        LXI  H,$          ;'$' wird als Zahl gewertet
HIER:   ;nächster Befehl
```

Im obigen Beispiel erhielte '$' den Wert, der auch dem symbolischen Namen HIER zugewiesen wird. Das würde auch funktionieren, wenn der Name HIER nicht definiert worden wäre.

Reservierte Wörter

Symbolische Namen, die genauso wie bestehende 8080-Befehle heißen, können auch als Argumente verwendet werden. Diese haben dann allerdings einen festen Wert, der ihrem Objektcode entspricht. Wenn der Objektcode von den Argumenten abhängt (zum Beispiel bei MVI), dann wird der niedrigste passende Wert verwendet. Bei MVI also der Wert 40h.

Weitere symbolische Namen mit festen Werten sind alle Registernamen. Über den numerischen Wert der einzelnen Registernamen gibt das Bild 4-1 Auskunft.

symbolischer Name	Wert
A	7
B	0
C	1
D	2
E	3
H	4
L	5
M	6
SP	6
PSW	6

Bild 4-1: *Werte reservierter Namen*

Diese Regelung ist zum Beispiel dann von Vorteil, wenn Sie ein Programm schreiben, das selbst ein anderes Programm entwirft, etwa eine Sprungtabelle (natürlich können Groß- und Kleinbuchstaben gemischt werden):

```
MOV  A,JMP      ;JMP hat den Wert C3h
LXI  H,ZIEL     ;Ziel des Sprungbefehls
STA  ADRESS     ;Der Sprungbefehl wird im Speicher abgelegt
SHLD ADRESS+1
```

Sie sehen, der symbolische Namen JMP hat, ohne daß Sie ihn extra definieren müssen, einen sinnvollen Wert. Sie brauchen also den Objektcode für den JMP-Befehl nicht in irgendeiner Tabelle zu suchen, wenn Sie ihn in einem Programm verwenden möchten. Dasselbe gilt auch für alle anderen Objektcodes.

Diese Methode birgt aber auch zusätzliche Gefahren:

Wenn Sie etwa das Register A mit dem Wert des Registers B laden möchten, dann sollten Sie den Befehl MOV A,B verwenden. Wenn Sie aus Unachtsamkeit aber stattdessen MVI A,B schreiben, dann spuckt der Assembler keine Fehlermeldung aus, sondern übersetzt den Befehl heimlich (fälschlicherweise) als MVI A,07H.

Gerade in längeren Programmen sind solche Schreibfehler kaum zu finden, wenn Sie nicht extra darauf achten.

Da die reservierten Wörter vorgegebene Werte haben, dürfen diese nur als Teil eines Arguments auftreten, nicht aber als Marke. Denn gerade durch die Verwendung als Marke wird ja einem symbolischen Namen ein anderer Wert zugewiesen.

Zahlensysteme des Assemblers

Numerische Konstanten können in einem Assembler-Quellprogramm binär, oktal, dezimal und hexadezimal angegeben werden. Zur Unterscheidung, auf welche Basis sich eine Zahlenangabe bezieht, folgt der Zahl ein Markierungszeichen. Dieses Markierungszeichen kann folgende Werte annehmen:

B	für eine binäre Konstante
O oder Q	für eine oktale Konstante. Das Zeichen 'Q' kann verwendet werden, um die Verwechslungsgefahr des Kennzeichens mit einer Null zu vermindern.
D	für eine dezimale Konstante. Dieses 'D' brauchen Sie nur zu verwenden, wenn Sie die Übersichtlichkeit erhöhen möchten. Fehlt nämlich ein Markierungszeichen ganz, so wird die Konstante vom Assembler automatisch als dezimal angesehen.
H	für eine hexadezimale Konstante. Wie üblich reräsentieren die Buchstaben A bis F die Zahlen 10D bis 15D des Hexadezimalsystems.

Eine hexadezimale Konstante, die mit einem Zeichen zwischen A und F beginnt, wird nach den bisherigen Regeln als symbolischer Namen inter-

pretiert. Es gilt deshalb die einschränkende Definition, daß auch hexadezimale Konstanten mit einer Ziffer zwischen 0 und 9 beginnen müssen. In der Regel stellt man in einem solchen Fall der Konstante einfach eine 0 voran, die den Wert ja nicht verändert. Um die Zahl ABH von der Konstante ABH zu unterscheiden, müssen Sie also die Zahl als 0ABH schreiben.

Beispiele für Konstanten verschiedener Zahlensysteme sind:

123, 123D, 102Q, 102O, 0101B, 1000$0000$0000$0001B, 7FH, 0F9H

Die numerischen Konstanten sind 16-Bit-Werte, das heißt sie können nur Werte zwischen 0 und 65535 annehmen. Hätte eine Konstante einen größeren Wert als 65535, so werden vom Assembler Vielfache von 65536 abgezogen. Die Konstante 100000 wird also als 100000-65536 = 34464 interpretiert.

Ein von einfachen Anführungsstrichen umgebenes druckbares Ascii-Zeichen wird ebenfalls als Konstante interpretiert. Der Wert der Konstante ist dann der Ascii-Code des Zeichens. MVI a,'A' ist also gleichbedeutend zu MVI A,41H.

Entsprechend werden 2 Ascii-Zeichen hintereinander zwischen Anführungszeichen als 16-Bit-Wert interpretiert. Das erste Zeichen wird dabei zum höherwertigen Byte der Konstante, das 2. Zeichen zum niederwertigen. Der Befehl LXI H,'AB' ist also gleichwertig zum Befehl LXI H,4142H. Diese Zeichenketten sind Spezialfälle der im folgenden Abschnitt besprochenen allgemeinen Zeichenketten.

Zeichenketten

Gelegentlich können in einer Assembler-Quelldatei auch längere Zeichenketten verwendet werden. Die absolut maximale erlaubte Länge ist dann 64 Zeichen, die von einfachen Anführungszeichen "'" eingeschlossen sind.

Als einzige Ausnahme in einem Assembler-Quellprogramm werden in einer Zeichenkette Kleinbuchstaben nicht in Großbuchstaben umgewandelt. Man kann also Texte mit Kleinschreibung und mit Großschreibung in ein Assemblerprogramm aufnehmen. Nur wenn es in einer Zeichenkette auftaucht, wird ein Ausrufezeichen nicht als logisches Zeilenende interpretiert. Die Zeichenfolge

```
'Kein Zeilenende!'
```

ist also zulässig.

Das Zeichen "'" in einer Zeichenkette wird als das Ende der Zeichenkette interpretiert. Soll in ein Assemblerprogramm aber das Zeichen "'" eingebaut werden, so gibt es trotzdem eine Möglichkeit: Ein doppeltes "'" in einer Zeichenkette wird nicht als Ende interpretiert, sondern als einfacher Apostroph in das Programm übernommen. Der Text "'abc''def'" liefert also die Zeichenfolge "abc'def", die Eingabe "''''" liefert den Text "'".

Operator	Priorität	Bedeutung
a + b	4	Addition
a - b	4	Subtraktion
+ b	4	Vorzeichen +
- b	4	Vorzeichen - (entspricht 0 - b)
a * b	5	vorzeichenlose Multiplikation
a / b	5	vorzeichenlose Division
a MOD b	5	Modulo-Funktion: Rest aus a / b
NOT b	3	Einerkomplement von b: In der Darstellungsform binäre 16-Bit-Integer-Zahl alle 0 durch 1 ersetzen und alle 1 durch 0
a AND b	2	Bitweises logisches Und: Jedes Bit aus a wird mit dem Bit gleicher Wertigkeit aus b verrechnet
a OR b	1	Bitweises logisches Oder
a XOR b	1	Bitweises logisches exclusiv-Oder
a SHL b	5	Die binäre Darstellung von a b-mal nach links schieben und rechts mit Nullen auffüllen. Entspricht a*(2^b).
a SHR b	5	Die binäre Darstellung von a b-mal nach rechts schieben und links mit Nullen auffüllen. Entspricht a*(2^(-b)).

Bild 4-2: *Assembler-Rechenoperationen*

Rechenoperationen

Ein Argument in einer Befehlszeile muß nicht unbedingt ein einfacher symbolischer Name oder eine Konstante sein, sondern kann auch aus mehreren Namen und Konstanten mittels Rechenoperationen zusammengesetzt werden.

Der 8080-Assembler kennt die in Bild 4-2 beschriebenen arithmetischen und logischen Operationen. 'a' und 'b' symbolisieren darin 2 beliebige Integerzahlen.

Die Vorrangstufe oder Priorität einer Operation bestimmt die Reihenfolge, in der mehrere Operationen in einer Zeile ausgeführt werden: Zunächst wird die Operation mit der höchsten Vorrangstufe berechnet. Erst mit derem Ergebnis wird die Operation mit der niederen Vorangstufe gefüttert. Der Assembler kennt also die Regel 'Punkt vor Strich': Im Ausdruck a+b*c wird zunächst b*c berechnet und erst das Ergebnis zu a addiert. Stehen einmal 2 Operationen mit der gleichen Priorität nebeneinander, so wird der Ausdruck von links nach rechts berechnet.

Diese Reihenfolge kann auch hier - wie normalerweise üblich - durch Setzen von Klammern durchbrochen werden. Wenn Sie sich einmal über die Vorrangstufen der verwendeten Operationen nicht sicher sind, schadet es also nicht, wenn Sie vorsichtshalber um die Teilausdrücke, die zuerst berechnet werden sollen, Klammern setzen.

Beachten Sie, daß alle Operatoren als 16-Bit-Integerzahlen betrachtet werden. (-1) zum Beispiel hat in dieser Darstellung den Wert FFFFh. Beim Versuch, diesen Wert in ein 8-Bit-Register zu schreiben, wird eine Fehlermeldung ausgegeben. Fügen Sie in diesem Fall noch AND FFh an den Ausdruck an, um das höherwertige Byte zu löschen.

Assembler-Direktiven

Zeilen mit Assembler-Direktiven beziehungsweise Pseudobefehlen können an einer beliebiger Stelle im Assemblerprogramm stehen und haben dieselbe Syntax wie gewöhnliche mnemonische Befehle. Ein Pseudobefehl ersetzt den Befehlsteil eines normalen Assemblerbefehls und kann genau wie dieser durch Label, Argumente und Kommentare ergänzt werden. Im Gegensatz zu den echten Assemblerbefehlen werden die Assemblerdirektiven nicht in einen ganz bestimmten einzelnen Objektcode übersetzt, sondern steuern die weitere Übersetzung des Programms im allgemeinen.

Folgende Pseudobefehle sind erlaubt:

Die ORG-Direktive

Form: <Marke> ORG <arithmetischer Ausdruck>
Mit diesem Befehl kann der Anfang des Maschinenprogramms festgelegt werden: Die nachfolgenden Assemblerzeilen werden vom Assembler fortlaufend ab der Adresse abgelegt, die der Wert des arithmetischen Ausdrucks angibt.
Im Programm können ORG-Befehle an beliebiger Stelle mit beliebigen Werten verwendet werden. Der Assembler übernimmt aber keine Kontrolle, ob sich Teile von Maschinenprogrammen überlappen. Darauf müssen Sie selbst achten. Kommt im Programm überhaupt kein ORG-Kommando vor, so beginnt die Ablage des Maschinenprogramms an der Adresse 0000h. Normale CP/M-Programme müssen aber an der Adresse 0100h beginnen. In der Regel wird also ORG 0100h eine der ersten Zeilen in Ihrem Assemblerprogramm sein.

Die END-Direktive

Form: <Marke> END
oder: <Marke> END <arithmetischer Ausdruck>
Dieser Pseudobefehl wird verwendet um das Ende des Assemblerprogramms zu markieren: Nur Zeilen vor dem END-Statements werden bei der Assemblierung berücksichtigt. Nach dem END kann also ein beliebiger Text folgen - zum Beispiel eine Programmerklärung. Wenn nach dem END kein Text mehr folgt, das Programmende also mit dem Dateiende zusammenfällt, dann dürfen Sie das END aber auch ohne Nachteile weglassen.
Die normale Form des Ende-Befehls ist ein alleinstehendes END. Folgt dem END aber ein arithmetischer Ausdruck, so wird dieser Ausdruck als Einsprungsadresse beim Start des Maschinenprogrammes interpretiert. Diese Einsprungadresse wird auch in der letzten Zeile der Hex-Datei vermerkt.

Die EQU-Direktive

Form: <Marke> EQU <arithmetischer Ausdruck>
Normalerweise würde die Marke den Wert der Adresse erhalten, an der Sie steht. Nicht aber, wenn sich die Marke vor einem EQU-Befehl befindet. In diesem Fall erhält die Marke den Wert des nachfolgenden arithmetischen Ausdrucks. Der Wert der Marke kann also - anders als vor einem normalen Assembler-Befehl - völlig frei und unabhängig von anderen Marken gewählt werden.
Der EQU- oder 'Equate'-Befehl entspricht also in etwa dem LET-Befehl in einem Basic-Programm. Mit dem EQU-Befehl kann einer

Marke jedoch nur ein einziges Mal ein Wert zugwiesen werden. Danach liegt der Wert im gesamten Assemblerprogramm fest und kann nicht mehr geändert werden. Jeder Versuch einer Neuzuweisung würde eine Fehlermeldung hervorrufen.
Der EQU-Befehl wird hauptsächlich verwendet, um häufig vorkommenden Konstanten einen der Bedeutung entsprechenden Namen zu geben. Dadurch wird das Programm lesbarer und Sie müssen den Wert der Konstante nur ein einziges Mal herausfinden. Wenn sich nachträglich herausstellt, daß für die Konstante doch besser ein anderer Wert gewählt werden sollte, dann müssen Sie nicht mehr das ganze Programm nach Vorkommnissen durchforsten. Die Änderung einer einzigen Zeile genügt dann. Sinnvollerweise fassen Sie also alle EQU-Anweisungen am Anfang des Programmes in einem Block zusammen. Das muß aber nicht unbedingt sein - für Spezialzwecke dürfen Sie eine EQU-Anweisung ruhig auch an anderen Programmstellen verwenden.

Die SET-Direktive

Form: `<Marke> SET <arithmetischer Ausdruck>`
Der SET-Befehl entspricht weitgehend einem EQU-Befehl. Auch mit ihm kann einer Marke ein beliebiger Wert zugewiesen werden.
Wird einer Marke bei Ihrem ersten Auftreten nicht mit einem EQU-Befehl, sondern mit einem SET-Befehl eine Wert zugewiesen, kann dieser Wert später im Programm mit einem weiteren SET-Befehl verändert werden. Durch einem SET-Befehl wird eine Marke also zu einer echten Variablen.

Die IF-Direktive

Form: `<Marke> IF <arithmetischer Ausdruck>`
Mit dem IF-Befehl ist es möglich, einen Teil des gesamten Assemblerlistings nur unter bestimten Bedingungen zu assemblieren und ins fertige Maschinenprogramm einzutragen.
Wenn der arithmetische Ausdruck den Wert 0 ergibt, dann werden die folgenden Zeilen vom Assembler übersprungen und nicht interpretiert. Ergibt der arithmetische Ausdruck einen Wert ungleich 0, dann wird die Abarbeitung des Assemblerlistings ganz normal fortgesetzt.
IF-Anweisungen dürfen aber nicht verschachtelt werden. Träte ein solcher Fall auf, so müßten Sie das Asssemblerlisting logisch so umformen, daß die IF-Anweisungen hintereinander stehen. Der Assembler selbst führt jedoch keine Überprüfung auf verschachtelte IF-Anweisungen durch. Wenn Sie nicht selbst darauf achten, kommt es zu undefinierten Resultaten.

Der IF-Befehl ist dann wichtig, wenn ein Assemblerprogramm für mehrere Anwendungen geeignet sein soll und zum Beispiel nur durch Änderung einer Konstante angepaßt werden soll.

Die ENDIF-Direktive

Form: <Marke> ENDIF
Ergab der arithmetische Ausdruck nach einer IF-Anweisung den Wert 0, so werden die folgenden Assemblerzeilen bis zu einer Zeile mit dem Inhalt ENDIF ausgelassen. Nach dem ENDIF-Befehl arbeitet der Assembler ganz normal weiter, egal ob die IF-Bedingung erfüllt war oder nicht.

Die DB-Direktive

Form: <Marke> DB <Ausdruck>, <Ausdruck>, ...
Mit Hilfe dieses Befehls können Sie im Maschinenprogramm Datenbereiche anlegen, zum Beispiel eine Tabelle, die von einem anderen Teil des Maschinenprogramms verwendet wird. Das kann etwa ein Text sein, der auf dem Bildschirm ausgegeben werden soll.
Nach einem DB-Befehl können beliebig viele Ausdrücke folgen, die durch Kommas voneinander getrennt sein müssen. Es können beliebig gemischt arithmetische Ausdrücke und Zeichenketten beliebiger Länge verwendet werden. Die Ergebnisse dieser Ausdrücke werden in der Reihenfolge ihres Auftretens im Programm abgelegt.
Der Wert eines arithmetischen Ausdrucks wird als ein Byte im Maschinenprogramm abgelegt. Das Ergebnis dieses Ausdrucks darf also höchstens 255 betragen. Die Zeichen einer Zeichenkette werden unverändert ins Maschinenprogramm übernommen und hintereinander als Bytes abgelegt. Beispiel:

```
DB 'Dieser Text wird im Maschinenprogramm abgelegt',13,10
DB 13,10,'und dieser auch'
```

Die DW-Direktive

Form: <Marke> DW <Ausdruck>, <Ausdruck>, ...
Der DW-Befehl wirkt ähnlich wie win DB-Befehl. Erlaubt sind hier aber nur arithmetische Ausdrücke und Zeichenketten mit einer Maximallänge von 2 Zeichen. Ein arithmeticher Ausdruck darf jetzt wirklich alle Werte zwischen 0 und FFFFh annehmen. Im Maschinenprogramm wird dieser Wert immer in 2 Bytes abgelegt - egal ob der Wert größer als 255 ist oder nicht. Dabei wird - wie im 8080-Assembler bei 16-Bit-Zahlen üblich - das niederwertige Byte vor dem höherwertigen Byte abgespeichert. Zeichenketten werden ebenfalls

immer in 2 Bytes abgespeichert. Ist die Zeichenkette nur ein Zeichen lang, so wird ein Null-Byte mit abgespeichert.

Die DS-Direktive

Form: <Marke> DS <arithmetischer Ausdruck>
Bei der Assemblierung überspringt der Assembler einen Bereich, der hinter dem DS-Befehl angegeben ist, ohne sich um den Inhalt zu kümmern. Im späteren Maschinenprogramm klafft dann eine Lücke. Sinnvoll ausgenützt werden kann diese Lücke, um etwa Platz für eine Tabelle zu reservieren. Das ist einfacher, als jedes einzelne Byte mit 'DW 0,0,0,..' zu reservieren.
Der DS-Befehl hat also genau dieselbe Wirkung wie die Befehlsfolge

```
ORG $ + <arithmetischer Ausdruck>
```

Ersatz wichtiger Z80-Befehle

Wenn Sie an den Z80-Prozessor gewöhnt sind, werden Sie einige besonders leistungsfähige Z80-Befehle im 8080-Assembler vermissen. In der Regel können Sie aber ein Programm so aufbauen, daß das Fehlen bestimmter Befehle nicht auffällt. Nur Befehlen, die zum Ersatz ein ganzes Programm benötigen, werden Sie wirklich nachtrauern. Wenn Sie sich keinen Z80-Assembler zulegen wollen und nicht unbedingt die ganze 8080-Norm erfüllen möchten, können Sie jedoch zu einigen Tricks greifen.

Ersatz der Z80-Blocktransferbefehle

Wenn sie im 8080-Assemblerprogramm Blocktransferbefehle verwenden wollen, dann sollten Sie dem Programm die folgenden Zeilen voranstellen:

```
ldi  equ  0A0EDh
ldir equ  0B0EDh
ldd  equ  0A8EDh
lddr equ  0B8EDh
cpi  equ  0A1EDh
cpir equ  0B1EDh
cpd  equ  0A9EDh
cpdr equ  0B9EDh
```

Wenn Sie dann später etwa den Z80-Befehl 'ldir' benötigen, schreiben Sie als Ersatz einfach 'dw ldir'. Genauso machen Sie es mit den anderen Blocktransferbefehle. Wenn Sie nur bestimmte Blocktransferbefehle benötigen, reicht es natürlich aus, wenn Sie nur den entsprechenden Ausschnitt aus der Erweiterungstabelle in das Assemblerprogramm setzen.

Ersatz der relativen Sprünge

Eine große Einschränkung ist das Fehlen der relativen Sprünge. Nur mit diesen ist die Erstellung von völlig frei verschiebbaren Maschinenprogrammen möglich. Analog zu den Blocktransferbefehlen können Sie sich auch hier wieder eine 'Befehlserweiterungstabelle' definieren:

```
jr   equ 18h
djnz equ 10h
jrz  equ 28h        ;Objektcode von 'jr  z,...'
jrnz equ 20h        ;Objektcode von 'jr nz,...'
jrc  equ 38h        ;Objektcode von 'jr  c,...'
jrnc equ 30h        ;Objektcode von 'jr nc,...'
```

Schwieriger wird es mit den relativen Sprungdistanzen: Diese Werte müssen durch einen arithmetischen Ausdruck simuliert werden. Zusätzlich sollte bei einem Wert größer als 128 beziehungsweise kleiner als -129 eine Bereichsüberschreitung auftreten. Da sowohl der Objektcode, als auch die Distanz 8-Bit-Werte sind, muß hier mit dem DB-Kommando gearbeitet werden. Lösungen für die Argumente sind:

```
relativer Vorwärtssprung:       Ziel-$+7Fh xor 80h
relativer Rückwärtssprung:      $-Ziel-7Fh xor 0FF80h
```

Ziel ist dabei die Marke, die auf das Sprungziel zeigt.

Ein DJNZ-Befehl mit einem Rückwärtssprung auf den Start der Schleife würde also im 8080-Assembler so aussehen:

```
loop:   ;Hier beginnt die Schleife
          .
          .
          .
        ;Ende der Schleife

    db  djnz,loop-$-7Fh xor 0FF80h
```

Ersatz weiterer Z80-Befehle

Hauptsächlich fehlen jetzt nur noch einige 16-Bit-Befehle. Wenn Sie einen derartigen in Ihrem Assemblerprogramm einsetzen wollen, dann sollten Sie sich selbst einen entsprechenden Namen definieren und einen EQU in Ihr Programm einfügen. Achten sie dabei auch auf die korrekte Darstellung der Befehlsparameter.

Für den Befehl 'ld (nn),de' ergäbe sich etwa folgendes:

```
ldnnde  equ     053EDh          ;'ld (nn),de
        dw      ldnnde,Adresse
```

4.1.3 Aufruf des Assemblers

Programmnamen

Bei der Übersetzung einer Quelldatei mit dem Namen <name>.ASM erzeugt der Assembler zwei Ergebnisdateien:

- Eine Datei mit dem Namen <name>.PRN.
  Diese Datei ist das Assemblerlisting. Sie enthält die vollständige Quelldatei, den sich ergebenden Objektcode und Hinweise auf Fehler, die beim Programmdurchlauf aufgetreten sind.

- Eine Datei mit dem Namen <name>.HEX.
  Diese Datei enthält nur den Objektcode des berechneten Maschinenprogrammes. Dort ist der Objektcode allerdings noch nicht in einer direkt ablauffähigen Form gespeichert, sondern als besondere Ascii-Datei. Lassen Sie sich doch einmal die HEX-Datei durch 'TYPE <name>.HEX' ausgeben! Die endgültige <name>.COM-Datei wird erst durch ein weiteres Übersetzungsprogramm erzeugt.

Der Assembler kann nur Quelldateien übersetzen, die die Namensextension ASM haben. Also mit einem Namen in der Form <name>.ASM. Die Namensextension braucht also beim Aufruf nicht angegeben zu werden. Ein einfaches 'ASM <name>' genügt.

Steuerung der Quell- und Zieldateien

Wird trotzdem eine Namensextension angegeben, wird diese nicht in der normalen Weise interpretiert, sondern dient zur Steuerung der Ablage der Zieldateien:

```
ASM <name>.<Asm-Datei><Hex-Datei><Prn-Datei>
```

<Asm-Datei>, <Hex-Datei> und <Prn-Datei> sind dabei einzelne Buchstaben.

- <Asm-Datei>:
  Der erste Buchstabe steuert das Laufwerk, auf dem die Quelldatei gesucht wird. Durch 'ASM <name>.B' wird also ein Assemblerprogramm auf dem Laufwerk B übersetzt. die anderen Buchstaben A-P haben eine entsprechende Bedeutung. Wahrscheinlich werden Sie aber niemals soviele Laufwerke angeschlossen haben.

- <Hex-Datei>:
  Mit dem 2. Buchstaben können Sie analog zur Assemblerdatei wählen, auf welchem Laufwerk die Hex-Datei abgelegt wird. Zusätzlich ist

hier aber noch der Buchstabe Z erlaubt. Durch Angabe dieses Buchstabens kann die Erzeugung einer Hex-Datei unterbunden werden. Das ist dann wichtig, wenn Sie nur untersuchen wollen, ob sich in Ihrem Assemblerprogramm noch Syntaxfehler befinden, und das Programm schnell übersetzt werden soll.

o <Prn-Datei>:
Mit dem 3. Buchstaben können Sie wählen, auf welchem Laufwerk die Prn-Datei abgelegt wird. Zusätzlich zu den Buchstaben A bis P ist auch hier wieder Z erlaubt: Bei Angabe dieses Buchstabens wird keine Prn-Datei erzeugt. Durch die Angabe des Buchstaben X wird die Prn-Datei auf dem Bildschirm ausgegeben. Die Ablage auf einem Laufwerk wird dafür unterbunden.

Beispiele:

`ASM <name>.AZZ`
Durch diesen Aufruf wird die Quelldatei auf dem Laufwerk A gesucht und die Ausgabe der Hex- und Prn-Datei unterbunden.

`ASM<name>.BAX`
Jetzt wird die Datei <name>.ASM auf dem Laufwerk B gesucht, die Datei <name>.HEX auf das Laufwerk A und das Assemblerlisting nur auf dem Bildschirm ausgegebem.

`ASM <name>.ASM`
Wenn Sie irrtümlich diese Eingabe machen, würde die ASM-Datei auf dem Laufwerk A gesucht, die HEX-Datei auf dem Laufwerk S abgelegt und die PRN-Datei auf dem Laufwerk M. Wahrscheinlich haben Sie aber nicht soviele Laufwerke angeschlossen. Deshalb erscheint die Fehlermeldung

```
Bdos Err On M: Select.
```

4.1.4 Fehlermeldungen

Fehlermeldungen, bei denen die Assemblierung abbricht

Auf Anhieb ist selten ein Assembler-Programm korrekt. Der Assembler teilt Ihnen deshalb zumindest die Fehler, die er entdeckt, mit, um Ihnen die Arbeit zu erleichtern.

Manche Fehler sind so schlimm, daß der Assembler nicht mehr weiterarbeiten kann. Das sind Fehler, die etwas mit der Disketten-Ein-/Ausgabe zu tun haben. Der Assembler bricht dann die Bearbeitung ab und gibt eine entsprechende Meldung auf dem Bildschirm aus. Diese Meldungen sind im einzelnen:

NO SOURCE FILE PRESENT
Das im Aufruf angegebene Assemblerprogramm existiert auf der Diskette nicht.

NO DIRECTORY SPACE
Das Inhaltsverzeichnis der Diskette ist voll, so daß die Zieldateien nicht mehr abgelegt werden können.

SOURCE FILE NAME ERROR
Im Aufruf wurde ein nicht korrekter Name angegeben, zum Beispiel mit einem enthaltenen Fragezeichen.

SOURCE FILE READ ERROR
Die Quelldatei kann vom Assembler nicht richtig gelesen werden. Das kann vorkommen, wenn ein unerlaubtes Steuerzeichen enthalten ist.

OUTPUT FILE WRITE ERROR
In die Zieldatei kann nichts geschrieben werden. Hauptsächlich tritt das bei einer vollen oder schreibgeschützten Datei.

CANNOT CLOSE FILE
Die Zieldatei kann nicht geschlossen werden. Dieser Fehler tritt vor allem bei schon vorhandenen alten und schreibgeschützten Zieldateien auf.

Fehler, die keinen Programmabbruch verursachen

Ist eine einzelne Assemblerzeile fehlerhaft, so wird die Bearbeitung des Programms nicht abgebrochen. Die fehlerhafte Zeile wird aber auf dem Bildschirm ausgegeben - unabhängig davon, ob das Listing ebenfalls auf dem Bildschirm ausgegeben wird. Jeder fehlerhaften Zeile wird zusätzlich ein Buchstabe vorangestellt, der die Art des Fehlers codiert. Diese Fehlercodes sind auch in der PRN-Datei enthalten, so daß Sie sie leicht entdecken können.

Der CP/M-Assembler kennt die folgenden Fehlermeldungen:

D *Data error*: Ein Argument in einem Data-Statement kann nicht korrekt im Datenbereich abgelegt werden, zum Beispiel eine Zeichenkette in einem DW-Befehl.

E *Expression error*: Ein Ausdruck ist nicht korrekt formuliert oder kann nicht berechnet werden. Etwa MARKE EQU MARKE.

L *Label error*: Eine Marke darf in diesem Zusammenhang nicht auftreten.

N *Not implemented*: Befehle, die erst von späteren Assemblerversion verarbeitet werden.

O *Overflow*: Ein Ausdruck ist zu kompliziert, um berechnet zu werden, etwa durch zuviele Klammerebenen.

P *Phase error*: Beim 2. Durchlauf hat eine Marke einen anderen Wert als beim ersten Durchlauf.
R *Register error*: Das in einem Argument angegebene Register darf bei diesem Befehl nicht verwendet werden.
S *Syntax error*: Ein Befehl wurde nicht richtig formuliert.
U *Undefined label*: Zur Definition eines symbolischen Namens wurde ein anderer symbolischer Name verwendet, der in den *vorhergehenden* Zeilen noch nicht definiert wurde.
V *Value error*: Ein Operand in einem Ausdruck ist nicht korrekt.

Nicht vom Assembler entdeckt werden natürlich Fehler, die zwar noch ein korrektes Maschinenprogramm erlauben, aber erst beim Ablauf unsinnige Berechnungen durchführen. Darauf müssen Sie schon selbst achten.

4.1.5 Erzeugen einer COM-Datei

Das LOAD-Programm

Der 8080-Assembler erzeugt noch kein COM-Programm aus einem Assemblerprogramm, sondern nur eine HEX-Datei. Erst das CP/M 2.2-Programm LOAD.COM übersetzt eine Hex-Datei in ein ohne Umstände aufrufbares COM-Maschinenprogramm.

Zum Erzeugen eines COM-Programmes müssen Sie also so vorgehen:

1. Erstellen Sie ein Assemblerprogramm mit dem Namen <Name>.ASM.
2. Übersetzen Sie es durch Eingabe von 'ASM <Name>'. Wenn Fehlermeldungen auftreten, dann sollten Sie das Assemblerprogramm korrigieren und den Assembler erneut aufrufen.
3. Hat der Assembler seine Arbeit ordnungsgemäß beendet, geben Sie 'LOAD <Name>' ein. Wenn keine Fehler auftreten, befindet sich danach eine Datei <Name>.COM auf der Diskette.
4. Immer, wenn sich die Datei <Name>.COM auf der Diskette im Bezugslaufwerk befindet, können Sie Ihr Assemblerprogramm wie gewöhnliche CP/M 2.2-Programme nur durch Eingabe von

```
<Name> <Parameter>
```

starten.

Das Intel-Hex-Format

Das Intel-Hex-Format wurde ursprünglich entwickelt, um Maschinenspracheprogramme auf Lochstreifen abzuspeichern. Um mit dessen eingeschränktem Zeichenvorrat auszukommen, ist ein Maschinenprogrammbyte

in 2 Hexadezimalziffern abgelegt. Die Hex-Datei ist also ein reiner Ascii-Text und nach Zeilen gegliedert.

Eine Zeile der Hex-Datei ist dabei wie folgt aufgebaut:

```
':'<Zeilenlänge><Adresse>'00'<Datenbytes><Prüfsumme>
```

- Das erste Zeichen jeder Zeile ist ein Doppelpunkt.
- <Zeilenlänge> gibt hexadezimal die Anzahl der Bytes an, die in dieser Zeile kodiert vorliegen. Gerechnet wird dabei die Anzahl der Bytes im Maschinenprogramm und nicht die Anzahl der Bytes in der Hex-Datei. Jedes zusätzliche Hexziffernpaar erhöht also die Zeilenlänge um 1. Die maximale Zeilenlänge liegt bei FF.
- <Startadresse> ist ein 16-Bit-Wert und gibt die Anfangsadresse an, an der das erste Datenbyte der Zeile abgelegt wird. Als Besonderheit wird hier das höherwertige Byte vor dem niederwertigen abgelegt - also umgekehrt zu einem Maschinenprogramm.
- '00' sind 2 Zeichen '0'.
- <Datenbytes>: Jetzt folgen hexadezimal codierte Datenbytes, deren Anzahl die <Zeilenlänge> angibt.
- <Prüfsumme>: Diese 8-Bit-Zahl beziehungsweise zweistellige Hexzahl ist die Prüfsumme. Diese ist so gewählt, daß die Summe aller zweistelligen Hexzahlen in einer Zeile (einschließlich der Prüfsumme und der Zeilenlänge) '00' als niederwertigste Hexziffern besitzt. Die Prüfsumme ist also der Rest, der noch zu einem vollen 256er fehlt beziehungsweise das Zweierkomplement Modulo 256 der Summe aller restlichen Hexziffern.

Um das Tabellenende zu bezeichnen gibt es eine besondere Abschlußzeile. Diese besteht in der Regel aus einem Doppelpunkt und 10 folgenden Nullen: ':0000000000'. (Zeilenlänge '0', Startadressse '0' und Prüfsumme '0').

Wenn das Assemblerprogramm mit einem End-Befehl beendet wurde, in dem auch eine Programmeinsprungadresse vereinbart wurde, so taucht diese Einsprungadresse in der letzten Hex-Zeile auf. Zusätzlich hat das Nullbyte nach der Adresse hier den Wert 1:

```
:00'<Adresse>'01'<Prüfsumme>
```

LOAD-Fehlermeldungen

Auch der Load-Befehl kennt bei falscher Anwendung einige Fehlermeldungen. Außer der eigentlichen Fehlermeldung wird die Zeile, in der der Fehler auftrat auf dem Bildschirm dargestellt.

Diese Fehlermeldungen sind im einzelnen:

INVALID HEX DIGIT
: Anstelle einer Hexadezimalzahl befindet sich in der Hex-Datei ein nicht erlaubtes Zeichen.

CHECKSUM ERROR
: Das Prüfsummenbyte paßt nicht mit dem Rest der Zeile zusammen. In diesem Fall liegt meistens eine Beschädigung der HEX-Datei vor.

ERROR: CANNOT OPEN SOURCE
: Die Quelldatei befindet sich nicht auf der Diskette.

ERROR: CANNOT CLOSE FILE
: Die Zieldiskette oder Zieldatei ist schreibgeschützt.

ERROR: DISK READ
: Fehler beim Versuch, einen Record von der Diskette zu lesen.

ERROR: DISK WRITE
: In die Datei konnte nicht geschrieben werden. Meistens ist in diesem Fall die Zieldiskette schreibgeschützt.

ERROR: INVERTED LOAD ADDRESS
: Die Hex-Datei enthält Zeilen, die an eine niedrigere Adresse als 0100h abgelegt werden sollen. Das wird meistens durch ein vergessenes ORG-Kommando im Assemblerprogramm verursacht.

ERROR: NO MORE DIRECTORY SPACE
: Das Inhaltsverzeichnis der Diskette ist voll. Sie sollten einige Dateien, die Sie nicht mehr benötigen, löschen.

4.2 Dateien testen und verändern

Der DDT.COM-Befehl

Das auf Ihrer CP/M 2.2-Diskette enthaltene Programm DDT ist ein 'Dynamic Debugging Tool', also ein Debugger. Im DDT enthalten ist ein Disassembler, ein Mini-Assembler, ein Tracer/Single Stepper und einige Befehle um Speicherinhalte zu verändern. Sie können mit ihm also alle Ihre Maschinenspracheprogramme testen. Passend zum CP/M-Assembler beherrscht er aber nur 8080-Mnemonics.

Gewöhnliche Anwenderprogramme werden vom Beginn des freien Arbeitsspeichers an aufwärts abgelegt. Würde das für den DDT auch gelten, müßte der Arbeitsspeicher solange er in Betrieb ist für andere Programme gesperrt sein. Deshalb wird der DDT nicht am unteren Ende des Arbeitsspeicher abgelegt, sondern direkt am oberen Ende. Zusätzlich werden einige Zeiger im Betriebssystem verbogen um den am oberen Ende benötigten Speicherplatz für andere Programme zu sperren. Ein nachträglich dazugekommenes Anwenderprogramm glaubt dann, daß das System einfach nicht über mehr Speicher verfügt und entdeckt gar nicht, daß sich der DDT auch noch im Speicher befindet. Wenn diesem Programm der restliche Speicher überhaupt ausreicht, kann es wirklich ohne Einschränkungen getestet werden.

Der Debugger wird einfach durch die Angabe des Namens ohne Parameter gestartet. Danach meldet sich das DDT-Programm mit der Bereitschaftsmeldung '-' und Sie können Ihre Befehle eingeben. Ein Befehl besteht aus einem einzelnen Buchstaben und eventuellen ohne Zwischenraum folgenden Parametern, wobei die Parameterwerte hexadezimal angegeben werden müssen. Der Befehl 'D0100,40FF' listet beispielsweise den Speicherbereich von 0100h bis 40FFh hexadezimal auf dem Bildschirm auf. Fehlt ein oder mehrere Parameter, so werden in der Regel Standardwerte angenommen. 'D0100h' ohne den 2. Parameter listet zum Beispiel genau 16 Bildschirmzeilen.

Alle länger dauernden Ausgaben lassen sich durch das Drücken von ^S anhalten und durch das Drücken einer beliebigen anderen Taste fortsetzen. Ohne vorhergehendes ^S bricht jede Taste die Ausgabe vollständig ab, so daß Sie eine neue Eingabe machen können.

Laden der zu untersuchenden Datei

Wenn Sie schon vor dem Programmstart wissen, welche Datei Sie untersuchen möchten, dann können Sie diesen Dateinamen gleich beim Aufruf von DDT angeben: 'DDT <Dateiname>'. Dabei sind sowohl Laufwerksangaben als auch alle möglichen Namensextensionen erlaubt.

Besonders behandelt wird nur die Namensextension .HEX. Soll eine solche Datei untersucht werden, so wird sie nicht Zeichen für Zeichen übernommen, sondern analog zum LOAD-Programm in die Maschinensprachebytes übersetzt. Erst das Ergebnis wird im Arbeitsspeicher abgelegt und kann mit dem DDT-Programm untersucht werden.

Unmittelbar nach dem Laden meldet das DDT-Programm die erste unbelegte Speicheradresse nach dem Programmende: Wurde etwa ein 1 KByte langes Maschinenprogramm in den Speicher geladen, so zeigt NEXT den Wert von 0100h+0400h = 0500h. PC ist die Einsprungadresse beim Starten des Programms, in der Regel also 0100h. Enthält aber eine geladene HEX-Datei in der letzten Zeile eine Programmstartadresse, so wird dieser Wert angezeigt.

Es ist aber auch möglich, nachträglich eine Datei zu der schon im Speicher befindlichen hinzuzuladen. So können sie in ein gerade übersetztes Assemblerprogramm ein schon vorhandenes einfügen oder ein 'Superprogramm' aus lauter kleinen Bausteinen zusammenzusetzen. Dazu dienen die beiden folgenden DDT-Befehle.

Der Input-Befehl I:

Form: `I<Dateiname>`
Mit diesem Befehl können Sie den Debugger für das Laden einer Datei vorbereiten. Alle nachfolgenden Lesebefehle beziehen sich auf den hier angegebenen Dateinamen. Der eigentliche Diskettenzugriff erfolgt jedoch erst durch den folgenden Lesebefehl.
Leider können Sie beim I-Befehl keine Laufwerksangabe machen. Eine Datei wird deshalb normalerweise immer vom Bezugslaufwerk gelesen.

Der Read-Befehl R:

Form: `R<Versatz>`
Durch diesen Befehl wird der eigentliche Diskettenzugriff auf die im I-Befehl angegebene Datei durchgeführt. Wie auch beim Laden einer Datei sofort beim Programmstart, wird Ihnen auch hier die Endadresse des Programms und die Einsprungadresse angegeben.
Ohne Angabe einer Versatzadresse wird eine gelesene Datei beliebigen Typs - wie bei ablauffähigen CP/M-Programmen auch sinnvoll - von

der Speicherstelle 0100h an aufwärts abgelegt. HEX-Dateien werden jedoch in dem Bereich abgelegt, der in den HEX-Zeilen angegeben ist. Die Programme wären also unter normalen Umständen sofort ablauffähig.
Durch Angabe einer Versatzadresse kann das Programm aber auch in einem anderen Speicherbereich abgelegt werden. Wenn Sie als Versatz etwa den Wert '0200h' angeben, so wird das Programm um 0200h Bytes nach oben verschoben. Es beginnt dann statt bei 0100h an der Adresse 0100h+0200h = 0300h.
Ein Verschieben nach unten ist auch möglich, allerdings können negative Zahlen nicht unmittelbar angegeben werden: -0080h zum Beispiel kann nicht als Versatzadresse angegeben werden. -0080h entspricht aber als vorzeichenbehaftete Binärzahl der Zahl 10000h-0080h = FF80h. Durch 'RFF80' wird die Datei so doch noch an der Adresse 0080h im Speicher abgelegt.

Speicher ansehen

Mit den folgenden Befehlen können Sie den gesamte Speicher des Computers betrachten - auch das Betriebssystem selbst. Der Adreßbereich der Befehle ist also nicht etwa nur auf den frei verfügbaren Arbeitsspeicher oder gar auf das untersuchte Programm beschränkt.

Der Display-Befehl D:

Form: D<Anfang>,<Ende>
Dieser Befehl gibt den Speicherbereich zwischen <Anfang> und <Ende> sowohl hexadezimal als auch im Ascii-Format aus. Ganz links in jeder Zeile sehen Sie dann die Anfangsadresse des dargestellten Bereiches, daneben 16 hexadezimalen Bytes aus dem dargestellten Bereich und ganz rechts die Bytes in Ascii-Codierung. Nicht druckbare Zeichen werden dabei als Punkte dargestellt. Beachten Sie, daß eine Zeile in der Regel immer an einer durch 16 teilbaren Adresse beginnt. Die erste und letzte Zeile der Auflistung können deshalb unvollständig erscheinen, wenn Sie 'krumme' Adressen verwenden.
Wenn Sie die Endadresse weglassen, werden genau 16 Zeilen aufgelistet. Wenn Sie auch die Anfangsadresse weglassen, beginnt die Auflistung mit der gerade aktuellen Adresse des Programmzählers, in der Regel also an der Adresse 0100h.
Das Drücken einer Taste hält - wie oben beschrieben - die Ausgabe an oder bricht sie ganz ab.

Der List-Befehl L:

Form: L<Anfang>,<Ende>
Dieser Befehl stellt die Disassemblerfunktion zur Verfügung. Mit ihm kann ein beliebiger Speicherbereich dargestellt werden. Im Gegensatz zum D-Befehl findet hier jedoch die Darstellung in 8080-Mnemonics statt.
Genauso wie beim D-Befehl wird aber die Startadresse und die Zahl der ausgegebenen Zeilen gesteuert. Ein Fehlen führt hier ebenfalls zur Wahl von Standardwerten.

Befehle zum Ändern des Speichers

Genauso wie beim Ansehen des Speichers können Sie auch hier jede einzelne Speicherzelle erreichen. Bei wichtigen Arbeiten sollten Sie darauf achten, nur Speicherzellen zu ändern, deren Bedeutung Sie verstehen. Sonst werden Sie Ihren Rechner sehr schnell zum Absturz bringen.

Besonders gefährlich sind hier Änderungen des Disketten-Betriebssystemes. Das ist aber schon passiert, wenn Sie statt '0A00' versehentlich 'A000' als Adresse angegeben haben. Wenn Sie einen solchen Fehler entdecken, ist es besser den Rechner sofort auszuschalten, als einen Verlust Ihrer Diskettendateien zu riskieren.

Der Set-Befehl S:

Form: S<Anfang>
Mit diesem Befehl können Sie fortlaufend hexadezimale Eingaben in einen zusammenhängenden Speicherbereich machen.
Nach der Eingabe von 'S' und der Anfangsadresse wird die Speicheradresse und der hexadezimale Wert des enthaltenen Bytes angezeigt und eine Eingabe von Ihnen verlangt. Wenn Sie nur ENTER drücken bleibt der Wert unverändert. Andernfalls können Sie einen neuen Wert hexadezimal eingeben. Nach dem Abschluß der Zeile durch ENTER wird sofort das Byte in der nächsten Speicherzelle angezeigt.
Durch irgendeine unsinnige Eingabe, etwa einem Punkt, beenden Sie das S-Kommando und können wieder andere Befehle eingeben.

Der Assembly-Befehl A:

Form: A<Anfang>
Die Bedienung dieses Befehls ist ähnlich der des S-Befehls. Jetzt müssen Sie allerdings Ihre Eingaben in 8080-Mnemonics machen.
Der A-Befehl ist also ein Mini-Assembler. Im Gegensatz zum richtigen Assembler stehen Ihnen hier aber keine Marken und Label zur Verfügung: Sie müssen also schon bei der Eingabe das genaue Sprungziel

wissen. Trotzdem ist er eine beachtliche Hilfe beim Testen von kurzen Maschinenspracheroutinen.
Den Assemblermodus können Sie durch eine leere Eingabe abbrechen.

Der Fill-Befehl F:

Form: F<Anfang>,<Ende>,<Wert>
Der Speicherbereich zwischen <Anfang> und <Ende> wird mit Bytes des Wertes <Wert> gefüllt. 'F0100,40FF,2A' löscht also die ersten 16 KByte des zur freien Verfügung stehenden Speicherbereiches mit Sternchen (das Zeichen '*' hat den Code 2A). Sie könnten aber auch Leerzeichen, Nullbytes oder irgendwelche andere Bytes verwenden.

Der Move-Befehl M:

Form: M<Anfang>,<Ende>,<Ziel>
Der Speicherbereich zwischen <Anfang> und <Ende> wird so kopiert, daß das Byte aus der Adresse <Anfang> auf die Adresse <Ziel> zu liegen kommt.
Achten Sie aber darauf, daß sich Start- und Zielbereich nicht überlappen, sonst kann es zu undefinierten Resultaten kommen.

Maschinenprogramme testen

Zum Testen von Maschinenprogrammen bietet das DDT-Programm leistungsfähige Hilfen. Programme können sowohl im Single-Step-Betrieb als auch im Echtzeitbetrieb abgearbeitet werden. Selbstverständlich ist, daß Sie vor dem Testlauf alle Prozessorregister auf den gewünschten Wert setzen können.

Für den eigentlichen Testlauf gibt es mehrere verschiedene Stufen, die sich in der Ablaufgeschwindigkeit und den Eingriffsmöglichkeiten unterscheiden.

- *langsamer Testlauf*: Nach jedem ausgeführtem Maschinensprachebefehl werden die Inhalte aller CPU-Register angezeigt. Bei Bedarf kann der Programmlauf zu jeder Zeit unterbrochen werden.
- *schneller Testlauf*: Die Anzeige der CPU-Register wird unterdrückt. Trotzdem kann der Programmdurchlauf zu jeder Zeit angehalten werden.
- *Echtzeitbetrieb*: Das zu testende Programm läuft in voller Geschwindigkeit ab und kann nicht vorzeitig unterbrochen werden. Nur beim 'zufälligen' Erreichen von besonderen vorgegebenen Haltepunkten wird die Kontrolle an Sie zurückgegeben.

In allen Stufen werden alle Betriebssystemroutinen in voller Geschwindigkeit ausgeführt. Vom Debugger werden diese also als 'Superbefehle' behandelt. Das Testen etwa eines Diskettenzugriffes wäre auch gar nicht sinnvoll: Diese Unterroutine ist garantiert fehlerfrei und würde Sie bei einem Durchlauf im Einzelschrittbetrieb nur vom Wesentlichen abhalten. Wegen des komplizierten Timings würde der Einzelschrittbetrieb bei derartigen Routinen sowieso nicht funktionieren.

Vermeiden Sie es unbedingt, Maschinenprogramme mit Hilfe des Debuggers zu testen, die Firmwareroutinen der CPC-Computer verwenden. Das hat 2 Gründe:

1. Die nicht-CP/M Firmwareroutinen der Schneider-Computer sind in Z80-Maschinensprache geschrieben. Leider kennt der 8080-Debugger viele dieser Befehle nicht.
2. Alle Firmwareroutinen benützen die Bankswitchinglogik der CPC-Computer. Ein Aufruf einer dieser Routinen würde der Debugger-Kontrolle 'den Boden unter den Füßen wegziehen'.

Wenn Sie trotzdem ein derartiges Maschinenprogramm mit dem Debugger testen, dürfen Sie sich nicht wundern, wenn Ihr Rechner abstürzt.

Der Examine-Befehl X:

Form: X<Register>
Durch ein alleinstehendes 'X' bringen Sie die Inhalte aller Register einschließlich der Flag-Register zur Anzeige. Dabei gelten die Bezeichnungen aus Bild 4-3:

Register	Bedeutung	erlaubte Werte
C	Carry Flag	0/1
Z	Zero Flag	0/1
M	Minus Flag	0/1
E	Gerade Parität	0/1
I	Halbübertragsflag	0/1
A	Akkumulator	0-FF
B	BC-Registerpaar	0-FFFF
D	DE-Registerpaar	0-FFFF
H	HL-Registerpaar	0-FFFF
S	Stapelzeiger	0-FFFF
P	Programmzähler	0-FFFF

Bild 4-3: *Angezeigte Register beim X-Befehl*

Wenn sie 'X' und unmittelbar dahinter einen der im Bild dargestellten Registernamen eintippen, können Sie dieses Register ändern. Dann wird zunächst der alte Wert des Registers angezeigt und die Eingabe des neuen

Wertes von Ihnen verlangt. Wenn Sie nur ENTER eintippen, bleibt das Register unverändert.

Der Trace-Befehl T:

Form: T<Anzahl>
Wenn Sie nur ein 'T' alleine eingeben, wird der nächste Befehl ausgeführt und eine Zeile mit den Inhalten der CPU-Register nach der Ausführung angezeigt. Am Ende der Bildschirmzeile steht die Adresse, an der das Programm fortgesetzt wird.
Die Zahl hinter 'T' gibt - wenn vorhanden - die Anzahl der Programmschritte an. Durch 'T10' werden also die nächsten 16 Befehle des Maschinenprogrammes durchgeführt, wobei bei jedem Befehl die CPU-Register angezeigt werden.

Der Untrace-Befehl U:

Form: U<Anzahl>
Dieser Befehl entspricht im wesentlichem dem T-Befehl. Nur die CPU-Register werden nicht bei jedem Schritt angezeigt, sondern erst nach dem letzten Programmschritt.

Der Go-Befehl G:

Form: G<Adresse>,<Breakpoint1>,<Breakpoint2>
Durch 'G<Adresse>' wird das zu testende Programm ab der Adresse <Adresse> gestartet. Dabei werden alle Kontrollen und Trace-Funktionen abgeschaltet! Das Programm wird dann genauso abgearbeitet, als ob es normal ablaufen würde. Ohne zusätzliche Maßnahmen läßt es sich auch nicht mehr stoppen.
Deshalb gilt folgendes:

- Die Befehlsfolge 'DDT <Name>.COM' gefolgt von einem 'G0100' ist absolut gleichwertig zu einem normalen Programmstart!
- Die Eingabe 'G0' entspricht einem Warmstart und ist äquivalent zu ^C. Mit dieser Eingabe kann das Disassemblerprogramm unterbrochen werden.

Wenn beim G-Befehl die Startadresse ausgelassen wird, wird das Programm an der aktuellen Position des Programmzählers gestartet. Diesen Wert können Sie zum Beispiel mit 'XP' herausfinden oder ändern.

Manchmal möchten Sie vielleicht ein Programm doch noch Abbrechen. Das kann der Fall sein, wenn Sie einen Programmteil testen wollen, der erst ganz am Schluß Ihres fehlerhaften Programmes steht, der vorhergehende Teil aber trotzdem ausgeführt werden muß um die Voraussetzungen zu lie-

fern. Aus Geschwindigkeitsgründen ist dazu ein T- oder U-Befehl schlecht geeignet.

Der Debugger bietet deshalb die Möglichkeit, 'Breakpoints' in das zu testende Programm einzufügen: Wird beim Programmdurchlauf ein solcher Haltepunkt erreicht, erhält der Debugger die Kontrolle über das System zurück. Diese Haltepunkte müssen aber wirklich ganz genau erreicht werden. Liegt der Haltepunkt etwa mitten in einem 3-Byte-Befehl, so entdeckt das Programm diese nicht: Beim letzten Befehl befindet es sich noch vor dem Haltepunkt und beim nächsten schon 'weit' dahinter. Wenn man darauf achtet, ist diese Technik aber ganz brauchbar.

Beim CP/M-Debugger dürfen Sie bei einem Programmstart 2 Haltepunkte angeben. Bereits wenn irgendeine der beiden (nicht etwa nur die erste) erreicht wird, bricht das zu testende Programm ab und gibt die Kontrolle dem Debugger zurück. Diese beiden Haltepunkte werden durch Kommas getrennt an den G-Befehl angehängt. Selbstverständlich dürfen Sie auch nur einen einzigen Breakpoint oder auch gar keinen an den G-Befehl anhängen.

Das Byte an der Programmstelle, die zu einem Breakpoint werden soll wird dabei vom Debugger an einer sicheren Stelle abgespeichert. An der Breakpoint-Stelle wird dann im Standard-DDT ein RST7-Befehl eingetragen. Der Debugger ist so aufgebaut, daß er bei jedem RST7-Befehl, der im Programm auftaucht, die Kontrolle übernimmt und die Breakpoints aus dem zu testenden Programm löscht. Die ursprünglich dort stehenden Werte werden natürlich restauriert. Als Preis für diesen Trick dürfen in zu testenden Programmen RST7-Befehle nicht auftreten. Sie werden sonst vom Debugger mit Breakpoints verwechselt. Diese Einschränkung dürfte Ihnen aber nicht allzu schwer fallen.

In den CPC-Computern würde ein RST7-Befehl in die Interruptroutine springen. Deshalb übernimmt im DDT der Schneider-Computer der RST6-Befehl die Aufgabe des RST7-Befehls.

Der Rechenbefehl

Der Hexadezimalrechenbefehl H:

Form: H<Zahl1>,<Zahl2>
Ausgegeben wird sowohl die hexadezimale Summe als auch die hexadezimale Differenz der beiden Zahlen.

Dateien speichern

Sicher ist Ihnen schon aufgefallen, daß bisher noch kein Debugger-Befehl besprochen wurde um die geänderte Version eines Programmes abzuspeichern. Diesen Befehl gibt es auch nicht.

Wenn Sie trotzdem eine geänderte Version abspeichern wollen, dann müssen Sie den normalen Save-Befehl des CP/M 2.2-Betriebssystemes verwenden. Nachdem Sie alle Verbesserungen am zu testenden Programm durchgeführt haben, sollten Sie also den Debugger mit ^C oder 'G0' unterbrechen. Danach können Sie das Programm mit 'SAVE <Länge> <Name>' abspeichern.

So berechnen Sie die richtige Länge des Programms:

Angenommen nach einem R-Befehl erscheint bei NEXT die Zahl 0900 als Ende eines geladenen Programmes. Diese Zahl sollten Sie sich merken. Das Programm hat eine Länge von 0900h-0100h = 0800h Bytes. Diesen Wert gebrochen durch 256 (dezimal) beziehungsweise 100h müssen Sie als Länge beim SAVE-Befehl angeben. Beim SAVE-Befehl muß der Wert aber dezimal angegeben werden! Sie müssen also die erhaltene Zahl ins 10er-System umrechnen. In diesem Fall ergibt sich also der Wert 0800h/0100h = 08h. Wenn die Division nicht genau aufgeht, müssen Sie unbedingt aufrunden, niemals abrunden! Sonst werden die letzten Bytes Ihres Programmes nicht mehr abgespeichert.

Eine hexadezimale Zahl können Sie ganz leicht durch 256 dividieren, indem Sie die letzten beiden Nullen abstreichen. Das ist dasselbe Prinzip, nach dem man im 10er-System eine Zahl ganz leicht durch 100 teilen kann.

Testen langer Programme

Der Debugger besteht aus 2 hierarchisch gegliederten Modulen:

- Den Kernroutinen
- und dem Assembler-/Disassemblerteil

Dabei liegt der Assembler-/Disassemblerteil an niedrigeren Speicherpositionen als der zentrale Teil. Der Kernteil ist außerdem so aufgebaut, daß er den Assembler-/Disasemblerteil zwar verwendet, wenn er vorhanden ist. Falls dieser fehlt, schadet das aber auch nichts: Der Debugger kennt dann eben die Befehle A und L nicht mehr.

Wenn Sie jetzt ein besonders langes Programm untersuchen möchten, dann können Sie dieses ohne weiteres laden. Der Debugger merkt das und schaltet den Assembler-/Disassemblerteil selbständig ab. Wenn er dann anschließend teilweise überschrieben wird, schadet das gar nichts.

Anwendbarkeit des Debuggers

Das DDT-Programm werden Sie in der Praxis hauptsächlich dazu verwenden, alle möglichen Arten von Dateien zu betrachten.

Der D-Befehl des Debuggers hilft Ihnen, Ascii-Dateien auszugeben, ohne daß die Bildschirmsteuerung zusammenbricht, auch wenn der Text einige Steuerzeichen besitzt. Andererseits können Sie aber in den Hex-Spalten immer noch die Codes der Steuerzeichen nachlesen. Sogar der Dateiheader von Basic-Programmen läßt sich mit dem Debugger betrachten.

Beim Entwurf kurzer Maschinenprogramme ist der Debugger tatsächlich schneller als der 'große' Assembler, mit dem auch für kleinste Programme noch einige Verarbeitungschritte nötig sind.

Speziell während der Arbeit mit diesem Buch können Sie den A-Befehl verwenden, um kurze Assembler-Beispielprogramme einzugeben. So sparen Sie sich bei jeder Änderung das langwierige Neuassemblieren. Mit ^C und 'SAVE 1 <NAME>.COM' können Sie Ihr Werk dann schnell abspeichern.

Erst in dritter Linie werden Sie den Debugger dazu verwenden, wirklich Fehler in langen Maschinenprogrammen aufzuspüren. Für diesem Zweck ist er aber immer noch sehr gut ausgerüstet.

5 Das Konzept von CP/M 2.2

5.1 Allgemeines

Die Geschichte von CP/M

Mit dem Aufkommen der ersten Diskettenlaufwerke war es nötig, ein Betriebssystem zu entwickeln, das diese neuen Speichermedien möglichst optimal verwaltet. Als eines der ersten wurde bei Digital Research das 'Control Program for Microprocessors' entwickelt.

Digital Research entschied sich ca. 1975, dieses Betriebssystem für den damals noch neuen 8080-Prozessor zu konzipieren. Gelegentlich werden alle CP/M-Versionen, die auf diesem Prozessor basieren, auch als CP/M-80 bezeichnet. Die bald darauf herausgegebene Version CP/M 1.4 fand schon bald eine größere Verbreitung und blieb lange in Gebrauch. Im Laufe der langen Anwendungszeit wurden aber einige kleinere und größere Mängel entdeckt. Deshalb wurde eine kräftige Überarbeitung in Angriff genommen, die außer 'richtigen' Fehler auch einige fundamental neue Möglichkeiten erhalten sollte. Diese neue Überarbeitung war nach einigen unvermeidlichen Verbesserungen fehlerfrei und wird unter dem Namen CP/M 2.2 verbreitet. Seit Mitte 1980 wurde die Version 2.2 nicht mehr verändert.

Diese Version kam zum ersten Einsatz von Personalcomputern im großen Stil gerade recht. CP/M 2.2 konnte so eine riesige Verbreitung erlangen und zu *dem* 8-Bit-Betriebssystem werden. Mittlerweile sind natürlich die 16-Bit-Computer eine große Konkurrenz, aber in kleineren Betrieben, die schon vor einigen Jahren mit dem Einsatz von Personalcomputern begannen, wird CP/M 2.2 immer noch eingesetzt.

Die Version Concurrent CP/M beziehungsweise CP/M-86 ist nicht als bloße Verbesserung zu werten. Da diese nicht mehr auf den 8080-/Z80-Prozessoren lauffähig sind, sondern nur noch auf einem 8086 ist CP/M-86 eigentlich ein neues Betriebssystem, das nur zufällig einen ähnlichen Namen wie die herkömmlichen Versionen trägt. Die Benutzeroberfläche hat zwar noch Ähnlichkeit mit der der älteren Versionen und auch Disketten mit reinen Textdateien können Sie sowohl mit Hilfe von CP/M-80 als auch mit Hilfe von CP/M-86 lesen. Die interne Struktur der beiden Betriebssysteme ist aber wirklich völlig verschieden.

Seit einiger Zeit ist eine neue CP/M-Version im Umlauf. Es ist die Version CP/M 3.0 beziehungsweise CP/M Plus und basiert weiterhin auf einem 8080- beziehungsweise Z80-Rechner. Diese Version ist wieder eine völlige Neubearbeitung mit fundamental neuen Möglichkeiten. Dazugekommen ist die Bankswitchlogik, so daß CP/M 3.0 auch mehr als 64 KByte Speicher sinnvoll verwalten kann. Weiterhin neu sind komfortablere Möglichkeiten zur Diskettenverwaltung. Obwohl noch eine Aufwärtskompatibilität zwischen CP/M 2.2 und CP/M 3.0 besteht, machen doch die neuen Möglichkeiten den wesentlichen Teil vom neuen CP/M aus.

Alle Möglichkeiten beider Versionen in ein Buch zu packen wäre zu verwirrend. Da die Version 3.0 aber (noch) sehr wenig verbreitet ist, werden hier nur die Möglichkeiten von CP/M 2.2 beschrieben. Die zusätzlichen CP/M 3.0-Möglichkeiten werden in einem späteren Band erklärt.

Die Möglichkeiten von CP/M 2.2

Virtuelle Ein-/Ausgabegeräte

Unter CP/M 2.2 können Sie über 4 verschiedene logische Kanäle Daten an die Ausgabegeräte senden oder von ihnen empfangen. Es handelt sich dabei um:

- *den Konsolenkanal*: In der Regel besteht dieser Kanal aus einer Tastatur (zum Empfangen) und einem Bildschirm (zum Senden). Das ist aber nicht die einzige Möglichkeit, wie Sie bald sehen werden. In der Regel wird dieser Kanal im CP/M 2.2-Betriebssystem mit CON: abgekürzt.
- *den Reader*: Mit diesem Namen ist ein Lochstreifenleser gemeint natürlich kann man über diesen Kanal nur Daten empfangen, nicht aber senden. Er wird im System meist mit RDR: abgekürzt.
- *den Puncher*: Dieser Kanal ist das Gegenstück zum Reader. über diesem Kanal können Sie Daten von einem Lochstreifenleser empfangen. Im System wird er häufig mit PUN: angekürzt.
- *den Printer*: Über diesen Kanal können sie Daten an den Drucker schicken. In der Regel ist dieser Kanal nur zur Ausgabe bestimmt, unter bestimmten Bedingungen können Sie aber empfangen, ob der Drucker bereit ist oder nicht. Der Name dieses Kanals wird oft zu LST: abgekürzt.

Schon an der Bezeichnungswahl der Kanäle erkennen Sie, daß das CP/M-Betriebssystem aus einer Zeit stammt, in der noch andere Ein-/Ausgabegeräte wichtig waren. Der Übergang von der Aufzeichnung auf Lochstreifen zur Aufzeichnung auf Diskette war anscheinend noch nicht ganz vollzogen. Die Entwickler von CP/M wollten sich wohl deshalb die Möglichkeit

offenhalten, schon früher auf Lochstreifen abgespeicherte Daten wiedereinzulesen. Heute haben solche Geräte natürlich keine große Bedeutung mehr. Sinnvoll wäre dafür aber etwa ein Kassetten-Streamer oder eine serielle Schnittstelle (zur Datenfernübertragung).

Glücklicherweise sind die Ein-/Ausgabekanäle nicht ein für allemal in ihrer Bedeutung festgelegt. Es handelt sich dabei nämlich nur um 'virtuelle Ausgabegeräte': Sie als Anwender oder die höheren Schichten des Betriebssystems senden ein Zeichen an einen bestimmten Kanal. An welches reale Gerät diese Daten dann endgültig weitergeleitet werden, können Sie getrennt davon wählen. Dasselbe gilt natürlich auch für die Eingabegeräte.

Reale Ein-/Ausgabegeräte

Die Zuordnung der virtuellen Ein-/Ausgabekanäle zu den realen Geräten können Sie mit Hilfe des CP/M-Befehls STAT erledigen. Dabei sind die folgenden Zuweisungen möglich:

Für den Konsolenkanal CON:

TTY: Teletype. Die Eingabe erfolgt über die Tastatur eines Fernschreibers und die Ausgabe über den Drucker eines Fernschreibers. Hier sehen Sie wieder eine nur historisch zu erklärende Möglichkeit: Zur Zeit als CP/M erstmals entwickelt wurde, waren Bildschirmgeräte noch nicht so verbreitet und wurden deshalb durch Fernschreiber ersetzt. Später wurde diese Ausgabemöglichkeit aus Kompatibilitätsgründen beibehalten. Deshalb bietet auch das Programm ED.COM nur so geringe Editiermöglichkeiten: Von einem Drucker lassen sich eben keine Zeichen löschen.

CRT: Cathode Ray Tube. Diese Zuweisung werden Sie in den meisten Fällen bei Ihrer Arbeit benötigen: Die Computertastatur als Eingabegerät und eine 'Kathodenstrahlröhre' (Bildschirm) als Ausgabeeinheit.

BAT: Batch. Nach dieser Zuweisung erfolgt die Konsoleneingabe über das Gerät, das dem Reader-Kanal zugewiesen wurde und die Ausgabe über den virtuellen Druckerkanal. Damit haben Sie eine Möglichkeit, mehrere Programme automatisch abarbeiten zu lassen. Wenn Sie reale Geräte als Lochstreifenleser/Stanzer angeschlossen haben, haben Sie einen Ersatz für Submit-Dateien und erhalten außerdem noch ein gedrucktes Arbeitsprotokoll. Hier können sie sogar nicht gepufferte Eingaben automatisch erledigen lassen, was selbst mit XSUB unmöglich ist.

UC1: User Console 1. Dieses Konsolengerät können Sie frei nach Ihren Wünschen definieren.

Für den Reader-Kanal RDR:

TTY: Teletype. Als Reader wird nach dieser Zuweisung der Lochstreifenleser des Fernschreibers verwendet.
PTR: Paper Tape Reader. Das ist ein besonderer Lochstreifenleser.
UR1: User Reader Device 1: Dieses Lesegerät dürfen Sie frei definieren.
UR2: User Reader Device 2: Dieses können Sie ebenfalls frei definieren.

Für den Lochstreifenstanzer-Kanal PUN:

TTY: Teletype. Hier wird der Fernschreiber zur Ausgabe verwendet. Diesmal jedoch der Lochstreifenstanzer des Gerätes.
PTP: Paper Tape Punch. Dieses Gerät ist ein besonderer Lochstreifenstanzer.
UP1: User Punch 1. Dieses Ausgabegerät dürfen Sie frei definieren.
UP2: User Punch 2. Ein weiteres frei definierbares Ausgabegerät.

Für den Druckerkanal LST:

TTY: Teletype. Nach dieser Zuweisung wird ebenfalls der Fernschreiber zur Ausgabe verwendet, dieses Mal aber der Druckerteil.
CRT: Cathode Ray Tube. nach dieser Zuweisung werden alle Druckerausgaben auf den Bildschirm umgeleitet. Das können Sie leicht mit 'STAT LST:=CRT:' durchführen. Wenn Sie danach ^P eintippen erscheinen alle eingebenen Zeichen doppelt auf dem Bildschirm: Einmal wie gewöhnlich und das 2. Mal als eigentlich für den Drucker bestimmter Protokollausdruck.
LPT: Line Printer. Das ist das normalerweise eingestellte Druckerausgabegerät.
UL1: User List Device 1. Dieses Gerät können Sie wieder nach Ihren eigenen Vorstellungen definieren.

An späterer Stelle im Buch werden Sie sehen, wie Sie selbstdefinierte Ausgabegeräte in Ihr System integrieren können.

Die Diskettensteuerung

Die Diskettenlaufwerke sind die universellen Ein-/Ausgabegeräte im CP/M-Betriebssystem. Die Entwickler legten deshalb auf die optimale Verwaltung der Disketten und auf die Datensicherheit großen Wert.

Wenn ein Stromausfall genau in dem Sekundenbruchteil auftritt, in dem ein Sektor auf der Diskette beschrieben wird, ist dieser Sektor unbrauchbar. Besonders wenn das während einer Änderung des Inhaltsverzeichnisses passiert ist das sehr unangenehm.

Beim Bearbeiten einer Datei bleibt die Originaldatei deshalb so lange wie möglich unverändert. Gewöhnlich legen CP/M-Programme alle Änderungen zuerst in eine Zwischendatei unter anderem Namen ab. Selbst wenn die Bearbeitung der Datei beendet ist, wird die Originaldatei nicht gelöscht. Sie wird nur umbenannt und ist mit unter dem Namen <Name>.BAK weiterhin auf der Diskette zu finden. Erst danach wird die Zwischendatei in den endgültigen Namen unbenannt. Nur eine eventuell schon vorhandene ältere Version von <Name>.BAK wird gelöscht.

Auf diese Weise wird eine optimale Datensicherheit erreicht:

- Tritt ein Schreibfehler während des Umbenennens der Originaldatei in die BAK-Datei auf, ist die Originaldatei zwar verloren, dafür finden Sie aber unter dem Namen der Zwischendatei immer noch die neueste Version Ihrer Datei.
- Tritt ein Schreibfehler während des Umbenennens der Zwischendatei in den Namen der Originaldatei auf, ist die neueste Version der Datei verloren. Dann steht Ihnen aber die BAK-Datei zur Verfügung, die den Stand der Datei vor der letzten Änderung beinhaltet.

Egal wann also ein Stromausfall oder eine andere Störung auftritt, Sie verlieren höchstens den letzten Bearbeitungsschritt Ihrer Datei.

Weiterhin braucht eine Datei nicht zusammenhängend in einem Stück auf der Diskette abgelegt sein. Es ist sogar in der Regel so, daß eine Datei quer über die ganze Diskette verteilt ist. Wo die einzelnen Teile der Datei stehen, ist im Inhaltsverzeichnis der Diskette vermerkt.

Wenn irgendwo eine Datei auf der Diskette gelöscht wird, brauchen also die nachfolgenden Dateien nicht verschoben zu werden, um den frei gewordenen Platz wieder zu besetzen. Eine neu dazugekommene Datei wird, ohne Speicherbereiche ungenutzt zu lassen, einfach irgendwo in die freien Teile geschrieben. Dadurch vermeidet man umständliche Kopieroperationen ähnlich der 'Garbage Collection' in einem Basic-Programm. Das kommt natürlich der Datensicherheit sehr zugute. Denn je häufiger kopiert wird, desto größer ist die Wahrscheinlichkeit, daß irgendwo ein Bit verfälscht wird.

Wie Sie sicher wissen, können Sie in einzelnen Dateien einen Vermerk anbringen, so daß diese schreibgeschützt ist. Außer mit Tricks (die ja ein gewöhnliches Anwenderprogramm nicht verwenden sollte) läßt sich eine solche Datei nicht löschen oder verändern. Sie ist also vor versehentlicher Zerstörung sehr gut geschützt.

Immer wenn es ohne nennenswerten Mehraufwand möglich ist, wird das Directory der Diskette gelesen und eine Prüfsumme berechnet. Das ist immer beim Öffnen oder Schließen einer Datei der Fall. Diese Prüfsumme wird mit einer gespeicherten Zahl verglichen. Wenn sich keine Übereinstimmung ergibt, weiß das Betriebssystem, daß die Diskette unzulässigerweise gewechselt wurde. Als Folge davon setzt das Betriebssystem die Diskette in den Nurlese-Status und gibt eine Warnung an den Benutzer aus. Wenn eigentlich an eine bestimmte Position der alten Diskette geschrieben werden sollte und sich an der entsprechenden Stelle auf der neuen Diskette schon wichtige Daten befinden, dann wird auf diese Weise die versehentliche Zerstörung wichtiger Daten verhindert.

Sie sehen, es wurde ein beträchtlicher Aufwand getrieben, um Ihre Aufzeichnungen zu schützen. Trotzdem sollten Sie aber kein blindes Vertrauen in die Technik haben, sondern gelegentlich eine Sicherheitskopie anfertigen.

Die Struktur von CP/M 2.2

Das CP/M 2.2-Betriebssystem ist weitgehend von der Hardware des verwendeten Computers unabhängig. Die Entwickler erreichten das, indem sie alle Betriebssystemfunktionen auf sehr einfache Ein-/Ausgabefunktionen zurückführten. Diese fundamentalen Ein-/Ausgabeoperationen werden nur von einer einzigen Stelle des Betriebssystems mit genau festgelegten Übergabeparametern aufgerufen. Nur durch eine einfache Änderung eines Sprungbefehles können Sie eine eigene Ein-/Ausgaberoutine in das Betriebssystem einfügen.

Nur diese wenigen Routinen müssen an den jeweiligen Computer angepaßt werden. Durch die umfangreiche Dokumentation macht Digital Research das dem Anwender darüber hinaus noch sehr leicht.

Diese bis ins Detail gehende Standardisierung ist auf der Welt wirklich einmalig. CP/M ist das einzige Betriebssystem, bei dem jedes Anwenderprogramm wirklich völlig kompatibel zu jedem beliebigen Computer mit diesem Betriebssysstem ist. Im Gegensatz zu anderen Betriebssystemen wurde dem Programmierer in CP/M die Anwendung der so beliebten nicht standardgemäßen 'Tricks', die den Entwicklern von 'kompatiblen' Computern große Schwierigkeiten machen, so schwer gemacht, daß er sich 'freiwillig' an den Standard hält. Lediglich bei Programmen, die schon vom Prinzip hardwareabhängig sind, ist nicht standardgemäße Programmierung ohne weiteres möglich. Das sind zum Beispiel Kaltstartlader oder Formatierprogramme. Bei diesen schadet aber die Nichtkompatibilität zu anderen Computern nicht.

Der Kern des Betriebssystem befindet sich ständig im RAM des Computers, und zwar am oberen Ende des Speichers. Dieser Kern hat ohne die hardwareabhängigen Teile einen Umfang von 3,5 KByte. Wenn die Bereitschaftsmeldung 'A>' oder 'B>' erscheint, wird zusätzlich ein einfacher Eingabeinterpreter in den Speicher direkt unterhalb des Betriebssystemkerns geladen. Dieser hat einen Umfang von 2 KByte und verarbeitet die 'residenten' CP/M 2.2-Befehle. Das sind die Befehle DIR, REN, ERA, TYPE, SAVE, USER und die Laufwerksumschaltungen. Alle weiteren und seltener gebrauchten Befehle werden wie ein Anwendungsprogramm erst bei Bedarf von der Diskette in den Arbeitsspeicher geholt und nach Gebrauch wieder gelöscht. So steht für die Anwendungsprogramme ein möglichst großer Arbeitsspeicher zur Verfügung. Diese nur bei Bedarf dazugeladenen Befehle heißen 'transiente Befehle' und haben auf der Diskettkette die Namensextension .COM.

Zum Abschluß jedes Anwendungsprogrammes wird eine Initialisierungsfunktion des Betriebssystemes aufgerufen. Sofern diese Routine überhaupt funktioniert, wird das gesamte residente Betriebssystem restauriert. Sie können also in speicherintensiven Anwendungen nicht benötigte Betriebssystemteile ohne weiteres überschreiben. Damit kann der verfügbare Arbeitsspeicher noch besser ausgenützt werden.

In den Schneider-Computern befinden sich aber ständig außer dem CP/M-Betriebssystem auch noch die CPC-Firmwareeinsprünge mit einem Umfang von 3,75 KByte und der 16 KByte-Bildschirm im Hauptspeicher. Der für das CP/M-Betriebssystem (nicht für den Anwender) frei verfügbare Speicher hat so nur noch eine Größe von 44 KByte. Da das CP/M-Betriebssystem selbst nur 5 KByte plus 1 KByte für die hardwareabhängigen Teile benötigt, stehen für Sie als Anwender nur noch knappe 39,5 KByte zur freien Verfügung.

Mehr freien Arbeitsspeicher erhalten Sie durch eine Speichererweiterung. Im CPC 464 kann so durch geschicktes Bankswitching der frei verfügbare Arbeitsspeicher auf etwa 60 KByte erweitert werden. Sie sollten aber nicht den Fehler machen, diese 20 KByte zusätzlicher Speicherplatz absolut zu sehen: Wenn Sie etwa ein Anwendungsprogramm verwenden, das schon im Ruhezustand 35 KByte benötigt, dann stehen für Ihre Dateneingaben nur noch etwa 5 KByte zur Verfügung. Durch die Erweiterung erhalten Sie dann 25 KByte zur Dateneingabe, so daß der Arbeitsspeicher effektiv sogar verfünffacht wird.

5.2 Die Speicheraufteilung unter CP/M 2.2

Die logische Untergliederung von CP/M 2.2

Das CP/M 2.2-Betriebssystem besteht aus 5 Teilen:

- Dem Basic Input Output System (abgekürzt BIOS)
- Dem Basic Disk Operating System (abgekürzt BDOS)
- Dem Console Command Processor (abgekürzt CCP)
- Der Transient Program Area (abgekürzt TPA)
- Der Zeropage

Dieses System benötigt einen zusammenhängenden und beschreibbaren Speicher (RAM) ab der Adresse 0000h. Nachdem in diesen Bereich das BIOS eingeladen wurde, müssen für BDOS, CCP, TPA und Zeropage noch mindestens 20 KByte Speicher frei bleiben. Die Schneider-Computer erfüllen diese Bedingungen, so daß Sie mit dem vollen Standard-CP/M arbeiten können.

Die beiden Teile BIOS und BDOS werden oft gemeinsam als FDOS bezeichnet. Diese beiden befinden sich ständig im obersten verfügbaren Speicherbereich. Das BIOS hat dabei eine beliebige Länge und das BDOS eine feste Länge von genau 3,5 KByte.

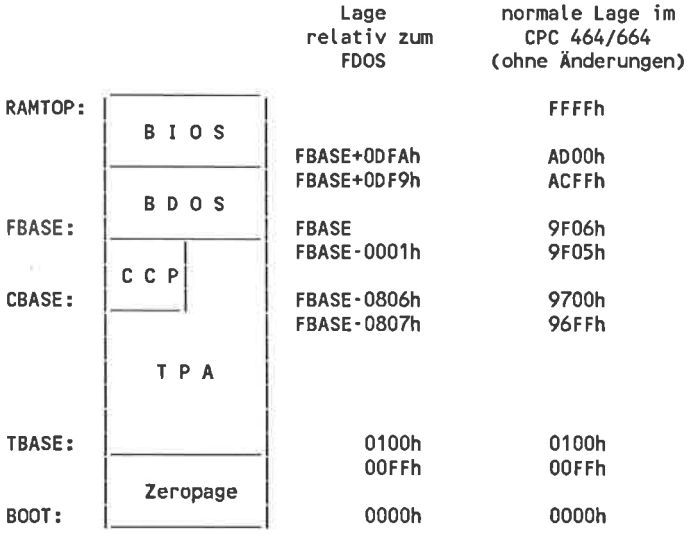


Bild 5-1: *Speicheraufteilung unter CP/M 2.2*

Die Zeropage liegt im Adressbereich von 0000h bis 00FFh und ist ebenfalls für das Betriebsssystem reserviert. Der ganze dazwischenliegende Bereich

mit einer unbestimmten Länge ist die 'Transient Program Area'. Diese kann von Anwendungsprogrammen beliebig verwendet werden. Die erste Speicherzelle außerhalb der TPA hat den Namen FBASE.

Der 'Console Command Prozessor' hat eine Länge von 2 KByte und befindet sich nur im Speicher, wenn er wirklich benötigt wird. Dann liegt er an der oberen Grenze der TPA aber noch innerhalb derselben. Das Bild 5-1 zeigt den genauen Aufbau des Speichers.

Wie Sie später noch sehen werden, können Sie das CP/M 2.2-Betriebssystem im Speicher verschieben. Die absoluten Zahlen ganz rechts im Bild 5-1 gelten deshalb nur für die originale CP/M-Version und ohne Speichererweiterung.

Das BIOS im CPC 464/664

Das BIOS stellt in einem CP/M-System wichtige hardwareabhängige Ein- und Ausgaberoutinen zur Verfügung. Dabei gibt es nicht nur Routinen, um zum Beispiel ein Zeichen an den Bildschirm auszugeben oder den Druckerstatus abzufragen, sondern auch einfache Routinen um die Diskettenlaufwerke anzusprechen.

Diese beschränken sich aber wirklich auf das notwendigste. Dem BIOS kann man nur mitteilen 'Gehe zur Spur 21', 'Suche den 7. Sektor' und 'Lese den gewählten Sektor'. Von der Verwaltung der Diskette, und ob jetzt Daten oder das Inhaltsverzeichnis gelesen wird, hat das BIOS keine Ahnung.

Ganz am Anfang des BIOS-Speicherbereiches befindet sich eine Sprungtabelle mit 17 Einträgen zu je 3 Byte, die die Aufrufe auf die realen Routinenadressen verteilt. Etwa ein Sprung zum 5. Tabelleneintrag wird auf eine Routine umgeleitet, die das Zeichen, dessen Code im Register C steht an den Bildschirm ausgibt.

Diese Tabelle beginnt an der Adresse FBASE+0DFAh und reicht bis zur Adresse FBASE+0E2Ch. Beim unveränderten Original-CP/M liegt sie also im Bereich von AD00h bis AD32h. Die Routine zur Zeichenausgabe wird dann mit 'call AD0Ch' gestartet.

Diese Tabelle ist der einzige Bereich im BIOS, den Sie als Benutzer im Standard-CP/M direkt ansprechen dürfen. Der restliche Teil ist vollständig für die Verwaltung durch das BIOS reserviert und für Sie tabu. Dieses BIOS kann zu verschiedenen Zeiten durchaus verschiedene Strukturen haben. In den CPC-Computern befindet sich das eigentlich BIOS-Programm in einem Erweiterungs-ROM, das an der Adresse C000h gelesen

werden kann. Gelegentlich kann an dieser Stelle aber auch der darunterliegende Bildschirmspeicher gelesen werden.

Während der Ausführung einer BIOS-Routine braucht der Speicher nicht unbedingt dem CP/M-Standard zu entsprechen. Das ist dann der Fall, wenn zum Beispiel umfangreiche Bankswitchingoperationen durchgeführt werden. Wichtig ist nur, daß diese Operationen für die höheren Schichten des Betriebssystemes völlig unsichtbar verlaufen.

Fordert also eine BIOS-Routine eine Bildschirmausgabe an, so wird zunächst das RAM im Bereich von 0000h bis 3FFFh abgeschaltet und durch ein ROM ersetzt. Umgekehrt wird das BIOS-ROM im Bereich von C000h bis FFFFh durch den Bildschirmspeicher ersetzt. Dann wird das gewünschte Zeichen auf dem Bildschirm ausgegeben. Bevor die Kontrolle wieder an das CP/M-Programm zurückgegeben wird, wird der vorherige Speicherzustand wiederhergestellt, so daß das Anwenderprogramm nichts von 'dem Vorfall' bemerkt.

Wenn Sie selbst CPC-Firmwareroutinen verwenden wollen, dann müssen Sie selbst auf alles Notwendige achten. Hüten Sie sich auf alle Fälle vor dauerhaften Speicherumschaltungen: Das wird den Rechner mit Sicherheit zum Absturz bringen. Achten Sie dann auch darauf, daß sich der Maschinenstapel immer im zentralen RAM befindet. Sonst ziehen sie dem Programm bei der Umschaltung den Boden unter den Füßen weg. Seien Sie sich aber bewußt, daß das so erzeugte Programm absolut hardwareabhängig ist und nur auf Ihrem CPC-Computer läuft. Sie hätten dann (außer in besonderen Fällen) genausogut ein normales Programm schreiben können.

Das BDOS des CP/M 2.2

Das BDOS ist bereits ein hardwareunabhängiger Teil des Betriebssystems. In allen CP/M 2.2-Computern ist dasselbe BDOS eingebaut. Das BDOS stellt genauso wie das BIOS einen Satz von Routinen zur Verwaltung des Rechners zur Verfügung - aber auf einem wesentlich höheren Niveau. Bei seiner Arbeit stützt sich das BDOS aber auf die BIOS-Routinen.

Wenn Sie BDOS-Routinen verwenden, dann brauchen Sie nur noch anzugeben 'Erzeuge eine Datei mit dem Namen XYZ' - die korrekte Ablage im Inhaltsverzeichnis und die Suche nach einem freien Speicherbereich auf der Diskette wird vom Betriebssystem selbst übernommen. Wenn Sie die Suche nach leeren Diskettenbereichen nicht unbedingt von Hand durchführen wollen und dem Betriebsystem diese Arbeit anvertrauen, können Sie sich also einiges an Arbeit sparen.

Mit Ausnahme der Druckerstatusabfrage haben auch alle anderen BIOS-Routinen ein BDOS-Äquivalent. In der Regel sollten Sie also die Verwendung der BIOS-Routinen vermeiden.

Der Console-Command-Prozessor

Die Ausgabe des CP/M-Bereitschaftszeichen 'A>' oder 'B>' und die Interpretation der daraufhin gemachten Eingaben übernimmt der CCP. Bei jedem Warmstart wird dieser zusammen mit dem BDOS in den Arbeitsspeicher geladen und sofort gestartet. Daraufhin untersucht er, ob eine Submit-Datei abzuarbeiten ist oder die Tastatur abgefragt werden soll. Die erhaltene Eingabe wird überprüft, ob es sich um einen residenten oder einen transienten Befehl handelt. Ist die Eingabe ein residenter Befehl, so wird er sofort ausgeführt. Ist die Eingabe ein transienter Befehl, so wird die zugehörige Datei in den Arbeitsspeicher geladen und das Programm gestartet.

Der CCP liegt am oberen Ende des freien Speicherbereiches. Im Gegensatz zum BDOS ist der CCP aber nicht vor Überschreiben geschützt. Lange Programme können ohne weiteres den CCP löschen und den freigewordenen Speicherplatz für eigene Zwecke verwenden. Diese Programme müssen dann aber unbedingt mit einem Warmstart abgeschlossen werden, damit das Betriebssystem restauriert wird.

Wenn Sie nur ein kurzes Programm haben ablaufen lassen und sicher sind, daß es das Betriebssystem nicht beschädigt hat, können Sie sich den Warmstart aber sparen. In diesem Fall sollten Sie das Programm einfach mit RET beenden: Der CCP erhält dann automatisch die Kontrolle über das System zurück und macht da weiter, wo er vor dem Programmaufruf unterbrochen wurde.

Der freie Arbeitsspeicher

Der freie Arbeitsspeicher oder auch 'Transient Program Area' beginnt an der Adresse 0100h und endet unmittelbar vor dem Anfang des FDOS. Über diesen Bereich können Anwenderprogramme und transiente Befehle während der Programmabarbeitung frei verfügen.

In der Regel befindet sich das abzuarbeitende Programm ganz am Anfang der TPA. Der CCP startet das transiente Programm durch einen Sprung an die Adresse 0100h. An dieser Stelle muß also jedes Programm seine Einsprungstelle haben. Wenn Sie dieses Programm mit einem Assembler entwickeln, sollten Sie es also mit den 2 Zeilen 'ORG 0100h' und 'JMP Start' beginnen lassen. Dann ist zumindest eine Voraussetzung für das Gelingen des Projektes gegeben.

Wenn Sie sicher sind, daß das Programm keinen Teil des Betriebssystems beschädigt hat, dürfen Sie es wie schon im Abschnitt 'TPA' erwähnt, einfach mit einem RET-Befehl beenden. Andernfalls muß der letzte Befehl ein 'JMP 0000h' sein.

Die Zeropage

Die Zeropage ist die erste Speicher-'Seite' im Computer, also der Bereich, bei dem das höherwertige Adreßbyte Null ist.

In diesem Bereich liegen ergänzende Schnittstellen zur Verbindung zwischen Anwenderprogramm und Betriebssystem. Den größten Teil nehmen Datenpuffer für die Diskettenausgabe und Informationen über die gerade bearbeitete Datei ein.

Zwei Zeiger auf das TPA-Ende und die Adresse der Warmstartroutine ermöglichen es dem Programm, den zur Verfügung stehenden Speicher auszumessen. Die Zuordnung der virtuellen Ein-/Ausgabegeräte ist ebenfalls in der Zeropage festgehalten.

In einem Standard-CP/M 2.2-System wären die RST-Befehle - der übrige Bereich in der Zeropage - unbenutzt und könnte für Anwendungsprogramme zu beliebigen Zwecken verwendet werden. In den CPC-Computern sind diese RST-Befehle aber mit den Routinen zur Speicherbankauswahl und der Interruptsteuerung belegt. Nur der RST 6-Befehl ist zunächst noch frei. Für Anwendungsprogramme sollte er trotzdem nicht verwendet werden, da dieser von Debuggern für Breakpoints benötigt wird. Würde dieser RST-Befehl trotzdem verwendet, ließe sich das Programm mit einfachen Mitteln nicht mehr testen.

5.3 Die Diskettenaufteilung in CP/M

Physikalische Diskettenaufteilung

Spuren

Anders als bei einer Schallplatte sind Ihre Daten auf der Diskette nicht in einer Spirale, sondern in konzentrischen Kreisen angeordnet. Diese Kreise nennt man 'Spuren' oder auch 'Tracks'. Üblicherweise befinden sich 40 oder 80 dieser Kreise auf einer Diskettenseite. Manche Computer verwenden aber andere Anzahlen. Diese Anzahl hängt von der Diskettengröße ab und von der maximalen Dichte, in der die Spuren stehen können, ohne daß der Schreib-/Lesekopf des Laufwerks zwei Spuren zugleich verarbeitet und zu unsinnigen Ergebnissen kommt. In geringerem Maße hängt dies auch von der Güte der Diskette und von den mechanischen Laufwerkseigenschaften ab: Die Diskette muß so genau zentriert werden, daß sich der Lesekopf während einer Umdrehung immer über derselben Spur befindet. Die verwendete Spurdichte wird in 'Spuren pro Zoll' angegeben. Diese Einheit wird mit 'tpi' (von englisch: 'Tracks Per Inch') abgekürzt. Heute sind bis zu 96tpi üblich, also etwa 4 Spuren pro Millimeter. Dieser Wert wird von einem 5 1/4", 80-Spur Laufwerk und vom Schneider-3"-Laufwerk erreicht.

Sektoren

Auf einer einzigen Spur finden je nach System und Aufzeichnungsverfahren etwa 4 KByte Daten Platz. Beim Schneider 3"-Laufwerk etwa sind es 4,5 KByte. Bei diesem Laufwerk hat die innerste Spur einen Umfang von etwa 12 cm. Auf einem Millimeter der Spur müssen also etwa 320 Bits Platz finden. Auf einer Fläche von 4*4 mm^2 wird also eine ganze Schreibmaschinenseite abgespeichert!

Zum äußeren Rand hin wird die Aufzeichnungsdichte allerdings geringer: Genau wie bei einer Schallplatte bleibt die Umdrehungsgeschwindigkeit konstant. Die in derselben Zeit aufgezeichneten Daten werden also auf einen größeren Umfang verteilt.

Auf einer Spur sind die Daten weiter aufgeteilt: Beim Schneider 3"-Laufwerk sind jeweils 512 Bytes zu einer Gruppe zusammengefaßt. Bei anderen Computern sind aber auch Größen von 128, 256 oder 1024 Bytes üblich. Eine Gruppe kann nur als ganzes gelesen oder geschrieben werden. Es ist also nicht möglich, etwa nur die ersten 150 Bytes einer Gruppe einzulesen: In diesem Fall müssen Sie eben die umsonst gelesenen Daten unbeachtet lassen. Diese Gruppen werden in der Regel 'Sektoren' genannt. Auf einer

Spur des 3"-Laufwerkes befinden sich genau 9 Sektoren. Für jeden dieser Sektoren ist auf der Diskette ein genau festgelegter Platz reserviert. Nur so ist es möglich, einzelne Sektoren neu zu beschreiben ohne daß sich der neue Sektor mit einem anderen überlappt.

Formatieren einer Diskette

Beim Formatieren einer Diskette wird an jeder Position, an der später ein Sektor beginnen soll, eine Markierung mit dessen Nummer angebracht. Ohne vorheriges Formatieren ist deshalb das Beschreiben einer Diskette nicht möglich: Der Computer würde vergeblich nach den Markierungen suchen, nach denen er sich auf der Diskette orientieren soll.

Bei den CPC-Computern mit einer 3"-Floppydisk können Sie 3 verschiedene Formate auswählen, die sich durch die Nummern der Sektoren unterscheiden. Unter CP/M läßt sich ohne Einschränkungen aber nur das Systemformat verwenden, bei dem die Sektoren Nummern zwischen 41h und 49h tragen. Die Nummern der Sektoren müssen also nicht bei Null beginnen und nicht einmal in einem zusammenhängenden Bereich liegen: Die Wahl ist ganz allein die Entscheidung des Systemprogrammierers. Bei der Vortex-Floppydisk zum CPC 464 tragen die Sektoren zum Beispiel die Nummern 01h bis 09h.

Bei manchen Laufwerkstypen können Sie eine zweiseitige Aufzeichnung durch einfaches Umdrehen der Diskette erreichen. Andere Laufwerke wie etwa das Vortex-Laufwerk haben für jede Seite einen eigenen Schreib-/Lesekopf eingebaut, so daß das Umdrehen entfällt. Beachten Sie aber, daß das Lesen der Diskettenrückseite einer Doppelkopfaufzeichnung mit einem normalen Laufwerk unmöglich ist: Durch das Umdrehen befindet sich die Aufzeichnung der Unterseite sozusagen 'linksherum' beziehungsweise rückwärts auf der Diskette. Genauso unmöglich ist es, eine Doppelkopfdiskette umzudrehen und die Unterseite als Oberseite zu verwenden. Die beiden Diskettenseiten werden deshalb vom Betriebssystem nicht als 2 verschiedene Disketten verwaltet, sondern als eine einzige Diskette. In der Regel haben dann Aufzeichnungen der Oberseite ungerade Spurnummern und Aufzeichnungen der Unterseite gerade Spurnummern. Das hardwareunabhängige BDOS kennt nur eine einzige Diskettenseite und verwendet einfach durchgehende Nummern für alle Spuren. Erst der hardwarabhängige BIOS-Teil des Betriebssystems übernimmt die Verteilung der Nummern auf die realen Spuren.

Der Sektorversatz

Jeder gelesene Sektor muß erst verarbeitet werden, was eine bestimmte Zeit dauert. Währenddessen dreht sich die Diskette unter dem Schreib-/Lesekopf weiter. Wenn Sie (was häufig vorkommt) den unmittelbar folgenden Sektor auch lesen wollen, müßten Sie warten bis die Diskette eine vollständige Umdrehung durchgeführt hat und der verpaßte Sektor erneut vorbeikommt.

Um dieses Wartezeit zu verkürzen greift man in der Regel zu einem Trick: Die eigentlich logisch hintereinanderfolgenden Sektoren werden mit einem gewissen Abstand voneinander auf einer Spur angeordnet. Dazwischen finden dann andere Sektoren Platz. Die 9 Sektoren des Schneider 3"-Laufwerkes sind zum Beispiel in der folgenden Reihenfolge angeordnet:

41h 46h 42h 47h 43h 48h 44h 49h 45h

Nachdem der Sektor 41h gelesen wurde, muß der Computer die Daten verarbeiten. Währenddessen streicht unter dem Schreib-/Lesekopf der Sektor 46h vorbei. Wenn unmittelbar danach der Sektor 42h gelesen werden soll, ist dieser fast ohne Verzögerung greifbar. Zwischen dem Lesen zweier aufeinanderfolgender Sektoren braucht also nicht mehr eine volle Diskettenumdrehung gewartet werden, sondern nur noch eine 1/5-Umdrehung. Beim Einladen langer Dateien ist das eine spürbare Beschleunigung.

In manchen CP/M-Systemen befinden sich trotz dieses Tricks Sektoren mit aufeinanderfolgenden Nummern hintereinander auf der Diskette. Irgendwo im BIOS befindet sich dann aber eine Umrechnungstabelle, die angibt, in welchem Sektor die gewünschten Daten wirklich stehen.

Sektor-Blocking/Deblocking

In verschiedenen Systemen können die Sektoren eine verschiedene Größe haben. Eine wichtige Eigenschaft von CP/M ist aber, daß Anwenderprogramme völlig hardwareunabhängig sind. Im Betriebssystem erfolgen deshalb alle Datentransfers in Gruppen zu je 128 Bytes. Bereits das BDOS - der erste hardwareunabhängige Teil des Betriebssystem - kennt die Übertragungseinheit 'Sektor' nicht mehr: Das BIOS sammelt selbständig alle vom BDOS eingegangenen 128-Byte-Einheiten, bis sich ein kompletter Sektor angesammelt hat, der auf der Diskette abgespeichert werden kann. Umgekehrt speichert das BIOS alle von der Diskette eingegangenen 512 Byte-Sektoren und gibt diese bei jedem Lesebefehl des BDOS nur in 128-Byte Portionen weiter. Zu diesem Zweck gibt es eine BIOS-Routine, die die logische Sektornummer in die Nummer des wirklichen Sektors übersetzt, der die Daten enthält. In den CPC-Computern wird die Nummer jedoch

doch nicht verändert, da hier die BIOS-Lese-/Schreibbefehle selbst eine entsprechende Routine enthalten.

Selbstverständlich achtet das BIOS darauf, daß keine Informationen verlorengehen: Wenn das BDOS etwa Daten lesen möchte, die sich in einem anderen Sektor befinden, schreibt das BIOS den vom BDOS teilweise beschriebenen Sektorpuffer auch dann auf Diskette, wenn dieser noch nicht ganz vollständig ist. So kann die kuriose Situation auftreten, daß bei einem Lesebefehl Daten auf die Diskette geschrieben werden. In bestimmten Fällen ist auch der umgekehrte Fall möglich.

In Anwenderprogrammen müssen Sie also ausschließlich mit 128 Byte-Übertragungseinheiten arbeiten. Dabei brauchen Sie sich überhaupt nicht darum zu kümmern, wo und wann diese auf der Diskette abgespeichert werden - auch nicht, wenn Sie in Maschinensprache programmieren. Eine 128-Byte-Einheit wird im folgenden immer 'Record' genannt. Das Wort 'Sektor' bezieht sich daher immer auf den physikalischen Datenabschnitt auf der Diskette mit variabler Größe.

Logische Reihenfolge der Sektoren	Spur 0	Spur 1
1	BOOT	CCP
2	PARA	CCP
3	leer	BDOS
4	leer	BDOS
5	leer	BDOS
6	leer	BDOS
7	leer	BDOS
8	CCP	BDOS
9	CCP	BDOS

Bild 5-2: *Organisation der Systemspuren*

Die logische Aufteilung der Disketten

Eine Diskette ist in 2 Abschnitte eingeteilt:

- o den Systembereich und
- o den Datenbereich.

Der Systembereich

Die beiden äußersten Spuren der Diskette sind für die ausschließliche Verwendung durch das CP/M-Betriebssystem reserviert und werden 'Systemspuren' genannt. In den CPC-Computern sind diese beiden Spuren so organisiert, wie Bild 5-2 zeigt. Diese Abbildung gilt beim CPC 464 sowohl für das 3"-Laufwerk als auch für das Vortex-Laufwerk. Die Systemspuren von Laufwerken anderer Firmen können jedoch anders organisiert sein.

Im Sektor BOOT befindet sich der sogenannte 'Kaltstartlader': Das ist ein Maschinenprogramm, das sofort nach dem Einschalten des CP/M-Modus in den Speicher geladen und ausgeführt wird. Dieses Programm steuert das Initialisieren des restlichen CP/M-Systems. Durch eine Veränderung dieses Sektors könnte also auch ein anderes Betriebssystem auf den CPC-Computern verwendet werden.

Der Sektor PARA enthält die Parameter zur Voreinstellung des Betriebssystems. Wenn Sie mit Hilfe von SETUP.COM oder INSTALL.COM die Standard-Tastaturbelegung Ihres Computers geändert haben, dann ist die neue Belegung in diesem Sektor vermerkt. Das gleiche gilt für alle anderen wählbaren Voreinstellungen.

Die nächsten fünf Sektoren sind unbelegt und können von Ihnen für eigene Zwecke verwendet werden.

Die übrigen 13 Sektoren enthalten den CCP und das BDOS des CP/M-Betriebssystemes. Nach jedem Warmstart wird das Betriebssystem ja restauriert um eventuelle Beschädigungen zu vermeiden. Dabei wird das Betriebssystem immer von den Systemspuren der Diskette im Laufwerk A geholt - gleichgültig welches Laufwerk gerade das Bezugslaufwerk ist. Die Diskette im Laufwerk B bräuchte also nicht unbedingt Systemspuren zu besitzen. Es ist aber trotzdem besser, wenn alle Ihre Disketten Systemspuren besitzen - auch wenn diese manchmal nicht gelesen werden. Nur so können Sie jede Diskette sowohl im Laufwerk A als auch im Laufwerk B verwenden.

Die meisten CP/M-Systeme besitzen wie die Schneider-Computer zwei Systemspuren. Möglich wäre aber auch eine andere Voreinstellung.

Durch einen neuen Kaltstartlader, der das Betriebssystem auf die neue Systemspurzahl vorbereitet, verbunden mit einem Autostart des auf der Diskette enthaltenen Anwenderprogramms wäre es auf diese Weise möglich, eine - für den normalen Anwender - kopiergeschützte Diskette herzustellen.

Der Datenbereich einer CP/M-Diskette

Alle übrigen Spuren der Diskette gehören zum Datenbereich: In diesem werden die Dateien abgelegt. Der Datenbereich der Diskette ist logisch in Blöcke eingeteilt: Jeder Block der Diskette kann höchstens zu einer Datei gehören. Endet eine Datei einmal vor dem Blockende, so bleibt der Rest des Blockes ungenutzt. Er wird also nicht vom Anfang der nächsten Datei aufgefüllt. Eine neue Datei beginnt so immer an einem Blockanfang. Der effektive Speicherbedarf einer Datei ist so immer ein Vielfaches der Blocklänge: Auch wenn die Datei nur ein einziges Byte lang ist, wird für sie ein ganzer Block reserviert.

Die Blocklänge ist eine rein logische Einheit und hat nichts mit den physikalischen Eigenschaften der Diskette zu tun. Sie kann also im Rahmen des Standards völlig frei gewählt werden und braucht nicht etwa mit der Sektorlänge zusammenfallen oder besonders elegant mit der Speicherkapazität einer Diskettenspur zusammenpassen. Die Mindestblocklänge für kleine Disketten ist 1 KByte. Dies ist auch die Blocklänge des 3"-Diskettenlaufwerkes. Disketten, die mehr als 256 Blöcke besitzen, haben eine Mindestblocklänge von 2 KByte. Erlaubt sind außerdem noch 4 KByte, 8 KByte und 16 KByte. Die Blöcke einer Diskette sind vom Anfang der Diskette bis zum Ende mit 0 beginnend durchnumeriert.

Das Inhaltsverzeichnis der Diskette

Die ersten Blöcke einer Diskette sind für das Inhaltsverzeichnis reserviert. Jeder Inhaltsverzeichniseintrag belegt 32 Bytes und beinhaltet außer einigen speziellen Informationen den Dateinamen und die Nummern der von der Datei belegten Blöcke. Aus dem gesamten Inhaltsverzeichnis können also die noch unbelegten Blöcke ermittelt werden. In einem 1 KByte-Block finden 32 Inhaltsverzeichniseinträge Platz. Für die 64 möglichen Einträge des Schneider 3"-Diskettenlaufwerkes sind also die ersten beiden Blöcke reserviert. Durch die Reservierung von weiteren Blöcken könnte das Inhaltsverzeichnis aber ohne weiteres erweitert (und auch eingeschränkt) werden.

Die Verwaltung der Disketten unter CP/M 2.2

Ständig aktiviert ist in einem CP/M-Computer nur das Diskettenlaufwerk A. Der Befehl STAT zeigt deshalb in der Regel nur die freie Speicherkapazität des Laufwerkes A an - auch wenn schon eine Diskette ins Laufwerk B eingelegt wurde.

Aktiviert wird ein Diskettenlaufwerk erst, wenn es benötigt wird. Dafür gibt es genau zwei Möglichkeiten:

1. Das Diskettenlaufwerk wird zum Bezugslaufwerk erklärt, etwa durch B:.
2. Auf ein Diskettenlaufwerk wird ohne Vorwarnung ein Zugriff durchgeführt.

Eine Diskette enthält kein unmittelbar lesbares Verzeichnis der belegten und freien Blöcke. Beim Aktivieren beziehungsweise Einloggen einer Diskette wird deshalb zunächst das gesamte Inhaltsverzeichnis gelesen und anhand der in den verschiedenen Dateien gefundenen Blockreservierungen eine zusammengefaßte Belegungstabelle erstellt. Außerdem wird über alle Inhaltsverzeichniseinträge eine Prüfsumme berechnet und im Speicher abgelegt.

Wenn Sie etwa erstmals auf das Laufwerk B umschalten, beginnt zunächst das Laufwerk B zu laufen. Daran erkennen Sie, daß zum Aktivieren wirklich einige Informationen von der Diskette geholt werden. Nach STAT wird jetzt der freie Speicherplatz beider Disketten angezeigt. Jetzt können Sie beliebig zwischen den beiden Laufwerken hin- und herschalten, ohne daß sie sich erst zu drehen beginnen.

Sie haben sicher bemerkt, daß das Lesen des Inhaltsverzeichnisses ungewöhnlich lange dauert, wenn gleichzeitig auch die Belegungstabelle erstellt wird. Wenn diese also bei jedem Diskettenzugriff neu erstellt werden müßte, würde das CP/M-Betriebssystem ziemlich langsam arbeiten. Deshalb wird die Belegungstabelle wirklich nur beim Aktivieren eines Laufwerkes erstellt. Das können Sie leicht nachprüfen:

- Tippen Sie 'STAT' ein und merken Sie sich die erhaltenen Angaben für den freien Speicherplatz.
- Wechseln Sie jetzt die Diskette aus und geben Sie 'DIR' ein. Im Inhaltsverzeichnis erscheint der neue Inhalt.
- Tippen Sie jetzt wieder 'STAT' ein und vergleichen Sie die erhaltenen Daten mit den gemerkten. Sie sehen, es wird immer noch die alte Kapazität angegeben.

Das nur einmalige Erstellen der Belegungstabelle hat aber für Sie die unangenehme Konsequenz, daß nur nicht aktivierte Disketten ausgewechselt werden dürfen. Wenn Sie sich nicht daran halten, würde das Betriebssystem auf die neue Diskette anhand der alten Belegungstabelle schreiben. Das hätte verheerende Folgen: Auf Ihrer neuen Diskette könnten wichtige Daten überschrieben werden, wenn die falsche Belegungstabelle an dieser Stelle einen leeren Block signalisiert.

Eine beliebte Unachtsamkeit selbst des erfahrenen CP/M-Anwenders ist es, bei großer Hektik versehentlich eine aktivierte Diskette auszuwechseln. Die Entwickler des CP/M 2.2-Betriebssystemes bauten deshalb eine Sicherheitsmaßnahme ein: Bei jedem Zugriff auf das Inhaltsverzeichnis der Diskette wird eine Prüfsumme über den Verzeichnisinhalt berechnet und mit der gespeicherten verglichen. Ergibt sich ein Unterschied, wurde die Diskette verbotenerweise gewechselt. Um Ihre Dateien vor Zerstörung zu schützen wird die Diskette in diesem Fall in den Nurlese-Status gesetzt. Beim nächsten Versuch, auf die Diskette zu schreiben, wird dann eine BDOS-Fehlermeldung ausgegeben. Das Lesen von der ausgewechselten Diskette funktioniert aber noch einwandfrei und ist auch erlaubt.

Wenn Sie eine Diskette wechseln wollen, müssen Sie sie zunächst deaktivieren. Die vorerst einzige Möglichkeit das zu tun ist, einen Warmstart auszuführen, denn bei jedem Warmstart wird das gesamte Diskettensystem zurückgesetzt. Die meisten CP/M-Programme führen unmittelbar vor dem Abschluß einen Warmstart aus. Wenn Sie sich aber nicht sicher sind, ob das letzte Programm einen Warmstart ausgeführt hat oder nicht, können Sie diesen Warmstart auch manuell durch ^C erzeugen. Danach können Sie die Diskette im Laufwerk B vorschriftsmäßig wechseln.

Ein Problem ist noch das Wechseln der Diskette im Laufwerk A, denn diese wird ja sofort nach dem Deaktivieren automatisch wieder aktiviert. In diesem Fall müssen Sie ^C nach dem Wechseln der Diskette eingeben, damit die Neuaktivierung gleich mit der neuen Diskette erfolgt. Das schadet nichts, denn zu Beginn des Warmstarts wird ja nichts auf die (vermeintlich noch alte) Diskette geschrieben.

Die Organisation des Inhaltsverzeichnis unter CP/M

In den ersten Blöcken der Spur 2 befindet sich wie schon erwähnt das Inhaltsverzeichnis der Diskette. Für jeden Verzeichniseintrag sind dabei 32 Bytes reserviert. Durch die spezielle Organisation können damit Dateien bis zu einer Größe von 8 MegaByte erfaßt werden. Enthalten sind aber auch Informationen über die tatsächliche Größe, die Usernummer der Datei und den Schreib-/Listschutz.

Der physikalische Extent

In 16 Bytes einen Verzeichniseintrages stehen die Nummern der Blöcke, die von der Datei belegt werden. Bei kleinen Disketten, die maximal 256 Blöcke besitzen, belegt jede Blocknummer 1 Byte. Bei größeren Disketten, die mehr als 256 Blöcke besitzen, belegt jede Blocknummer 2 Bytes. Mit den 16 Bytes können also bei kleinen Blöcken maximal 16 Blöcke, bei größeren Disketten sogar nur 8 Blöcke pro Verzeichniseintrag für eine Datei reserviert werden.

Auf der Schneider 3"-Diskette mit ihren 171 Blöcken zu je 1 KByte kann also pro Verzeichniseintrag höchstens 16 KByte als Bestandteil einer Datei registriert werden. Bei anderen Disketten können diese Werte anders sein. Die Vortex-Floppydisk etwa, die eine Blockgröße von 4KByte besitzt, kann 64 KByte pro Verzeichniseintrag registrieren. Unter CP/M können Dateien aber einen Umfang von bis zu 8 MB annehmen (natürlich nur bei entsprechender Diskettengröße).

Die zusätzlichen Informationen, die zum vollständigen Erfassen einer Datei nötig sind, werden deshalb unter CP/M 2.2 in einem weiteren Verzeichniseintrag abgelegt, der unter demselben Namen wie der erste zu finden ist. Dieser enthält dann in den 16 Blockreservierungsbytes die Lage von weiteren zur Datei gehörenden Blöcke. Bei der Schneider Floppydisk finden sich also zum Beispiel zu einer 50 KByte langen Datei auf der Diskette 4 Verzeichniseinträge unter demselben Namen. Beim DIR-Befehl wird aber nur ein einziger davon angezeigt. So kann es durchaus vorkommen, daß eine Diskette, die etwa nur 50 Dateien enthält bereits keine weiteren mehr aufnehmen kann.

Die verschiedenen Verzeichniseinträge zu einer einzigen Datei heißen 'physikalische Extents'. Jeder Extent enthält an einer bestimmten Position eine Nummer, die ihn von den anderen Extents zur selben Datei unterscheidet. Beim Aufsuchen einer bestimmten Datei geht das Betriebssystem dabei so vor:

Zunächst wird der erste (und eventuell einzige) Eintrag der Datei gesucht. In diesem ist auch die Anzahl der tatsächlich in diesem Eintrag registrierten Records. Ist diese Anzahl gerade der maximal erlaubte Wert, so weiß das Betriebssystem, daß zur Datei noch ein weiterer Extent vorhanden sein muß. Bei Bedarf kann dieser ebenfalls anhand seiner Nummer aufgesucht werden. Das ganze wird solange fortgesetzt, bis einmal der ganze Rest der Datei in einen einzigen Extent paßt.

Der logische Extent

Um auf der Ebene des Programmierers Verwirrung zu vermeiden, gibt es noch eine weitere Einheit: den logischen Extent. Ein logischer Extent hat unabhängig von der sonstigen Diskettenorganisation eine Größe von genau 16 KByte. Beim Schneider 3"-Laufwerk stimmt ein logischer Extent zufällig mit dem physikalischen Extent überein. Bei Disketten, die etwa 64 KByte lange physikalische Extents haben, sind aber jeweils 4 logische Extents in einem physikalischen Extent zu finden.

Die Dateigröße ist dann folgendermaßen angegeben: Als Extentnummer steht in einem Verzeichniseintrag die Nummer des letzten logischen Extents, der noch sinnvolle Daten enthält. Alle vorhergehenden logischen Extents sind vollständig mit 16 KByte gefüllt. Angegeben werden muß also nur noch der Umfang des letzten logischen Extents. Dieser letzte kann zwischen 0 und 128 Records beinhalten - und genau diese Anzahl wird im Verzeichniseintrag auch angegeben. Mit 128 Records (= 128*128 Bytes=16 KByte) ist der logische Extent vollständig gefüllt. Die Zahl 128 im Recordzähler signalisiert also, daß noch ein weiterer Extent vorhanden sein muß.

Im Grenzfall, wenn der 128. Record gleichzeitig das Dateiende ist, erhält eine Datei also einen weiteren Extent, obwohl alle belegten Blöcke schon in diesem Extent registriert sind. Der neue Extent ist dann vollständig leer und belegt 'unnötig' Diskettenspeicherplatz (wirklich unnötig ist er ja nicht). Dieser Fall tritt aber sehr selten auf, so daß Sie getrost damit leben können.

Ein Recordzähler mit dem Wert 128 tritt nur am Ende eines physikalischen Extents auf. Wenn der Wert 128 am Ende eines logischen Extents, aber noch in der Mitte des physikalischen Extents auftritt, wird ja der Recordzähler einfach auf Null gesetzt und der Extentzähler um den Wert 1 erhöht. Der allererste logische Extent in einer Datei hat dabei die Nummer 0.

Im neuen physikalischen Extent wird die Zählung da fortgesetzt, wo sie im alten Extent aufgehört hat. Die logischen Extents werden ununterbrochen weitergezählt. In einem System, das physikalische Extents der Länge 64 KByte verwendet, beginnt die Zählung zum Beispiel im 2. physikalischen Extent mit der logischen Extentnummer 4. Wenn diese etwa insgesamt einen Umfang von 113 KByte, also 8 Records zu je 128 Byte + 7 logische Extents zu je 16 KByte besitzt, ist im 2. Inhaltsverzeichniseintrag der Wert 7 als Extentnummer und der Wert 8 als Länge des letzten Extents eingetragen.

Zum besseren Verständnis zeigt Bild 5-3 einige Beispiele von Dateigrößen im Zusammenhang mit den dazugehörigen Einträgen im Inhaltsverzeichnis

der Diskette. Andere Informationen als die angegebenen sind in den Verzeichniseinträgen zur Dateigröße nicht enthalten. Selbstverständlich kommen aber zu jedem Eintrag jeweils noch die Nummern der belegten Blöcke dazu.

Dateigröße		Werte in den Verzeichniseinträgen 1. physik. Extent		2. physik. Extent	
Kilo-Bytes (gerundet)	Records	Extent-zähler	Record-zähler	Extent-zähler	Record-zähler
a) Disketten mit 64 KByte langen physikalischen Extents					
15	120	0	120	-	-
16	127	0	127	-	-
16	128	1	0	-	-
17	129	1	1	-	-
17	136	1	8	-	-
63	504	3	120	-	-
64	511	3	127	-	-
64	512	3	128	4	0
65	513	3	128	4	1
65	520	3	128	4	8
80	639	3	128	4	127
80	640	3	128	5	0
127	1016	3	128	7	120
128	1023	3	128	7	127
b) Disketten mit 16 KByte langen physikalischen Extents					
0	0	0	0	-	-
1	1	0	1	-	-
15	120	0	120	-	-
16	127	0	127	-	-
16	128	0	128	1	0
17	129	0	128	1	1
17	136	0	128	1	8
31	248	0	128	1	120
31	254	0	128	1	126
31	255	0	128	1	127

Bild 5-3: *Dateigrößen und Verzeichniseinträge unter CP/M 2.2*

Sie sehen, sobald der 2. physikalische Extent begonnen wurde bleibt der 1. physikalische Extent trotz Anwachsen der Datei unverändert. Bei noch längeren Dateien bliebe auch der 2. Extent unverändert und nur der 3. ändert sich mit der Dateilänge. Die Zählung würde analog zu den ersten beiden Extents fortgesetzt. Bei 64 KByte langen physikalischen Extents würde also der 2. Extent beim Wert 7/128 stehenbleiben und der 3. Extent mit dem Wert 8/0 beginnen. Bei Disketten, die mit 16 KByte langen physikalischen Extents verwaltet werden, bleibt die Zählung des 2. physikalischen Extents bei 1/128 stehen und die Zählung des 3. beginnt mit 2/0. Theoretisch möglich sind außer diesen beiden Beispielen auch noch die physikalischen Extentgrößen 32 KByte, 128 KByte und 256 KByte.

Bei Dateien, die sequentiell erzeugt wurden, sind alle Extents der Datei lückenlos im Verzeichnis enthalten. Eine Datei, die mit wahlfreiem Zugriff erzeugt wurde, kann aber auch 'Löcher' enthalten: Vollständig leere Extents fehlen im Verzeichnis. Die Inhalte der vorhandenen Einträge entsprechen aber trotzdem genau denen, die beim sequentiellen Schreiben erzeugt wurden.

Ein physikalischer Extent muß mindestens einen logischen Extent aufnehmen, also die Nummern der Datenblocks für 16 KByte Daten speichern können. Bei großen Disketten, die mehr als 256 Blöcke besitzen, kann das Betriebssystem nur 8 Datenblöcke pro Extent für die Datei reservieren. In diesem Fall beträgt die Mindestblocklänge 2 KByte.

Aufbau eines Inhaltsverzeichniseintrages

Ein 32 Byte langer Directoryeintrag ist so aufgebaut, wie im Bild 5-4 gezeigt ist.

```
             +--+--+    +--+--+--+--+--+--+--+--+--+    +--+
Bezeichnung  |UN|F1| .. |F8|T1|T2|T3|EX|S1|S2|RC|D0| .. |DF|
             +--+--+    +--+--+--+--+--+--+--+--+--+    +--+
Byte          00 01  ..  08 09 0A 0B 0C 0D 0E 0F 10  ..  1F
```

Bild 5-4: *Aufbau eines Directoryeintrages*

Die einzelnen Abkürzungen haben dabei die folgenden Bedeutungen:

UN — Die 'Usernummer':
In diesem Byte ist die Benutzernummer verschlüsselt, zu der die Datei gehört. Hat dieses Byte einen Wert zwischen 0 und 15, so handelt es sich um eine gültige Datei, die dem angegebenen Userbereich zugeordnet ist. Hat dieses Byte aber den Wert E5h, so handelt es sich um eine gelöschte Datei. Die Daten in den Bytes 01 bis 1F sind dann ungültig.
Beim Löschen einer Datei wird nur dieses eine Byte verändert. Durch Zurücksetzen auf den alten Wert können Sie eine versehentlich gelöschte Datei wieder restaurieren. Das ist zum Beispiel mit einem Diskmonitor möglich. Das ist aber immer mit einem Risiko verbunden: Wenn Sie nach dem Löschen schon etwas auf die Diskette geschrieben haben, kann es sein, daß zwar der Inhaltsverzeichniseintrag restauriert werden kann, die zugehörigen Datenblöcke aber schon wieder Daten anderer Dateien enthalten. Außerdem dürfen Sie nicht vergessen, eventuelle weitere Extents ebenfalls zu restaurieren.

`F1...F8` Der 'Filename':
Die unteren 7 Bits dieser 8 Bytes enthalten den Dateinamen in Ascii-Codierung. Die achten Bits werden vom Ascii-Code nicht verwendet und stehen für andere Daten zur Verfügung. Diese 8. Bits sind so belegt:

`F1'..F4'` Diese 4 Bits werden nicht benötigt. Sie können sie deshalb selbst setzen/löschen, um für besondere Zwecke eine Markierung in der Datei anzubringen.

`F5'..F8'` Diese 4 Bits sind für eine eventuelle spätere Verwendung von Digital Research reserviert.

`T1...T3` Der 'Filetype':
Die unteren 7 Bits dieser 3 Bytes enthalten den Dateityp in Ascii-Großschrift. Wie beim Dateinamen haben auch hier die 8. Bits andere Bedeutungen:

`T1'` Ein gesetztes Bit an dieser Stelle bedeutet, daß die Datei schreibgeschützt ist.

`T2'` Ein gesetztes Bit an dieser Stelle kennzeichnet einen SYS-File, der nicht im Inhaltsverzeichnis erscheint, aber normal verwendet werden kann.

`T3'` Dieses 8. Bit wird unter CP/M 2.2 nicht verwendet, ist aber für eine spätere Verwendung durch Digital Reearch reserviert.

`EX` Das 'Extent Byte':
Eine Datei kann bis zu 512 logische Extents besitzen (512 = 8 Megabytes/128 Bytes). Die Extensionnummer ist also eine 9-Bit-Zahl. Die niederwertigen 5 Bits dieser Nummer sind in diesem Byte enthalten.

`S1` Dieses Byte wird vom CP/M 2.2-Betriebssystem nicht benutzt. Spätere Betriebssystemversionen könnten aber von diesem Byte Gebrauch machen. Es ist also das sicherste, dieses Byte unberührt zu lassen.

`S2` Analog zum EX-Byte enthält dieses Inhaltsverzeichnis-Byte die 4 höherwertigen Bits der Verzeichnis-Extensionsnummer.

`RC` Das 'Record Count Byte':
Dieses Byte enthält die Anzahl der Records im letzten logischen Extent dieses Verzeichniseintrages. Es kann Werte zwischen 0 und 128 annehmen.

`D0..DF` Diese 16 Bytes enthalten die Nummern der Blöcke, die für die Datei in diesem Verzeichniseintrag reserviert sind. Kleine Disketten, die höchstens 256 Blöcke besitzen, haben 8-Bit-Blocknummern. Bei diesen Disketten können hier 16 Blöcke reserviert werden.

Größere Disketten haben 16 Bit lange Blocknummern. Deshalb ist dieser Bereich für diese in der Form von 8 je 16 Bit langen Worten organisiert. Bei diesen können nur 8 Blöcke in einem Verzeichniseintrag reserviert werden.

Die Diskettenbeschreiber unter CP/M 2.2

Für jedes in Ihrem Computersystem vorhandene Diskettenlaufwerk gibt es im BIOS eine Reihe von Tabellen und Zwischenspeicher, die Informationen über die Organisation dieses speziellen Diskettenlaufwerks enthalten. Das BDOS kann diese Informationen abfragen und so Besonderheiten in der Hardware der Diskettenlaufwerke abfangen. So wird (wieder einmal) erreicht, daß Anwenderprogramme absolut hardwareunabhängig sind.

Der Disk-Parameter-Header

Der Schlüssel zu den Unterlagen der Verwaltung eines Diskettenlaufwerkes ist eine 16 Bytes lange Tabelle, in der 8 Worte zu je 16 Bit enthalten sind. Für jedes Diskettenlaufwerk existiert im BIOS ein eigenes Exemplar dieser Tabelle und wird 'Disk-Parameter-Header' genannt. Deren Aufbau sehen Sie im Bild 5-5. Der DPH-Vektor (so wird der Name der Tabelle oft abgekürzt) muß zur Ausführungszeit unbedingt im RAM des Computers stehen. Sonst kann das System nicht richtig arbeiten.

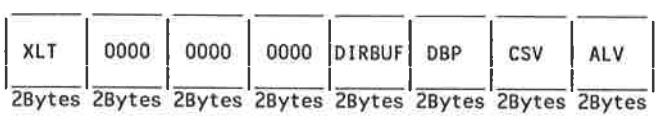


Bild 5-5: *Format des Disk-Parameter-Headers*

Die einzelnen Abkürzungen bedeuten dabei folgendes:

XLT — Adresse der Übersetzungstabelle zwischen logischen und physikalischen Recordnummern: Damit kann das BIOS die Reihenfolge herausfinden, in der die einzelnen Records auf der Diskette abgelegt sind. Wenn dieses 'Blocking und Deblocking' wie bei den CPC-Computern automatisch von den BIOS-Lese-Schreibroutinen durchgeführt wird, hat dieses Tabellenelement den Wert 0000h. Damit wird signalisiert, daß im System keine Übersetzungstabelle vorhanden ist.

0000 — Die 6 Bytes beziehungsweise 3 Worte, die in der Tabelle so gekennzeichnet sind, werden vom BDOS als Zwischenspeicher benützt. Normalerweise wird darin die Spurnummer, die Recordnummer und die Lage auf der Diskette des im

DIRBUF zwischengespeicherten Inhaltsverzeichnisausschnittes festgehalten. Das geschieht durch die Angabe, wieviele Records der gespeicherte Eintrag vom Beginn des Inhaltsverzeichnisses entfernt ist. Die Anfangswerte diser 6 Bytes sind beliebig: Vor Gebrauch werden Sie vom BDOS automatisch mit den richtigen Werten versorgt.

DIRBUF — In diesen beiden Bytes ist die Adresse eines 128 Bytes langen frei verfügbaren RAM-Speichers enthalten. Dieser RAM-Speicher wird bei Inhaltsverzeichnisoperationen vom BDOS zur Zwischenspeicherung des gerade behandelten Eintrages verwendet. Dieser Zwischenspeicher braucht auch wenn Sie mehrere Laufwerke besitzen nur ein einziges Mal im System vorhanden zu sein, weil das BDOS immer nur ein einziges Laufwerk zur gleichen Zeit behandeln kann. Die DIRBUF-Zeiger haben deshalb in allen Disk-Parameter-Headern denselben Wert.

DPB — Diese beiden Bytes enthalten die Anfangsadresse des Disk-Parameter-Blocks. Dieser DPB ist selbst eine Tabelle, in der alle wichtigen Laufwerkseigenschaften wie Inhaltsverzeichnis- und Speicherplatzgröße aufgelistet sind, damit sich das BDOS darauf einstellen kann. Dessen genauer Aufbau wird später noch beschrieben. Wenn Sie mehrere Laufwerke mit völlig gleichen Eigenschaften angeschlossen haben, dürfen sich diese einen einzigen Disk Parameter Block teilen. Die Zeiger in den verschiedenen Disk-Parameter-Headern zeigen dann eben auf ein- und dieselbe Tabelle.

CSV — Diese beiden Bytes enthalten die Anfangsadresse eines unbelegten RAM-Speicherbereiches, in dem das BDOS die Prüfsumme für das Inhaltsverzeichnis der Diskette ablegen darf. Durch spätere Bezugnahme auf diese Prüfsumme, kann ein unerlaubter Diskettenwechsel erkannt werden. Für jedes vorhandene Laufwerk muß ein eigener Prüfsummenzwischenspeicher reserviert werden. Die Zeiger in den verschiedenen DPH's müssen also auch auf verschiedene Speicherbereiche zeigen.

Für jeden Record des Inhaltsverzeichnisses, der in der Prüfsumme berücksichtigt wird, muß in diesem Zwischenspeicher ein Byte reserviert werden. Bei der Schneider-Floppydisk werden alle 64 Einträge überprüft. Diese belegen 64*32/128=16 Records zu je 128 Byte. Der Prüfsummenzwischenspeicher muß also 16 Bytes lang sein. Werden bei der Prüfsummenberechnung nur ein Teil der Inhaltsverzeichniseinträge berücksichtigt (was möglich ist) so darf der

Zwischenspeicher ebenfalls um die entsprechenden Bytes gekürzt werden. Werden gar keine Einträge überprüft (wie bei einer Festplatte oder einer RAM-Disk), so haben die beiden CSV-Bytes den Wert 0000.

ALV Diese beiden Bytes zeigen auf einen freien RAM-Bereich, der die Blockbelegungstabelle der eingelegten Diskette aufnehmen kann. In dieser Tabelle ist für jeden Block der Diskette ein Bit reserviert, das vom BDOS verändert werden kann. Ein gesetztes Bit zeigt an, daß der entsprechende Block der Diskette schon belegt ist. Das Schneider 3"-Diskettenlaufwerk besitzt inclusive der Directory-Blöcke eine Speicherkapazität von 171 Blöcken. Die Blockbelegungstabelle muß also Platz für 171 Bits bieten, das sind 22 Bytes. Diskettenlaufwerke mit größeren Kapazitäten brauchen einen entsprechend vergrößerten Tabellenbereich. Jedes am System angeschlossene Laufwerk benötigt seine eigene Blockbelegungstabelle. Die verschiedenen ALV-Zeiger in den verschiedenen DPH-Vektoren müssen also auf verschiedene Speicherbereiche zeigen.

Die Disk-Parameter-Header befinden sich für alle Laufwerke im RAM des Computers. Durch verändern der Zeiger in einem Disk-Parameter-Header können Sie die dadurch adressierten Tabellen durch eigene ersetzen. Besonders interessant ist dabei der Disk-Parameter-Block: Wenn Sie diesen durch eine Kopie im RAM ersetzen, können Sie die softwaremäßigen Laufwerkseigenschaften durch eigene Eigenschaften ersetzen.

Die Disketten-Parameter-Tabelle

Im Disk-Parameter-Block stehen für das BDOS bestimmte Informationen über die logische und physische Organisation eines Diskettenlaufwerkes. Für jedes Diskettenformat, das im System verwendet wird, gibt es einen eigenen Disk-Parameter-Block. Identische Diskettenlaufwerke brauchen aber nur einen gemeinsamen Disk-Parameter-Block.

Der Standard-Disk-Parameter-Block

Die in dieser Stufe angegebenen logischen Charakteristika des Floppydisk-Laufwerkes können vom Programmierer - innerhalb bestimmter Grenzen - frei bestimmt werden. Das BDOS richtet sich dann automatisch nach dieser Einstellung. Das Schneider 3"-Laufwerk arbeitet zum Beispiel mit 1 KByte langen Blöcken. Wenn Sie keinen Wert auf Kompatibilität mit anderen CPC-Benutzern legen, können Sie das System durch einfaches Ändern eini-

ger Bytes im Disk-Parameter-Block aber auch auf 2 KByte lange Blocks umstellen. Andere Vorbereitungen oder Eingriffe sind dazu nicht nötig!

Der 15 Bytes lange Disk-Parameter-Block ist so aufgebaut, wie das Bild 5-6 zeigt.

Bild 5-6: *Der Disk-Parameter-Block*

Dabei bedeuten die Abkürzungen folgendes:

SPT — Sectors Per Track (2 Bytes)
In diesen beiden Bytes steht die Gesamtzahl der Records pro Spur. Eine Spur einer 3"-Diskette hat 9 Sektoren zu je 512 Bytes, wobei jeder Sektor 4 Records zu je 128 Bytes enthält. Im BIOS finden Sie also in den SPT-Bytes den Wert 36.

BSH — Block Shift Factor (1 Byte)
In diesem Byte ist die Blockgröße definiert, mit der die Diskette verwaltet wird. Die Zahl 2^(BSH) gibt die Anzahl der 128-Byte-Records an, die in einem Block enthalten sind. Für eine Blockgröße von 1 KByte enthält dieses Byte also den Wert 3 (1024=128*2^3), für eine Blockgröße von 2 KByte den Wert 4 (2048=128*2^4) usw. Der maximale BSH-Wert ist 7 für Disketten mit 16 KByte langen Blöcken (16384=128*2^7). Mit welcher Blockgröße Sie Ihre Disketten verwalten wollen, ist ganz allein Ihre Entscheidung. Durch eine einfache Änderung dieses Bytes erreichen Sie, daß Ihre Dateien im neuen Format abgespeichert werden. Beachten Sie aber, daß einige weitere Elemente im Disk-Parameter-Block vom Wert dieses Bytes abhängen. Eventuell müssen Sie auch für die Blockbelegungstabelle einen größeren Speicherplatz definieren.

BLM — Block Mask (1 Byte)
Der Wert dieses Bytes hängt direkt vom Wert des BSH-Vektors ab. Im BLM-Byte sind soviele Bits gesetzt, wie der Wert des BSH-Bytes angibt. Für eine Blockgröße von 4 KByte hat das BSH-Byte den Wert 5. Im BLM-Byte sind also 5 Bits gesetzt und es hat den Wert 31. Diesen Wert erhalten Sie auch durch die Formel BLM=(2^BSH)-1.

EXM — Extent Mask (1 Byte)
Das Byte enthält die um 1 verminderte Zahl der logischen Extents, die in einem physischen Extent beziehungsweise Verzeichniseintrag enthalten sind. Hat die Diskette etwa weniger als 256 Blöcke, so können in einem physischen Extent die Nummern von 16 Blöcke vermerkt werden. Bei einer Blockgröße von 4 KByte enthält ein physischer Extent die Reservierungen für 64 KByte oder 4 logische Extents. Das EXM-Byte hat also in diesem Fall den Wert 3 (= 4-1). Hat die Diskette aber mehr als 256 Blöcke, so können nur noch 8 Blöcke beziehungsweise in diesem Fall 32 KByte in einem physischen Extent reserviert werden. Das entspricht 2 logischen Extents. Das EXM-Byte hat in diesem Fall den Wert 1 (=2-1). Der Wert des EXM-Bytes hängt also ab vom Wert des BSH-Bytes und von der Anzahl der Blöcke, die auf der Diskette gespeichert werden können.

DSM — Disk Size Mask (2 Bytes)
Diese 16-Bit-Zahl ist die um 1 verminderte Zahl der Blöcke, die die Diskette enthält. Gezählt werden dabei auch die Blöcke, die vom Inhaltsverzeichnis belegt werden, nicht aber die Systemspuren. Eine 3"-Diskette enthält 38 Datenspuren zu je 4,5 Blöcken. Die DSM-Bytes haben deshalb da einen Wert von 170, der sich aus 4,5*38-1 errechnet. Diesen Wert können Sie ohne weiteres erniedrigen: Die Blöcke mit größeren Nummern als der Grenzwert werden dann vom Betriebssystem einfach nicht mehr benützt. So können Sie ein Diskettenformat mit einer geringeren virtuellen Kapazität erzeugen. Die nicht registrierten Bereiche der Diskette können Sie dann für eigene Zwecke verwenden. Der Eintrag eines größeren Wertes in die DSM-Bytes ist jedoch nicht zu empfehlen: Das BDOS würde sonst versuchen, eine nicht vorhandene 41. Spur anzusteuern.

DRM — Directory Mask (2 Bytes)
Diese 16-Bit-Zahl enthält die um 1 verminderte Anzahl der Inhaltsverzeichniseinträge, die auf der Diskette verwaltet werden können. Bei den 3"-Laufwerken enthalten die DRM-Bytes also den Wert 003Fh. Diesen Wert können Sie ebenfalls nach eigenem Ermessen ändern. Achten Sie aber darauf, daß noch einige andere Bytes im Disk-Parameter-Block von dieser Änderung beeinflußt werden.

AL0,AL1 — (jeweils 1 Byte)
Diese beiden Bytes geben die Anzahl der Blöcke an, die für das Inhaltsverzeichnis reserviert sind. Die beiden werden

beim Aktivieren einer Diskette direkt in die Blockbelegungstabelle übertragen, so daß die in den AL0/1-Bytes gesetzten Bits Blöcke repräsentieren, die schon von vornherein als belegt gelten. Die reservierten Blöcke können dann das Inhaltsverzeichnis aufnehmen. Das höchstwertige Bit von AL0 repräsentiert den Block 0, das niederwertigste Bit den Block 7. Die Bits des AL1-Bytes repräsentieren die Blöcke 8 bis 15. Beim 3"-Laufwerk belegt das Inhaltsverzeichnis 2 KByte, also 2 Blöcke. Das AL0-Byte hat also den Wert C0h und das AL1-Byte den Wert 00h.
Sie können ohne weiteres weitere Blöcke der Diskette reservieren, die Sie dann für besondere Zwecke verwenden dürfen. Sie sollten sich aber davor hüten, weniger Blöcke zu reservieren, als das Inhaltsverzeichnis benötigt: Der erste Datenblock und der letzte Inhaltsverzeichnis-Block werden dann auf denselben Ort der Diskette geschrieben. Das hat in der Regel sehr unangenehme Folgen.

CKS — Directory Check Vector (2 Bytes)
Die 16-Bit-Zahl CKS ist ein Maß für die Anzahl der Directoryeinträge, über die gelegentlich eine Prüfsumme berechnet wird um einen unerlaubten Diskettenwechsel festzustellen. Angegeben ist jedoch nicht direkt die Anzahl der geprüften 32 Byte-Einträge, sondern die Anzahl der überprüften 128 Byte-Records, die Directoryeinträge enthalten. CKS enthält also nur ein Viertel der Eintragszahl. Sollen etwa 64 Verzeichniseinträge in die Prüfsumme einfließen, enthält CKS nur den Wert 16. Bei auswechselbaren Speichermedien wie Disketten wird man in der Regel alle Verzeichniseinträge überprüfen. Die Verzeichniseinträge von Festplatten brauchen aber überhaupt nicht überprüft werden. Ein unerlaubtes Wechseln ist da ja nicht möglich. CKS hat in diesem Fall den Wert 0. Erlaubt sind aber auch beliebige Werte zwischen diesen beiden Extremen.

OFF — Offset (2 Bytes)
Diese Zahl gibt die Anzahl der für das Betriebssystem reservierten Spuren an. In der Regel werden das 2 Spuren sein. Andere Werte werden aber auch vom Standard abgedeckt. Disketten im Laufwerk B benötigen zum Beispiel nicht unbedingt eine Systemspur. Bei diesen kann der Spuroffset auch 0 sein. Große Festplatten haben meistens nur eine - dafür aber längere - Systemspur.

Der erweiterte Disk-Parameter-Block

Der CP/M 2.2-Standard verlangt nur diesen eben vorgestellten 15 Bytes langen Disk-Parameter-Block. Gewöhnliche CP/M-Programme beziehen sich also nur auf die hier abgelegten Parameter. Im BIOS des Schneider 3"-Laufwerk besitzt der DPB aber noch eine 10 Bytes lange Erweiterung. Der gesamte DPB hat hier also eine Länge von 25 Bytes.

Diese Erweiterungsbytes enthalten detaillierte hardwareorientierte Informationen über die Formatierung einer Diskette. Zu der Änderung der meisten benötigen Sie genaue Informationen über den Aufbau der Hardware Ihres Computers und der in den ROMs eingebauten Software. Die meisten Bytes sind also höchstens interessant, wenn Sie sich ein eigenes Betriebssystem erstellen wollen. Für Sie nützlich sind nur 2 Bytes dieser Erweiterung:

Sektor-Offset

Das erste Byte des erweiterten DPB ist das 16. Byte des gesamten DPB. Dieses gibt an, wie die Sektoren auf der Diskette hardwaremäßig numeriert sind. Auf einer normalen 3"-CP/M-Diskette haben die Sektoren Nummer zwischen 41h und 49h. Das Sektor-Offset-Byte hat den Wert der ersten Sektornummer, also hier 41h. Bei einer Diskette, die im Datenformat formatiert wurde, beginnen die Sektornummern mit dem Wert C1h und bei einer Diskette im IBM-Format mit dem Wert 01h. Wenn Sie diese Werte eintragen, können Sie (unter bestimmten weiteren Voraussetzungen) also auch Disketten im IBM oder Datenformat unter CP/M lesen.

Anzahl der physischen Sektoren

Wenn Sie eine Diskette mit einer anderen Sektorzahl pro Spur lesen wollen, genügt es in den Schneider-Computern nicht, nur die SPT-Bytes des Disk-Parameter-Blocks zu ändern. Angepaßt werden muß auch das 2. Byte des erweiterten Disk-Parameter-Blocks, beziehungsweise in anderer Zählung das 17. Byte des Standard-DPBs. Dieses Byte enthält die Anzahl der physischen Sektoren pro Spur. Bei einer Diskette im System- oder Datenformat also den Wert 9, bei einer Diskette im IBM-Format den Wert 8.

Die anderen 8 Bytes können nicht ohne weiteres verändert werden, da sie zu sehr auf die interne Struktur des Rechners Bezug nehmen. Sie sollten sich bewußt sein, daß Programme, die den erweiterten Disk-Parameter-Block verwenden, nur auf einem Schneider-Computer mit 3"-Laufwerk ablauffähig sind. Änderungen im Standard Disk Parameter Block sind aber auch zu allen anderen CP/M-Computern kompatibel - denn diese beziehen sich nur auf die virtuelle Verwaltung der Diskette, nicht aber auf die tatsächliche physische Struktur.

Wenn Sie einen selbstentworfenen Disk-Parameter-Block irgendwo im RAM ablegen, dann sollten Sie an diesen den erweiterten DPB unverändert anhängen - auch wenn Sie nicht wissen, was diese 10 Bytes genau bedeuten. Es könnte ja sein, daß irgendeine BIOS-Routine auf diese Bytes zugreift.

Genau dieselbe Erweiterung des Disk-Parameter-Blocks gibt es auch, wenn Sie einen Vortex-Floppydisk-Controller verwenden.

Regeln zum Verändern des DPB

Die Anzahl der Sektoren/Records pro Spur und die maximale Speicherkapazität einer Diskette können Sie nur durch 'Umformatieren' der Diskette ändern. Alle anderen oben angegeben Änderungen haben aber nur einen Einfluß auf die Interpretation der gelesenen Daten - nicht aber auf die physische Unterteilung der Diskette selbst.

Wenn Sie irgendwo im Disk-Parameter-Block eine Änderung einfügen, müssen Sie in der Regel auch einige unmittelbar daraus folgenden Änderungen durchführen. Besonders heimtückisch sind hier Änderungen der Blockanzahl der Diskette und der Anzahl der überprüften Inhaltsverzeichniseinträge: Wenn Sie diese Zahlen vergrößern, müssen Sie dem Prüfsummenspeicher und der Blockbelegungstabelle ebenfalls einen größeren Speicherbereich zuweisen. Deren Adressen können Sie im Disk-Parameter-Header finden. Durch eine Änderung der Zeiger auf die beiden Tabellen können Sie diese in einen Speicherbereich Ihrer Wahl legen.

Die Werte der Bytes BSH, BLM und EXM hängen voneinander ab. Wenn Sie also eines der Bytes ändern, müssen sie auch die anderen entsprechend anpassen. Die Folge einer Änderung ist eine neue Blockgröße, mit der die Dateien verwaltet werden. Die Blockgröße ist nirgends im System in 'Klarschrift' festgehalten. Es ist also schon wichtig, daß Sie sich die Wahl dieser Werte gut überlegen. Im Bild 5-7 ist der Zusammenhang zwischen Blockgröße, Diskettengröße und Wert im Disk-Parameter-Header festgehalten.

Block-größe (Bytes)	BSH-Wert	BLM-Wert	EXM-Wert	
			Gesamtblock-anzahl <=256	Gesamtblock-anzahl > 256
1024	3	7	0	(verboten)
2048	4	15	1	"
4096	5	31	3	1
8192	6	63	7	3
16384	7	127	15	7

Bild 5-7: *Disk-Parameter-Bytes in Abhängigkeit von der Blockgröße*

Experimente zur Diskettenverwaltung

Nach soviel Theorie dürfen Sie jetzt etwas exerimentieren. Ihre erste Aufgabe ist dabei, die Lage der Diskettenbeschreiber herauszufinden. Starten sie dazu zunächst das Programm DDT.COM.

Die Lage des Disk-Parameter-Headers für das Laufwerk A können Sie mit Hilfe der folgenden Routine herausfinden:

```
0100 mvi c,0
0102 lxi d,0018
0105 lhld 0001
0108 dad d
0109 call 0110
010C nop
010D nop
010F nop
0110 pchl
```

Diese Routine können Sie nach der Eingabe von 'A0100' direkt so eintippen, wie Sie da steht. Sie sollte mit dem Befehl 'g0100,010C' aufgerufen werden. Nach dem Durchlauf steht im CPU-Register HL die Anfangsadresse des Disk-Parameter-Headers des Laufwerks A. Durch die Eingabe von 'X' wird dieses Register (zusammen mit den anderen Registern) angezeigt. Wie diese Routine genau funktioniert, wird später noch erklärt. Wichtig ist jetzt nur, daß sie überhaupt funktioniert.

Die Zahl hinter dem Befehl 'MVI C,...' ist ein Code für das Laufwerk, dessen Disk-Parameter-Header ausgegeben werden soll. 0 steht für das Laufwerk A, 1 für das Laufwerk B usw.

Bei der in den CPC-Computern üblichen CP/M-Version sollte sich der DPH für das Laufwerk A an der Adresse AE58 befinden und der für das Laufwerk B an der Adresse AE68. Das 10./11. Byte des DPH enthält die Adresse des Disk-Parameter-Blocks DPB. In der Regel sollte sich der DPB des Laufwerkes A an der Adresse ADD8 befinden und der DPB des Laufwerkes B an der Adresse AE18. Wenn Sie eine geänderte CP/M-Version besitzen, können sich aber auch andere Werte ergeben. Das ist zum Beispiel dann der Fall, wenn Sie eine Speichererweiterung verwenden.

Mit Hilfe der D- und S-Befehle können Sie sich die Tabellen ansehen und bei Bedarf verändern.

Zum Experimentieren legen Sie sich am besten eine spezielle Diskette an. Wenn Sie 2 Diskettenlaufwerke besitzen, ist die Sache einfach: Sie ändern nur die Parametertabellen zum Laufwerk B ab. Mit Hilfe von PIP können Sie dann wie gewöhnlich vom Laufwerk A auf das Laufwerk B kopieren -

nur mit dem Unterschied, daß die Dateien auf der B-Diskette im neuen Format abgespeichert werden.

Wenn Sie nur ein Laufwerk besitzen, dann ist die Sache schon schwieriger. In diesem Fall sollten Sie die zu kopierende Datei mit 'DDT <Name>' in den Speicher holen. Dann ändern Sie - im DDT - die Diskettenparameter nach Ihren Wünschen ab. Jetzt legen Sie die Diskette, die Daten im neuen Format erhalten soll ein und unterbrechen den DDT mit ^C. Durch 'SAVE' wird die Datei <Name> gleich im neuen Format abgespeichert. Wenn Sie eine weitere Datei kopieren wollen, müssen Sie den Rechner aus- und wieder einschalten und die Prozedur mit der neuen Datei wiederholen. Auf diese Weise können Sie nach und nach alle Ihre Programme überspielen.

Am Anfang sollten Sie mit einfacheren Änderungen wie zusätzlichen Directoryeinträgen oder verminderter Blockanzahl beginnen. Sie werden bemerken, daß Sie in diesen Fällen ein und dieselbe Diskette immer noch unter beiden Formaten lesen können. Eine Ausnahme ist jedoch zum Beispiel die 65. Datei unter einer Verwaltung für 64 Inhaltseinträge: Die von dieser Datei belegten Blöcke werden dann irrtümlich als frei betrachtet. Eine einfache Erfolgskontolle haben Sie mit dem Befehl 'STAT DSK:'.

Später können Sie sich auch an schwierigere Aufgaben, wie die Änderung der Blockgröße machen. In diesem Fall werden Dateien, die unter der einen Verwaltung geschrieben wurden unter der anderen wirklich nicht mehr gelesen werden. Wenn Sie gerne etwas Besonderes haben, könnten Sie sich also auf eine Diskettenverwaltung umstellen, deren Disketten vollständig inkompatibel zu den Disketten sind, die unter den Betriebssystemen anderer Anwender erstellt wurden. Ihre Disketten haben dann so eine Art 'Kopierschutz'. In der Regel ist es aber sinnvoll, beim Standardformat zu bleiben.

Wenn Sie die Anzahl der Inhaltsverzeichniseinträge oder die Blockanzahl vergrößern, benötigen Sie zusätzlichen Pufferspeicherplatz für die Prüfsumme und die Belegungstabelle. Ein guter Platz dafür ist beim CPC 464 der Speicherbereich von B800h bis B8DBh und beim CPC 664/6128 der Bereich zwischen B118h und B1ECh. Diese Bereiche sind für die Fileheader bei der Kassettenausgabe reserviert und werden deshalb normalerweise unter CP/M nicht benötigt. Lediglich auf die Befehle CSAVE und CLOAD müssen Sie dann verzichten. Aber für die Dauer dieser Experimente dürfte Ihnen das nicht schwer fallen.

Nach Beendigung Ihrer Experimente sollten Sie die Experimentierdiskette unbedingt neu formatieren. Nur so vermeiden Sie böse Überraschungen, wenn Sie diese Diskette einmal mit einer 'normalen' Diskette verwechseln.

Lesen von Disketten im Daten- oder IBM-Format

Normalerweise ist es nicht möglich, eine 3"-Diskette, die Sie im Datenformat formatiert haben, unter CP/M auch im Laufwerk A zu verwenden: Bei jedem Versuch, dem Betriebssystem eine solche Diskette unterzuschieben gibt es nur Lesefehler. Dasselbe gilt auch für die Disketten im IBM-Format. Sie müssen also auf die komfortablen Befehle zur Dateipflege, die Ihnen CP/M bietet, verzichten. Durch einen Eingriff in die Diskettenparameterblöcke haben Sie aber vielleicht doch noch eine Chance, das Betriebssystem auf diese Disketten einzustellen.

Eine Datendiskette unterscheidet sich von einer CP/M-Systemdiskette vor allem durch das Fehlen der Systemspuren: Das Inhaltsverzeichnis beginnt schon auf der Spur 0. Der Einsatz einer Datendiskette im Laufwerk A ist damit unmöglich. Denn bei jedem Warmstart holt CP/M ja das BDOS und den CCP von den Systemspuren der Diskette A, unabhängig davon, welches gerade das Bezugslaufwerk ist. Wenn Sie bei der Arbeit Warmstarts vermeiden, schadet aber das Fehlen der Systemspuren nicht. Zu Spezialzwecken können sie hier also kurzzeitig die Systemspuren wegdefinieren. Für das Laufwerk A sind nur die Änderungen nach Bild 5-8 im Disk Parameter Block nötig. Angegeben sind nur die Bytes, die sich in den verschiedenen Formaten unterscheiden.

Parameter-Block-Bytes		Stan-	Werte bei der Anpassung		
Byte-Nummer	Bezeichung	dard-adresse	System-format	Daten-format	IBM-format
0/1	SPT(Rec/Track)	ADD8h	0024h	0024h	0020h
5/6	DSM(Blockzahl)	ADDDh	00AAh	00B3h	009Bh
14/15	OFF(Sys-spur)	ADE5h	0002h	0000h	0001h
16	Sektoroffset	ADE7h	41h	C1h	01h
17	Sektoren/Spur	ADE8h	09h	09h	08h

Bild 5-8: *Unterschiedliche Werte bei 3"-Formaten*

Eine erste Kontrolle der richtigen Einstellung ist mit 'STAT DSK:' möglich. Wenn das Laufwerk B zum Lesen einer 3"-Datendiskette vorbereitet ist, sollte sich ein Ergebnis wie in Bild 5-9 ergeben. Für das Laufwerk A können Sie selbstverständlich auch andere Werte erhalten.

Nach der Bearbeitung einer Datenformat-Diskette sollten Sie Ihren Computer kurz aus- und wieder einschalten. Bei einem eventuell nachfolgenden Lesen von einer normalen Diskette kommt es sonst zu unnötigen Lesefehlern.

```
A>stat dsk:

    A: Drive Characteristics
 5664: 128 Byte Record Capacity
  708: Kilobyte Drive  Capacity
  128: 32  Byte Directory Entries
   64: Checked  Directory Entries
  512: Records/ Extent
   32: Records/ Block
   36: Sectors/ Track
    2: Reserved Tracks

    B: Drive Characteristics
 1440: 128 Byte Record Capacity
  180: Kilobyte Drive  Capacity
   64: 32  Byte Directory Entries
   64: Checked  Directory Entries
  128: Records/ Extent
    8: Records/ Block
   36: Sectors/ Track
    0: Reserved Tracks
```

Bild 5-9: *Ergebnis von 'STAT DSK:'*

6 Das BDOS: die Benutzerschnittstelle

6.1 Allgemeines zum BDOS

Sie haben bis jetzt gelernt, wie man mit Hilfe des 8080-Assemblers oder Debuggers Maschinespracheprogramme schreibt und diese so auf der Diskette ablegt, daß sie wie gewöhnliche CP/M-Befehle aufgerufen werden können. Bis jetzt wurde aber mit noch keinem Wort erwähnt, wie man zum Beispiel eine Bildschirmausgabe produziert.

Auf den ersten Blick scheint es möglich, dafür einfach über die Sprungvektoren im Betriebssystem des CPC 464/664 die Firmwareroutine zur Zeichenausgabe aufzurufen. Das Problem dabei ist jedoch folgendes: Ein auf diese Art und Weise geschriebenes Maschinenprogramm wird nur auf dem Computertyp laufen, auf dem es geschrieben wurde. Das weiter oben erwähnte Prinzip der 'maximalen Kompatibilität' ist damit jedoch nicht erfüllt. Natürlich wissen die Entwickler von CP/M ebenfalls um diese Schwierigkeit.

Eine Sammlung standardisierter Funktionen

Das CP/M-Betriebssystem enthält deshalb eine Sammlung von im praktischen Betrieb häufig benötigten Operationen. Enthalten ist also nicht nur die Routine zur Ausgabe eines Zeichens am Bildschirm, sondern auch Routinen zum Lesen/Schreiben auf die Diskette, zum Suchen eines Directoryeintrages, zum Wechseln des gerade benützten Laufwerkes und so weiter... (insgesamt 39 Stück).

Das besondere daran ist, daß die Aufrufe dieser Funktionen standardisiert sind, das heißt die Funktionen stehen wirklich in genau der gleichen Weise auf allen Computertypen zur Verfügung, die den Namen 'CP/M' tragen. Das gilt ausdrücklich auch für die Routinen, die eigentlich von der Hardware des verwendeten Computers abhängen müßten. Jedes Programm, das alle vom Computertyp abhängigen Tätigkeiten mit Hilfe dieser standardisierten Aufrufe ausführt, ist so trotzdem auf jedem beliebigen CP/M-Computer lauffähig.

Das BDOS (Basic Disk Operating System) ist genau genommen nichts als die Sammlung dieser Routinen.

Aufruf und Rückgabeparameter

Die Ausführung jeder BDOS-Routine wird durch ein und denselben Call-Befehl eingeleitet: durch 'call 0005h' Zur Unterscheidung, welche BDOS-Funktion ausgeführt werden soll, ist jeder BDOS-Funktion eine Code-Nummer zugeordnet.

- Diese Codenummer muß vor jedem 'call 0005h' in das C-Register der CPU übertragen werden.
- Im DE-Registerpaar (für 16-Bit-Werte) oder im E-Register (für 8-Bit-Werte) können der Funktion Parameter zur Bearbeitung übergeben werden. Im Falle der Routine, die ein Zeichen auf dem Bildschirm ausgibt, wäre das also der Code des auszugebenden Zeichens.
- Jetzt kann 'call 0005h' ausgeführt werden.
- Das Ergebnis der Routine steht entweder im Register A (wenn es sich um einen 8-Bit-Wert handelt) oder im Registerpaar HL (wenn es sich um einen 16-Bit-Wert handelt). Der Inhalt der anderen Register ist dabei unbestimmt. Wenn Sie die ursprünglichen Inhalte nach dem Aufruf also noch benötigen, müssen Sie diese mit 'push' auf den Stapel retten.

Verwendung der BDOS-Routinen in Turbo-Pascal

In Turbo-Pascal können Sie auf die BDOS-Routinen zugreifen, ohne daß Sie Maschinensprache beherrschen müßten. Dazu gibt es die Standardprozedur *BDOS* und die Funktionen *BDOS* und *BDOSHL*. Diesen Routinen werden als Parameter Werte übergeben, die in die CPU-Register übertragen werden. Bei den Funktionsversionen erhalten Sie den Wert, der nach der Ausführung der BDOS-Routine im HL- oder A-Register steht. wie die Pascal-Prozeduren das im Detail erledigen, brauchen Sie nicht zu wissen.

Der Pascal-Funktion/Prozedur wird im ersten Parameter der Wert übergeben, der in den nachfolgenden Beispielen mit 'c:=' bezeichnet ist und im zweiten Parameter der Wert, der mit 'de:=' oder 'e:=' bezeichnet ist. Wenn die spezielle BDOS-Routine keinen definierten Wert im E-/DE-Register verlangt, können Sie den zweiten Parameter auch weglassen.

Wenn sie als Ergebnis den Wert im HL-Register erhalten und weiterverarbeiten möchten, müssen Sie die Funktion *BDOSHL* verwenden und wenn Sie den Inhalt des Registers A erhalten möchten, die Funktion *BDOS*. Die Aufrufe könnten also etwa so lauten:

```
BDOS(c,de);
hl:=BDOSHL(c,de);
a:=BDOS(c,de);
```

Obwohl die Beispiele in Maschinensprache beschrieben sind, können Sie sie so mit wenig Änderungen in Turbo-Pascal testen.

Wenn die BDOS-Routine die Adresse eines Speicherbereiches verlangt, können Sie mit *ADDR(<Variable>)* die Anfangsadresse einer genügend langen Variable angeben. Die BDOS-Routine verändert diese Variable dann direkt, ähnlich einem VAR-Parameter in einer selbstdefinierten Prozedur.

Ihr erstes CP/M-Programm

Jetzt ist es Zeit, die bisher gewonnenen Erkennnisse in ein allererstes CP/M-Programm umzusetzen. Zunächst sollten Sie aber die Dateien DDT. COM, ASM.COM, LOAD.COM und einen beliebigen Texteditor auf eine leere Arbeitsdiskette kopieren, da Sie diese im folgenden immer wieder brauchen.

Starten Sie nun den DDT und geben Sie entsprechend dem ersten Beispiel folgendes ein:

Eingabe	Bedeutung
`ddt`	Start des Debuggers
`a0100`	Eingabe ab Adresse 0100h
`mvi e,26`	Code für ein '&'(oder anderes Zeichen)
`mvi c,02`	Code für die Funktion Bildschirmausgabe
`call 0005`	Aufruf der BDOS-Funktion Nr. 2
`ret`	Programmende
`^C`	Abbruch des DDT
`save 1 zeichen.com`	Programm auf Diskette speichern

Bild 6-1: *Erzeugen eines kurzen COM-Programmes*

Geben Sie jetzt genauso, wie Sie sonst STAT oder FILECOPY eingeben, 'ZEICHEN' ein. Auf dem Bildschirm erscheint dann das Zeichen '&' - also genau das, was durch den 'call 0005h' zu erwarten war.

Neben ASM.COM und DDT.COM und so weiter befindet sich jetzt auch eine Datei ZEICHEN.COM auf der Diskette. ZEICHEN wurde also gewissermaßen zu einem neuen CP/M-Befehl, der völlig gleichberechtigt zu den originalen CP/M-Befehlen ist. Dieser neue CP/M-Befehl kann zwar nichts als das Zeichen '&' auszugeben, aber für den Anfang ist das doch gar nicht schlecht! Wenn Sie üben wollen, können Sie dieses Programm so ausbauen, daß es Ihren Namen ausgibt.

Jetzt folgt eine genaue Beschreibung aller BDOS-Funktionen, die es sonst noch gibt. Bei den dabei genannten Anwendungsbeispielen werden die Befehlsfolgen zum Aufrufen des Programms nicht mehr so ausführlich beschrieben. Das ist jetzt Ihre Aufgabe!

6.2 Einfache BDOS-Ein-/Ausgabefunktionen

Es gibt grundlegende Routinen, die einfach jeder Computer braucht, egal ob es sich um eine Großrechenanlage handelt oder um einen Homecomputer, mit dem man Daten nur auf dem Kassettenrekorder abspeichern kann: Es sind die Routinen zum Datenverkehr mit Bildschirm, Tastatur und eventuellen weiteren Peripheriegeräten. Ein CP/M-Computer besitzt diese Routinen natürlich auch - nur im Vergleich zu einem normalen Homecomputer sehr bequem aufzurufen. Diese Routinen sind in diesem Kapitel zusammengestellt.

In den Erklärungen zu den Funktionen werden nur die im allgemeinen üblichen Wirkungen benannt, zum Beispiel 'Bildschirmausgabe'. Sie sollten sich aber immer bewußt sein, daß Sie den Ein-/Ausgabekanälen auch andere reale Geräte zuweisen können.

Die Konsolenausgabe

Funktion: *Ausgabe eines Zeichens auf dem Konsolenkanal*

Die Steuerzeichen ^I, ^P und ^S werden von der Funktion ausgewertet: ^I (Tabulator) setzt den Cursor auf die nächste durch 8 teilbare Schreibposition. Wenn Sie während der Ausgabe eines Zeichens ^S drücken, wartet der Computer solange, bis Sie eine weitere Taste drücken. Die erste Eingabe von ^P während einer Zeichenausgabe schaltet das Mitprotokollieren auf dem Druckerkanal ein und die zweite Eingabe von ^P wieder aus.

Aufruf: c:= 2
e:= Code des auszugebenden Zeichens

Rückgabe: nichts

Beispiel:

```
mvi  e,'A'
mvi  c,02h
call 0005h
```

---> Auf dem Bildschirm erscheint ein 'A'.

Konsoleneingabe

Funktion: *Liest ein Zeichen vom Konsolenkanal*

Die Steuerzeichen ^S und ^P werden von der Funktion wie bei 'Konsolenausgabe' beschrieben ausgewertet.

Die Routine wartet nach dem Aufruf solange, bis eine gültige Taste gedrückt wird.

Alle druckbaren Zeichen werden automatisch als Echo auch auf dem Konsolenkanal ausgegeben. Deshalb eignet sich diese Funktion vor allem zur Eingabe von Texten. Ausgewertet werden hier auch die Zeichen Wagenrücklauf oder Enter (^M), Zeilenvorschub (^J), Tabulator (^I) und Backspace (^H).

Aufruf: c:= 1

Rückgabe: a:= Code des eingegebenen Zeichens

Beispiel:

```
mvi  c,01h
call 0005h
```

---> Auf dem Bildschirm ist ein eingegebenes Zeichen zu sehen und im A-Register steht dessen Code.

Konsolenstatus abfragen

Funktion: *Stellt fest, ob auf dem Konsolenkanal eine Eingabe bereit ist.*

Diese Funktion sollten Sie verwenden, wenn Ihr Programm die Kontrolle über den Computer auch dann zurückerhalten soll, wenn keine Taste gedrückt ist. Zurückgegeben wird zunächst nur die Information, ob überhaupt eine Taste gedrückt wurde. Wenn ja, wird aber zusätzlich der Code der gedrückten Taste zwischengespeichert. Die nächste Konsoleneingabe liefert dann dieses Zeichen, auch wenn die Taste inzwischen wieder losgelassen wurde.

Solange der zwischengespeicherte Tastendruck nicht 'abgeholt' wird, liefert jede neue Statusabfrage das Ergebnis 'ja' und alle späteren Tastatureingaben gehen verloren. Um den Computer nicht zu blockieren, muß also nach jeder erfolgreichen Konsolenstatusabfrage das eingetippte Zeichen auch eingelesen werden, auch wenn der Wert dann gar nicht mehr benötigt wird!

Aufruf: c:= 11

Rückgabe: a:= 0, wenn keine Taste gedrückt war
255, wenn doch eine Taste gedrückt war

Beispiel:

```
mvi  c,0b
call 0005h
```

---> Sie können auswerten, ob a=0 oder a=255.

Direkte Konsolen Ein- und Ausgabe

Funktion: *Weitergeben der Ein-/Ausgabe an oder vom Konsolenkanal ohne irgendeine Verarbeitung*

^P, ^S, ^C und ^I werden nicht ausgewertet und bei der Eingabe wartet die Routinen nicht auf einen Tastendruck. Auch ein Echo auf dem Bildschirm unterbleibt. Diese Funktion eignet sich also vor allem für Aufgaben, bei denen nur auf einen beliebigen Tastendruck gewartet werden muß, zum Beispiel bei 'Machen Sie dies oder jenes und drücken Sie dann eine beliebige Taste'.

Aufruf: c:= 6
e:= 255, wenn von der Konsole gelesen werden soll,
Code des auszugebenden Zeichens, wenn ein Zeichen an die Konsole ausgegeben werden soll.

Rückgabe: a:= Code des eingelesenen Zeichens, wenn eine Konsoleneingabe verlangt wurde, nichts, wenn ein Zeichen ausgegeben werden soll.

Beispiel:

```
mvi  e,ffh
mvi  c,06h
call 0005h
```

---> Das gelesene Zeichen kann ausgewertet werden.

Zeichenkette an die Konsole ausgeben

Funktion: *Ausgabe eines Speicherbereiches auf den Konsolenkanal*

Der Speicherbereich zwischen einer anzugebenden Adresse und dem nächstfolgenden Byte 24h (Dollarzeichen '$') wird auf dem Konsolenkanal ausgegeben. Erlaubt sind in diesem Speicherbereich alle Zeichen außer dem $-Zeichen - auch Steuerzeichen. Ein $-Zeichen muß aber als nicht auszugebende Abschlußmarkierung am Ende des Bereiches stehen. Die Eingabe der Steuerzeichen ^S und ^P während der Ausgabe wird berücksichtigt.

Aufruf: c:= 9
de:= Adresse der Zeichenkette

Rückgabe: nichts

Beispiel:

```
        lxi  d,string
        mvi  c,09h
        call 0005h
        ret
string: db   'Text ausgegeben',13,10,'$'
```

---> Auf dem Bildschirm ist der Text zu sehen.

Zeichenkette von der Konsole einlesen

Funktion: *String vom Konsolenkanal einlesen: Entspricht CP/M-Zeileneditor.*

Ermöglicht die Eingabe einer ganzen Zeile, die in einem anzugebenden Puffer abgelegt wird. Dabei werden alle Editiermöglichkeiten geboten, die auch zur Verfügung stehen, wenn Sie im CP/M-Betriebssystem etwa nach einem 'A>' einen Programmaufruf eintippen. Also die Funktionen:

^C Dieser Code als erstes Zeichen einer Zeile eingegeben führt zu einem Warmstart. So kann (fast) jedes CP/M-Programm während einer Eingabezeilenabfrage unterbrochen werden.
^E Führt auf dem Bildschirm einen Zeilenvorschub aus. Dieser wird jedoch im Betriebssystem nicht registriert. Im Eingabepuffer ist der Zeilenvorschub also nicht zu finden.
^H Eine Spalte zurück und letztes Zeichen löschen.
^J Entspricht ^M beziehungsweise Enter.
^M Wirkt wie die Enter-Taste
^R Gibt das bisher Eingetippte nochmal korrekt aus. Diese Funktion ist nur wichtig, wenn Sie einen Fernschreiber als Bildschirmgerät verwenden, bei dem das 'Zeichenlöschen' ja nicht dargestellt werden kann.
^U Löscht die bisher gemachten Eingaben
^X Wirkt genauso wie ^U.
DEL Löscht das letzte eingegebene Zeichen. In älteren CP/M-Versionen, die noch auf Fernschreiber-Ein-/Ausgabe spezialisiert sind, wird nicht versucht, das letzte Zeichen zu löschen. Das gelöschte Zeichen wird stattdessen zur Kennzeichnung verdoppelt. Weil dadurch das Bildschirmbild unübersichtlich wird, kann die derzeit gültige Eingabe durch ^R restauriert werden.

Außer diesen Standard-Editiermöglichkeiten gibt es auf den CPC-Computern noch die speziellen Schneider-Steuerzeichen. Zum Beispiel entspricht ^L einem CLS-Befehl und mit ^Z können Sie die Definition eines Windows einleiten. Die Bedeutung der Steuerzeichen ist im Anhang Ihres Computerhandbuches erklärt.

Alle eingegebenen Zeichen werden als Echo auf dem Bildschirm ausgegeben. Nach Abschluß der Zeile wird der Cursor auf die Position vor dem Editieren zurückgesetzt. Wenn der Puffer voll ist, wird die Eingabe abgebrochen, ohne daß die Zeile mit ENTER abgeschlossen werden braucht.

Aufruf: c:= 10
de:= Anfangsadresse des Puffers, der folgendermaßen aufgebaut ist:
1. Byte: muß die maximale Eingabelänge enthalten (zwischen 1 und 255)
2. Byte: reserviert für die Verwendung durch den Zeileneditor.
3. bis letztes Byte: reserviert für die eingegebenen Zeichen.

Rückgabe: steht im Puffer.
2. Byte: enthält die tatsächliche benötigte Eingabelänge,
3. bis letztes Byte: enthalten die eingegebenen Zeichen.

Beispiel:

```
        lxi  d,puffer  ;Pufferadresse
        mvi  c,0ah
        call 0005h     ;Eingabezeile holen
        lda  laenge    ;Eingabelänge ins a-Register
        lxi  h,text    ;Adresse der Eingabe ins hl-Register
        ret
puffer: db   80        ;maximale Eingabelänge
laenge: ds   1         ;1 Byte reservieren
text:   ds   80        ;Platz für Eingabe
```

---> Anforderung einer Benutzereingabe. Nach der Eingabe steht der Text sowohl im Puffer als auch auf dem Bildschirm.

Zeichen an den Lochstreifenstanzer ausgeben

Funktion: *Sendet ein Zeichen unverändert zum Lochstreifenstanzer-Kanal.*

Aufruf: c:= 4
e:= Code des auszugebenden Zeichens

Rückgabe: nichts

Beispiel:

```
mvi  e,'A'
mvi  c,04h
call 0005h
```

---> Das Zeichen 'A' wurde auf einen Lochstreifen gestanzt.

Zeichen vom Lochstreifenleser empfangen

Funktion: *Holt ein Zeichen ohne Auswertung vom Lochstreifenleser-Kanal.*

Die Routine wartet solange, bis ein Zeichen auf dem Lochstreifenleser anliegt. Wenn dieses Gerät überhaupt nicht existiert, sollte das Ergebnis bei jedem Aufruf ein ^Z oder 1Ah sein, das auch als Ende-Markierung einer Datei gilt. So wird unnötiges Lesen von sowieso ungültigen Daten vermieden.

Aufruf: c:= 3

Rückgabe: a:= Code des gelesenen Zeichens

Beispiel:

```
mvi  c,03h
call 0005h
```

---> Die Zahl in Register A auswerten.

Zeichen an den Drucker ausgeben

Funktion: *Gibt ein Zeichen unverändert an den Druckerkanal aus.*

Wiederholt den Übertragungsversuch solange, bis er einmal erfolgreich war. Bei nicht angeschlossenem Drucker wird der Rechner durch die Verwendung dieser Routine also blockiert.

Aufruf: c:= 5
e:= Code des auszugebenden Zeichens

Rückgabe: nichts

Beispiel:

```
mvi  c,05h
mvi  e,'A'
call 0005h
```

---> Auf dem Drucker erscheint beim nächsten Zeilenvorschub das Zeichen 'A'.

6.3 BDOS-Funktionen zum Systemeinstellen

Einige BDOS-Funktionen dienen dazu, das CP/M 2.2-Betriebssystem insgesamt einzustellen: Damit können Sie auch einige tiefgreifende Funktionen durchführen, ohne sich auf spezielle Speicherstellen im BIOS zu beziehen (die natürlich nur in den CPC-Computern gelten würden).

Input-/Output-Byte setzen

Funktion: *Zuordnung der realen Ein-/Ausgabegeräten zu den virtuellen Kanälen.*

Unter CP/M können Sie Ihre Ein- und Ausgaben nur über virtuelle Kanäle abwickeln. Die Zuordnung zu den realen Kanälen können Sie mit Hilfe dieser Funktion setzen und abfragen.

Die Informationsübergabe erfolgt dabei in einem einzigen Datenbyte. In diesem sind für jeden Ein-/Ausgabekanal 2 Bits reserviert. Jedes reale Ein /Ausgabegerät können Sie mit einer bestimmten Bitkombination anwählen. Mit 2 Bits sind genau 4 Kombinationen möglich - und genau das ist auch die Zahl der realen Geräte, die an einen Kanal zugewiesen werden kann. Im Detail bedeuten die Bits folgendes:

Bits 0/1: Konsolenkanal (CONSOLE, CON:)
- 00: (TTY:) Konsolen-Ein-/Ausgabe über die Fernschreiber-Tastatur und Drucker.
- 01: (CRT:) Konsolen-Ein-/Ausgabe über Bildschirm und Tastatur.
- 10: (BAT:) Eingabe über den Lochstreifenleser, Ausgabe auf den Druckerkanal.
- 11: (UC1:) vom Benutzer frei definierbares Konsolengerät

Bits 2/3: Lochstreifenleserkanal (READER, RDR:)
- 00: (TTY:) Reader-Eingabe über den Lochstreifenleser des Fernschreibers.
- 01: (PTR:) Reader-Eingabe über einen besonderen Lochstreifenleser.
- 10: (UR1:) 1. benutzerdefiniertes Reader-Gerät
- 11: (UR2:) 2. benutzerdefiniertes Reader-Gerät

Bits 4/5: Lochstreifenstanzerkanal (PUNCH, PUN:)
- 00: (TTY:) Puncher-Ausgabe auf den Lochstreifenstanzer im Fernschreiber.
- 01: (PTP:) Puncher-Ausgabe auf einen besonderen Lochstreifenstanzer
- 10: (UP1:) vom Anwender frei definierbares Puncher-Gerät
- 11: (UP2:) weiteres frei definierbares Puncher-Gerät

Bits 6/7: Druckerkanal (LIST, LST:)
- 00: (TTY:) Druckerausgabe auf den Drucker im Fernschreiber
- 01: (CRT:) Druckerausgabe auf den Bildschirm
- 10: (LPT:) Ausgabe auf einen besonderen Drucker
- 11: (UL1:) frei definierbares Ausgabegerät

Aufruf: e:= Input/Output-Byte
c:= 8

Rückgabe: nichts

Beispiel:

```
mvi  e,01$00$00$01B
mvi  c,08h
call 0005h
```

---> Ab diesem Befehl wird wie gewöhnlich die Reader Ein-/Ausgabe über den Fernschreiber und die Konsolenausgabe über ein Bildschirmgerät abgewickelt. Die Druckerausgabe erfolgt jedoch jetzt ebenfalls auf den Bildschirm. ^P verursacht deshalb eine Verdopplung jedes Zeichens, das auf den Bildschirm ausgegeben wird.

Input/Output-Byte abfragen

Funktion: *Feststellen, welche Geräte derzeit aktiv sind*

Als Ergebnis erhält man genau das Input/Output-Byte zurück, das mit einer BDOS-Funktion 8 abgeschickt wurde. Die Bits haben also dieselbe Bedeutung, wie sie auch bei 'Input/Output-Byte setzen' beschrieben wurde.

Aufruf: c:= 7

Rückgabe: a:= Input/Output-Byte

Beispiel:

```
mvi  c,07h
call 0005h
```

---> Das Register A enthält einen Code, welche Ein-/Ausgabegeräte gerade aktiviert sind.

Warmstart auslösen

Funktion: *Beenden der Bearbeitung eines Programms und Rückkehr ins Betriebssystem.*

Normalerweise wird vom Betriebssystem ein Warmstart durch einen Sprung zur Adresse 0000h ausgelöst. Das Verändern des Vektors an dieser Adresse ist jedoch erlaubt, so daß Sie unerwünschte Warmstarts abblocken können. Die BDOS-Routine führt aber in jedem Fall zu einem Warmstart.

Aufruf: c:=0

Rückgabe: nichts

Beispiel:

```
mvi  c,00h
call 0005h
```

---> Die Bearbeitung des Anwenderprogramms wird abgebrochen, die Diskettenlaufwerke zurückgesetzt und der CP/M-Prompt 'A>' oder 'B>' erscheint.

CP/M-Versionsnummer ermitteln

Funktion: *Stellt fest, welches Betriebssystem installiert ist.*

Nicht jede BDOS-Funktion gibt es in allen CP/M-Versionen. Unter CP/M 1.4 konnte man zum Beispiel weder die Usernummer verändern noch relative Dateien anlegen. Wenn Sie in Ihren CP/M-Programmen diese Funktionen verwenden wollen, sollten Sie diese Abfrage einfügen. Bei einer unpassenden Version können Sie dann eine entsprechende Warnung auf dem Bildschirm ausgeben lassen und das Programm abbrechen. Diese Meldung können aber auch Sie erhalten, wenn Sie versehentlich ein Programm verwenden, das für die CP/M-Version 3.1 geschrieben wurde. Das wird wohl jedem, der einen CPC 6128 besitzt, schon einmal passiert sein.

Aufruf: c:= 12

Rückgabe: hl:= Versionsnummer
h: 01h für MP/M und 00h für CP/M
l: 00h für Versionen einschließlich CP/M 1.4,
20h, 21h, 22h für CP/M 2.0, 2.1 und 2.2,
30h, 31h für Versionen 3.0 und 3.1

Beispiel:

```
mvi  c,0ch
call 0005h
```

---> Auf den CPC-Computern unter CP/M 2.2 enthält dann das HL-Register den Wert 0022h.

Benutzernummer abfragen und setzen

Funktion: *Abfragen und ändern des aktuellen Userbereichs*

Eine Diskette kann unter CP/M 2.2 in bis zu 16 logische Benutzerbereiche eingeteilt werden. Mit Hilfe dieser Funktion kann ein Anwenderprogramm selbständig die Benutzernummer ändern um zum Beispiel eine Datei von einem Userbereich zum anderen zu kopieren. Auch die Abfrage der aktuellen Benutzernummer ist möglich.

Aufruf: c:= 32
e:= 255 (zum Abfragen des aktuellen Userbereiches)
0..15 (Nummer zum Wählen des Userbereiches)

Rückgabe: nichts (wenn eine Usernummer gesetzt wurde)
a:= Nummer des aktuellen Userbereiches (Wenn der Userbereich abgefragt wurde).

Beispiel:

```
mvi  e,FFh
mvi  c,20h
call 0005h
```

---> Register A enthält die Nummer des derzeitigen Userbereiches.

Diskettenparameter ermitteln

Funktion: *Gibt die Adresse des DPB-Vektors der Bezugsdiskette zurück.*

Um zum Beispiel die noch freie Speicherkapazität der Diskette zu ermitteln, benötigt man detaillierte Informationen über die Verwaltung des speziellen Diskettenlaufwerkes. Deshalb gibt es diese Funktion, mit der man

die Anfangsadresse des Disk-Parameter-Blocks (nicht Disk-Parameter-Header) herausfinden kann.

Aufruf: c:= 31

Rückgabe: hl:= Adresse des DPB

Beispiel:

```
mvi  c,1Fh
call 0005h                    ;hl enthält DPB-Adresse
lxi  d,0005h!dad d            ;relative Adresse DSM
dad  d!mov e,m!inx h!mov d,m
inx  d
```

---> DE enthält die Anzahl der Blöcke, die die Diskette besitzt.

Belegte Blöcke der Diskette ermitteln

Funktion: *Feststellen der Adresse der Belegungstabelle des Bezugslaufwerkes*

Im BIOS gibt es für jedes Laufwerk eine Tabelle, in der für jeden Block der Diskette ein Bit reserviert ist. Ein gesetztes Bit bedeutet, daß der Block belegt ist. Die Anfangsadresse dieser Tabelle für das Bezugslaufwerk wird von der Funktion berechnet.

Aufruf: c:= 27

Rückgabe: hl:= Adresse der Belegungstabelle

Beispiel:

```
mvi  c,1Bh
call 0005h
mov  a,m
rrc             ;Bit 0 ins Carry-Flag
```

---> Wenn der 8. Block der Diskette belegt ist, ist das Carry-Flag gesetzt.

6.4 BDOS-Funktionen zur Laufwerkssteuerung

Diese Gruppe der BDOS-Funktionen brauchen Sie, um Ihre Laufwerke grundlegend zu steuern ohne eine bestimmte Datei anzusprechen. Zum Beispiel wenn Sie ein Diskettenlaufwerk inaktivieren möchten um die Diskette wechseln zu können.

Einige Funktionen verwenden zur Parameterübergabe den 'Laufwerksvektor'. Dieser Laufwerksvektor ist eine 16-Bit-Zahl, wobei jedem Laufwerk ein Bit zugeordnet ist. Je nach Anwendung enthält ein bestimmtes Bit eine bestimmte Information über das entsprechende Laufwerk. Ist nach Anwendung der BDOS-Funktion 29 etwa das Bit 1 des Laufwerksvektors gesetzt, dann ist die Diskette im Laufwerk B schreibgeschützt.

Das niedrigste Bit des Laufwerksvektors entspricht dem Laufwerk A und das höchstwertige Bit dem 16. Laufwerk P. Wenn Sie nur 2 Laufwerke besitzen, enthalten aber nur die beiden niederwertigsten Bits eine sinnvolle Information. Da wohl nur die wenigsten mehr als 8 Diskettenlaufwerke angeschlossen haben, genügt es in der Regel nur die unteren 8 Bits des Laufwerksvektors auszuwerten, also nur ein Byte.

Diskettensystem zurücksetzen

Funktion: *Alle Laufwerke inaktivieren*

Eine Diskette in einem aktiven Laufwerk darf unter CP/M 2.2 nicht gewechselt werden. Durch diese Funktion werden alle Laufwerke inaktiviert, so daß Sie alle Disketten ohne weiteres wechseln dürfen.

Dabei wird auch das Laufwerk A inaktiviert, obwohl es sonst, wie nach einem Warmstart, immer aktiv ist. Beim Inaktivieren wird auch der Schreibschutz für alle Laufwerke gelöscht.

Aufruf: c:= 13

Rückgabe: nichts

Beispiel:

```
mvi  c,0Dh
call 0005h
```

---> Alle Laufwerke sind inaktiviert.

Einzelne Laufwerke zurücksetzen

Funktion: *Hier kann auch nur ein bestimmtes Laufwerk inaktiviert werden.*

In bestimmten Situationen ist das Zurücksetzen des gesamten Diskettensystems zum Wechseln einer Diskette nicht zulässig. Das ist zum Beispiel dann der Fall, wenn ein anderes Laufwerk noch nicht geschlossene Dateien enthält (wird später noch besprochen). Deshalb können mit dieser Funktion einzelne Laufwerke selektiv inaktiviert werden.

Die Informationsübergabe an die Funktion erfolgt hier durch den Laufwerksvektor: Ein gesetztes Bit bedeutet, daß das entsprechende Laufwerk zurückgesetzt wird. Auch hier wird der Schreibschutz des zurückgesetzten Laufwerkes gelöscht.

Aufruf: c:= 37
de:= Laufwerksvektor

Rückgabe: nichts

Beispiel:

```
lxi  d,0000$0000$0000$0010b
mvi  c,25h
call 0005h
```

---> Die Diskette im Laufwerk B kann jetzt gewechselt werden. Der Zustand der anderen Laufwerke bleibt unverändert.

Bezugslaufwerk festlegen

Funktion: *Wählen, welches Laufwerk bei Dateioperationen ohne Laufwerksangabe verwendet wird.*

Bei der Ausführung einer Dateioperation kann eine Laufwerksnummer angegeben werden, auf die die Operation wirkt. Die Dateioperation kann aber auch ohne Laufwerksangabe angewendet werden. In diesem Fall wird automatisch mit dem Bezugslaufwerk gearbeitet.

Nach dem Einschalten ist A das Bezugslaufwerk. Mit dieser Funktion kann das aber geändert werden. Diese Funktion entspricht also im wesentlichen dem Befehl B:. Hier wird aber nach einem Warmstart automatisch wieder das alte Bezugslaufwerk gewählt.

Die anzugebenden Laufwerksnummern sind dabei 0 für das Laufwerk A, 1 für das Laufwerk B und so weiter bis 15 für das Laufwerk P.

Aufruf: c:= 14
e:= Nummer des ausgewählten Laufwerks

Rückgabe: nichts

Beispiel:

```
mvi  c,0Eh
mvi  e,01h
call 0005h
```

---> Alle zukünftigen Dateioperationen ohne Laufwerksangabe beziehen sich auf das Laufwerk B.

Bezugslaufwerk abfragen

Funktion: *Gibt die Nummer des aktuellen Laufwerks zurück*

Am Anfang eines Programms ist nicht klar, welches Laufwerk gerade das Bezugslaufwerk ist. Mit dieser Funktion kann diese Nummer abgefragt werden. Dabei hat das Laufwerk A wieder die Nummer 0.

Aufruf: c:= 25

Rückgabe: a:= Nummer des Bezugslaufwerks

Beispiel:

```
mvi  c,19h
call 0005h
```

---> Register A enthält die Nummer des Bezugslaufwerkes.

Laufwerk schreibschützen

Funktion: *Setzt das Bezugslaufwerk logisch in den 'Nurlese-Status'*

Der mit dieser Funktion eingestellte Schreibschutz ist nur ein 'logischer' Schreibschutz: Nur der Versuch, mit BDOS-Funktionen das Laufwerk zu beschreiben, wird unterbunden. Mit Hilfe von 'Tricks' (oder Programmfehlern) kann man die so geschützte Diskette trotzdem noch beschreiben. Der einzige wirkliche Schutz ist also das Überkleben (bei 5 1/4"-Disketten) beziehungsweise Öffnen (bei 3"-Disketten) der Schreibschutzkerbe.

Aufruf: c:=28

Rückgabe: nichts

Beispiel:

```
mvi  c,1Ch
call 0005h
```

---> Das aktuelle Laufwerk wird schreibgeschützt.

Schreibgeschützte Laufwerke ermitteln

Funktion: *Gibt an, welche Laufwerke schreibgeschützt sind.*

Das versehentliche Beschreiben einer schreibgeschützten Diskette würde einen Programmabbruch verursachen. Deshalb gibt es diese Funktion, mit der Sie schon vorbeugend einen Schreibschutz entdecken und vielleicht eine Warnung ausgeben können.

Als Ergebnis der Funktion erhalten Sie einen 'Laufwerksvektor': Die Bits der schreibgeschützen Laufwerke sind gesetzt. Entdeckt wird jedoch nicht der hardwaremäßige Schreibschutz durch die Schreibschutzkerbe einer Diskette, sondern nur ein vom BDOS gesetzter softwaremäßiger Schreibschutz.

Aufruf: c:= 29

Rückgabe: hl:= Laufwerksvektor

Beispiel:

```
mvi  c,1Dh
call 0005h
mov  a,l
rra             ;Bit 0 vom L-Register ins Carry-Flag
```

---> Wenn das Laufwerk A schreibgeschützt ist, ist das Carry-Flag gesetzt.

Aktive Laufwerke ermitteln

Funktion: *Gibt an, welche Laufwerke aktiviert sind.*

Nur die Belegungstabelle eines aktiven Laufwerkes ist gültig. Wenn Sie nun zum Beispiel die freie Kapazität der eingelegten Disketten bestimmen wollen, dürfen Sie deshalb nur aktive Disketten berücksichtigen. Sie benötigen also eine Routine, mit der Sie feststellen können, welche Laufwerke aktiv sind. Auch hier erfolgt die Informationsübergabe im Laufwerksvektor: Nur die Bits aktiver Laufwerke sind gesetzt.

Aufruf: c:= 24

Rückgabe: hl:= Laufwerksvektor

Beispiel:

```
mvi  c,18h
call 0005h
mov  a,l
rar!rar
```

---> Wenn das Laufwerk B aktiv ist, ist das Carry-Flag gesetzt.

6.5 BDOS-Funktionen zur Dateiverwaltung

6.5.1 Allgemeines zur Dateiverwaltung

Übertragungseinheiten

Die Übertragung von Daten in oder aus einer Datei erfolgt immer in Portionen zu 128 Bytes (= 1 Record), egal wieviele Bytes davon wirklich benötigt werden: Auch für eine Datei, die nur ein einziges Byte enthält, schreibt das BDOS einen ganzen Record auf die Diskette. An diese Regel müssen Sie sich als Anwender ebenfalls halten: Es gibt eine BDOS-Funktion, mit der Sie einen beliebigen 128 Bytes langen Speicherbereich als Datenpuffer deklarieren können. Alle nachfolgenden Schreib- und Leseoperationen schreiben diesen Datenpuffer auf die Diskette beziehungsweise lesen Daten von der Diskette in diesen Datenpuffer.

Im BIOS findet hardwareabhängig eine weitere Vergröberung statt: Mehrere Records werden solange im RAM zwischengespeichert, bis sich ein vollständiger Sektor (in den CPC-Computern 512 Bytes) angesammelt hat. Erst dieser wird dann im Stück auf der Diskette abgespeichert. Davon gibt es nur wenige Ausnahmen: Ein Eintrag ins Inhaltsverzeichnis wird zum Beispiel aus Sicherheitsgründen sofort zurückgespeichert. Aber selbst da wird zuerst der vollständige 512-Bytes Sektor gelesen, die betroffenen 32 Bytes geändert und der ganze Sektor wieder zurückgespeichert.

Im Inhaltsverzeichnis der Diskette werden immer ganze Gruppen von Records für eine Datei auf einmal reserviert. So eine Gruppe nennt man einen Block. Ein Block kann je nach System zwischen 8 und 128 Records enthalten. Gültige Daten enthalten aber nur die Records, die tatsächlich beschrieben wurden. Alle anderen werden von den BDOS-Routinen nicht verändert und enthalten zufällige Werte: zum Beispiel die Daten einer gelöschten Datei, die früher an dieser Stelle lag. Immerhin enthält das Inhaltsverzeichnis aber einen Wert, der angibt, wieviele der Records eines Blocks sinnvolle Daten enthalten. Die nicht belegten Records in diesem Block sind für andere Dateien gesperrt.

Das Inhaltsverzeichnis der Diskette ist wirklich der einzige Ort, an dem Informationen über die Lage der Datei zu finden sind. Beim Schreiben und Lesen jedes einzelnen Records einer Datei wird also das Inhaltsverzeichnis benötigt.

Zugriffszeiten

Es wäre sehr ungünstig, wenn für jeden zu lesenden Record ein echter Inhaltsverzeichniszugriff nötig wäre: Das eigentliche Lesen eines Records (=1024 Bits) dauert bei einer Übertragungsrate von 250 Kilobit pro Sekunde etwa 1/250 Sekunde beziehungsweise 4 Millisekunden. Das Aufsuchen der Inhaltsverzeichnisspur und das Zurückfahren in die neue Position dauert bei einem 3"-Laufwerk im Mittel 2*20*12 msec = ca. 2/3 Sekunde. Diese Zeit setzt sich zusammen aus der Zeit von 12 msec für einen einzigen Spurwechsel und dem zweimaligen Abfahren von im Mittel der halben Diskette. Die Vorbereitungen zum Lesens eines Records würden also 250 mal so lange dauern wie der eigentliche Lesevorgang. Das Laden einer 16 KBytes langen Datei würde so etwa 2 Minuten dauern!

Solche Werte sind wirklich unzumutbar. In der Tat dauert das Laden einer 16-KByte-Datei nur etwa 3 Sekunden (diese Zeit setzt sich aber immer noch zusammen aus 1 sec Hochlaufzeit des Motors, 1 sec für das Aufsuchen der richtigen Spur und nur 1/2 sec für den eigentlichen Lesevorgang). Im CP/M Betriebssystem wird dazu zu folgendem 'Trick' gegriffen:

Öffnen einer Datei

Es wurde vereinbart, daß jede Datei vor der Verwendung erst 'geöffnet' werden muß. Bei diesem Öffnen wird der vollständige Inhaltsverzeichniseintrag der Datei ins RAM kopiert. Alle folgenden Dateizugriffe beziehen sich dann auf die Kopie im RAM - das physische anfahren der Directoryspur erübrigt sich so. Lediglich beim Wechseln des logischen Extents muß die Kopie im RAM aktualisiert werden. Aber das kommt ja nur bei jedem 128. Record vor.

Die Art des Öffnens einer Datei ist unabhängig davon, ob die Datei nur gelesen oder auch beschrieben werden soll. Im BDOS gibt es deshalb nur eine einzige Dateiöffnungsfunktion. Während der Bearbeitung einer Datei können Sie auch beliebig zwischen Schreiben und Lesen wechseln. Auch wenn Sie ursprünglich nur von der Datei lesen wollten, dürfen Sie irgendwo zwischendrin mal einen Record auf die Diskette schreiben.

Schließen einer Datei

Beim Beschreiben einer neuen Datei wird ähnlich vorgegangen: Sofort auf der Diskette abgespeichert wird nur der Inhalt der Datei. Die Nummern der zusätzlich benötigten Blöcke werden zunächst nur in der RAM-Kopie des Verzeichniseintrages vermerkt. Außerdem wird das dem neuen Block entsprechende Bit in der Blockreservierungstabelle gesetzt.

Wenn die Datei vollständig ist, müssen Sie das dem Betriebssystem durch eine besondere BDOS-Funktion mitteilen. Erst dann wird die neue Directoryinformation auf die Diskette zurückgeschrieben. Diesen Vorgang nennt man auch 'Schließen einer Datei'.

Diese Vorgehensweise hat jedoch einen folgenschweren Nachteil: Wenn Sie die Diskette aus dem Laufwerk entfernen ohne daß alle Dateien geschlossen sind oder den Rechner ausschalten, enthält sie zwar schon die neuen Dateiinformationen, im Inhaltsverzeichnis sind aber die Blöcke, die diese Informationen enthalten, noch nicht verzeichnet. Bei einer späteren Verwendung der Diskette erhält das Betriebssystem also keinen Hinweis darauf, daß diese Blöcke eine sinnvolle Information enthalten und werden einfach von neuen Dateien überschrieben.

Disketten, die noch zum Schreiben geöffnete Dateien enthalten, dürfen sie deshalb in einem CP/M-System niemals wechseln! Selbst das deaktivieren einer Diskette nützt dabei nichts. Nur Dateien, auf die Sie nichts geschrieben haben, brauchen Sie nicht zu schließen: Denn der Inhaltsverzeichniseintrag braucht ja nicht geändert zu werden.

Bearbeiten mehrer Dateien gleichzeitig

Für jede Datei, die Sie bearbeiten wollen, müssen Sie dem BDOS einen - je nach Funktion - 33 oder 36 Bytes langen Zwischenspeicher zur Verfügung stellen, in der alle systeminternen Informationen zur Bearbeitung der Datei angelegt werden - einschließlich dem Namen der Datei. Intern benötigt das BDOS keinen weiteren Speicherplatz zur Verwaltung der Datei. Diesen Speicherbereich nennt man einen 'Dateibeschreiber'.

Wenn Sie eine Operation auf eine bestimmte Datei durchführen wollen, dann müssen Sie dem BDOS die Anfangsadresse eines Dateibeschreibers mitteilen: Die Operation wirkt dann automatisch auf die Datei, deren Namen in diesem Beschreiber genannt ist, wobei sich das BDOS bei der Ausführung nur auf die im Beschreiber enthaltenen Informationen stützt.

Vergessen Sie niemals das Eröffnen der Datei! Solange die Datei noch nicht eröffnet ist, enthält der Dateibeschreiber noch zufällige Daten. Eine Schreib- oder Leseoperation kann diese jedoch nicht von sinnvollen unterscheiden. Der Record der Datei würde so an eine zufällige Stelle auf der Diskette geschrieben. In der Regel hat das katastrophale Folgen.

Zur Bearbeitung einer neuen Datei brauchen Sie nur einen weiteren 33 oder 36 Bytes langen Speicherbereich zu reservieren, der in der gesamten freien TPA liegen kann und ihn mit bestimmten Startwerten zu versorgen. Je nachdem ob Sie bei einer Dateioperation dann die Adresse des einen

oder des anderen Dateibeschreibers angeben, wird dann ein Record der einen oder der anderen Datei gelesen oder beschrieben. Selbstverständlich dürfen sie nicht vergessen, auch diese neue Datei zu eröffnen.

Sie können also soviele Dateien gleichzeitig geöffnet haben, wie es der freie Speicherplatz zuläßt. Sogar das gleichzeitige Beschreiben mehrerer Dateien ist möglich: Neu belegte Blöcke werden ja sofort in der Blockbelegungstabelle reserviert, so daß die anderern Dateien diesen Block selbst dann nicht beschreiben können, wenn auf sie unmittelbar danach zugegriffen wird. Verwirrung mit doppelt belegten Blöcken wird so vermieden.

Aufbau des Dateibeschreibers

Ein Dateibeschreibers ist sehr ähnlich zu einem Inhaltsverzeichniseintrag aufgebaut. Im Bild 6-1 sehen Sie deshalb diesen im Vergleich zu einem Verzeichniseintrag. Besonders achten sollten Sie auf die Erweiterungen des Dateibeschreibers gegenüber dem Verzeichniseintrag.

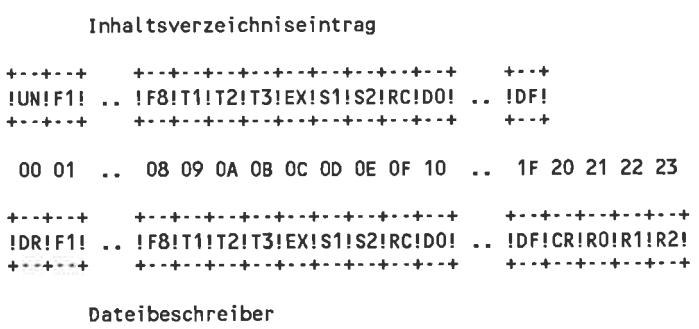


Bild 6-1: *Aufbau eines Dateibeschreibers*

Die einzelnen Abkürzungen haben dabei die folgende Bedeutungen:

DR — Der 'Drive Code':
An der entsprechenden Stelle eines Inhaltsverzeichniseintrag steht die Nummer des Userbereiches, in der sich die Datei befindet beziehungsweise ob sie gelöscht ist. Alle Dateioperationen beziehen sich aber auf die gerade aktuelle Usernummer. Selbstverständlich ist auch kein Dateibeschreiber für eine gelöschte Datei vorhanden.
Das erste Byte des Dateibeschreibers wurde deshalb durch die Laufwerksnummer ersetzt, auf der sich die Datei befindet. Hier steht der Wert 0, wenn die Datei auf einem beliebigen eingestelltem Bezugslaufwerk zu finden ist und 1 bis 16 für die speziell gewählten Laufwerke A bis P.

`F1...F8` Der 'Filename':
Diese Bytes entsprechen genau dem Verzeichniseintrag.
Die unteren 7 Bits dieser 8 Bytes enthalten den Dateinamen in Ascii-Codierung. Die 8. Bits werden vom Ascii-Code nicht verwendet und stehen für andere Daten zur Verfügung. Diese 8. Bits sind so belegt:

`F1'..F4'` Diese 4 Bits sind nicht belegt. Sie können sie deshalb selbst für beliebige Markierungen verwenden.

`F5'..F8'` Diese 4 Bits sind für eine eventuelle spätere Verwendung von Digital Research reserviert.
Nicht benötigte Namensbytes werden durch Leerzeichen (Code 20h) aufgefüllt.

`T1...T3` Der 'Filetype':
Diese Bytes entsprechen ebenfalls genau einem Inhaltsverzeichniseintrag.
Die unteren 7 Bits dieser 3 Bytes enthalten den Dateityp in Ascii-Großschrift. Wie beim Dateinamen haben auch hier die 8. Bits andere Bedeutungen:

`T1'` Ein gesetztes Bit an dieser Stelle bedeutet, daß die Datei schreibgeschützt ist.

`T2'` Ein gesetztes Bit an dieser Stelle kennzeichnet einen SYS-File, der nicht im Inhaltsverzeichnis erscheint, aber normal verwendet werden kann.

`T3'` Dieses 8. Bit wird unter CP/M 2.2 nicht verwendet, ist aber für eine spätere Verwendung durch Digital Research reserviert.

`EX` Das 'Extent Byte':
Eine Datei kann bis zu 512 logische Extents besitzen (512 = 8 Megabytes/128 Bytes). Die Extensionnummer ist also eine 9-Bit-Zahl. Die niederwertigen 5 Bits dieser Nummer sind in diesem Byte enthalten.
Im Verzeichniseintrag ist hier die Nummer des letzten gültigen logischen Extents des physischen Extents registriert. Im Gegensatz dazu steht an dieser Stelle im Dateibeschreiber die Nummer des gerade behandelten logischen Extents.

`S1, S2` Diese Bytes sind im Dateibeschreiber für die interne Verwendung durch das BDOS reserviert. Im Gegensatz zum Verzeichniseintrag enthält das Byte S2 nur in besonderen Fällen die höherwertigen Bits der aktuellen Extentnummer. Außer mit den Spezialbefehlen zum wahlfreien Zugriff können deshalb nur die ersten 512 KByte einer Datei adressiert werden.

RC — Das 'Record Count Byte':
Im Inhaltsverzeichnis enthält dieses Byte die Anzahl der Records im letzten logischen Extent dieses Verzeichniseintrages. Im Dateibeschreiber enthält es die Anzahl der Records im gerade behandelten logischen Extent. Es kann Werte zwischen 0 und 128 annehmen.

D0...DF — Diese 16 Bytes enthalten die Nummern der Blöcke, die für die Datei in diesem Verzeichniseintrag reserviert sind. Kleine Disketten, die höchstens 256 Blöcke besitzen, haben 8-Bit-Blocknummern. Bei diesen Disketten können hier 16 Blöcke reserviert werden.
Größere Disketten haben 16 Bit lange Blocknummern. Deshalb ist dieser Bereich für diese in der Form von 8 je 16 Bit langen Worten organisiert. In diesem Fall können nur 8 Blöcke in einem Verzeichniseintrag reserviert werden.

CR — Dieses Byte im Dateibeschreiber enthält die Nummer des Records im logischen Extent, von dem als nächstes gelesen oder in den als nächstes geschrieben wird. Es kann Werte zwischen 0 und 128 annehmen. Zusammen mit der Extentnummer im Byte EX kann so jeder Record in einem 512-KByte-File adressiert werden.

R0...R2 — Diese Bytes werden nur zum wahlfreien Dateizugriff benötigt. Sie enthalten dann die relative Recordnummer vom Beginn der Datei an gerechnet. Die Bytes R0 und R1 stellen einen 16-Bit-Wert dar. Mit diesem können 65536 * 128 Bytes, also 8 Megabyte adressiert werden. Das ist gleichzeitig auch die maximale Dateigröße unter CP/M 2.2. Das Byte R2 dient nur dazu, einen eventuellen Überlauf aufzunehmen.

Der Dateibeschreiber heißt auch 'File Control Block'. Oft wird er deshalb auch mit 'FCB' abgekürzt.

Die Bytes R0, R1 und R2 werden nur für ganz spezielle Dateioperationen benötigt. Wenn Sie sicher sind, daß Sie keine solchen Operationen verwenden, brauchen Sie nur 33 Bytes für den Dateibeschreiber reservieren. Dieser Teil ist der 'Standard-FCB'. Bei Operationen, die die zusätzlichen 3 Bytes benötigen, heißt der Dateibeschreiber 'Extended File Control Block' und wird zu EFCB abgekürzt.

Initialisierung eines Dateibeschreibers

Schon beim Eröffnen einer Datei müssen einige Informationen über die Datei angegeben werden, wozu die Eröffnungsfunktion schon auf den

Dateibeschreiber Bezug nimmt. Bei der Programmierung müssen Sie deshalb die folgenden Bytes des Dateibeschreibers selbst vorbelegen:

DR (Byte 0) Laufwerksnummer: In dem hier angegebenen Laufwerk wird die zu öffnende Datei gesucht.

F1..F8, T1..T3 (Bytes 1..11) Das ist der Name der Datei. Nicht benötigte Stellen werden mit Leerzeichen aufgefüllt. Die Werte der 8. Bits für die Dateimerkmale müssen hier Null sein.

X, S2, CR (Bytes 12, 14,32) Diese Bytes bestimmen die Nummer des Records, der beim ersten Dateizugriff gelesen wird. In der Regel sollten Sie diese Bytes also beim Dateieröffnen auf 0 setzen.

Alle anderen Bytes des Dateibeschreibers werden vom BDOS automatisch mit den richtigen Werten versorgt. Außer in speziellen Anwendungen brauchen Sie sich darum nicht zu kümmern.

Das Arbeiten mit sequentiellen Dateien

Einige BDOS-Funktionen wurden speziell zum sequentiellem Lesen und Schreiben ins Betriebssystem eingefügt, das heißt für Dateien, die in der Regel 'von vorne nach hinten' in einem Durchgang bearbeitet werden. Bei diesen Funktionen wird nach der Bearbeitung des aktuellen Records der Datei automatisch der Record mit der nächsten Nummer zum aktuellem erklärt. Sollte dabei ein logischer Extent voll werden, wird automatisch ein neuer Extent eröffnet.

Aufeinanderfolgende Lesebefehle lesen also ohne weitere Vorkehrungen auch aufeinanderfolgende Records aus der Datei. Beim Versuch, einen Record zu lesen, in den vorher noch nichts geschrieben wurde, erhalten Sie eine Fehlermeldung. In der Regel tritt das auf, wenn die aktuelle Recordnummer 'hinter' dem Ende der Datei liegt. Bei Dateien, die mit den Befehlen zum relativen Bearbeiten einer Datei erzeugt wurden, kann die Fehlermeldung aber auch bei einem 'Loch' in der Datei auftreten.

Aufeinanderfolgende Schreibbefehle wirken ebenfalls auf aufeinanderfolgende Records. Ist der Record, in den gerade geschrieben wurde schon in der Datei enthalten, so wird dessen Inhalt einfach durch die neuen Daten ersetzt. Anders ist es jedoch, wenn die aktuelle Recordnummer den Record bezeichnet, der hinter dem letzten Record der Datei folgen würde: Der neue Record wird dann durch den sequentiellen Schreibbefehl einfach an die Datei angehängt. Auf diese Weise ist auch die Verlängerung einer schon bestehenden Datei kein Problem.

Direkter Zugriff auf sequentielle Dateien

Die Bytes EX und CR des Dateibeschreibers geben die Nummer des Records an, der als nächstes gelesen wird. Die Startwerte werden jedoch vom BDOS nicht automatisch eingetragen, sondern müssen von Ihnen eingegeben werden. Das S2-Byte enthält beim sequentiellem Zugriff keine Informationen über den Extent. Trotzdem muß es aber manuell auf einen definierten Startwert gesetzt werden. Wenn Sie diese 3 Bytes beim Eröffnen der Datei auf den Wert 0 setzen, werden die Records beginnend mit dem Record Nr. 0 sequentiell gelesen oder beschrieben.

Mit anderen Startwerten ist es aber möglich, die sequentielle Bearbeitung bei einem anderern Record beginnen zu lassen. Das ist zum Beispiel dann sinnvoll, wenn Sie eine Datei nur am Ende fortsetzen wollen und die alten Daten weder ändern noch lesen wollen.

Eine Recordnummer ist beim sequentiellem Zugriff eine 12-Bit-Zahl. Diese Zahl müssen Sie zum adressieren eines Records in die Bytes EX und CR übertragen. Die 12 Bits der Recordnummer müssen also irgendwie auf 16 Bits verteilt werden. Die Zuordnung der einzelnen Bits der Recordnummer zu den Bits in den Dateibeschreiber-Bytes ist im Bild 6-2 dargestellt.

Recordnummer	Darstellung im Dateibeschreiber
Bits 0 - 6	Byte CR, Bits 0 - 6
Bits 7 - 11	Byte EX, Bits 0 - 4

Bild 6-2: *Codierung der Recordnummer im Dateibeschreiber*

Die hier nicht verwendeten Bits in den Bytes CR, EX haben den Wert 0. Zum Adressieren des Records mit der Nummer 200 in einer Datei müssen Sie also in das Byte EX den Wert 1 und in das Byte CR den Wert 72 eintragen. Ein einfacher Umwandlungsalgorithmus, der eine Recordnummer im HL-Register richtig im Dateibeschreiber ablegt, könnte so aussehen:

```
hl:= Recordnummer
mov  a,l
ani  7Fh
sta  fcbcr      ;Recordnummer im aktuellen Extent
mov  a,l
add  a
mov  a,h
ral
ani  1Fh
sta  fcbex      ;laufende Extentnummer
```

Nach dem sequentiellen Lesen oder Schreiben eines Records werden diese Bytes automatisch auf den nächstfolgenden Record gestellt. Danach können

Sie diese Bytes aber selbst auf einen anderen Record einstellen. Damit ist auch mit sequentiellen Dateioperationen ein direkter Zugriff auf wahlfreie Records möglich.

Auf einfache Weise ist ein direkter Zugriff nur möglich, wenn beim Eintragen der Recordnummer der logische Extent nicht geändert wird: Die Blocknummern in den Bytes D0 bis DF des Dateibeschreibers beziehen sich ja auch noch auf den alten Extent. Das Lesen von Records ist mit einer unpassenden Extentnummer zwar möglich, Sie erhalten dann aber dieselben Daten, die Sie auch schon beim Lesen des letzten Extents erhalten haben.

Wenn sich also beim direkten Zugriff die Extentnnummer ändert, müssen Sie unbedingt die Blocknummern in den Bytes D0 bis DF anpassen. Am einfachsten können Sie das erreichen, indem Sie die Datei mit den geänderten Recordnummern neu erföffnen. Denn dadurch wird der Dateibeschreiber ja vollständig initialisiert.

Dieses Neueröffnen müssen Sie unbedingt bei jedem neuen logischen Extent durchführen, also alle 16 KByte. Auch wenn Ihre Disketten in größeren physischen Extents organisiert sind, enthält der Dateibeschreiber Daten, die sich speziell auf den logischen Extent beziehen. Ein Beispiel dafür ist das Byte RC, das nur die Information enthält, ob noch ein weiterer logischer Extent folgt. Außerdem erreichen Sie nur so eine vollständige Unabhängigkeit von den verschiedenen Diskettenformaten.

Die Blocknummern des letzten Extents werden beim Neueröffnen der Datei zerstört. Wenn diese Blocknummern noch nicht auf die Diskette zurückgespeichert sind, müssen Sie das vor dem Neueröffnen tun. Am einfachsten geschieht das durch das Schließen der Datei. Die Blocknummern sind nur dann noch nicht auf die Diskette zurückgespeichert, wenn Sie seit dem letzten Extentwechsel die bearbeitete Datei um einige Records erweitert haben. In der Regel können Sie sich diesen Arbeitsschritt also sparen.

Beim Ergänzen einer Datei ist jedoch mit dieser Methode das Erzeugen eines neuen physischen Extents nicht möglich. Wenn Ihre Disketten in 16 KByte langen physischen Extents organisiert sind, können Sie in den 150. Record erst schreiben, wenn der der 2. Extent bereits existiert. Das ist nur der Fall, wenn mindestens der Record mit der Nummer 127 beschrieben wurde. Nur innnerhalb eines Extents können Sie Records überspringen und so 'Löcher' erzeugen.

Bild 6-3 zeigt nochmals alle Vorgänge, die nötig sind, um einen beliebigen Record der Datei anzuwählen.

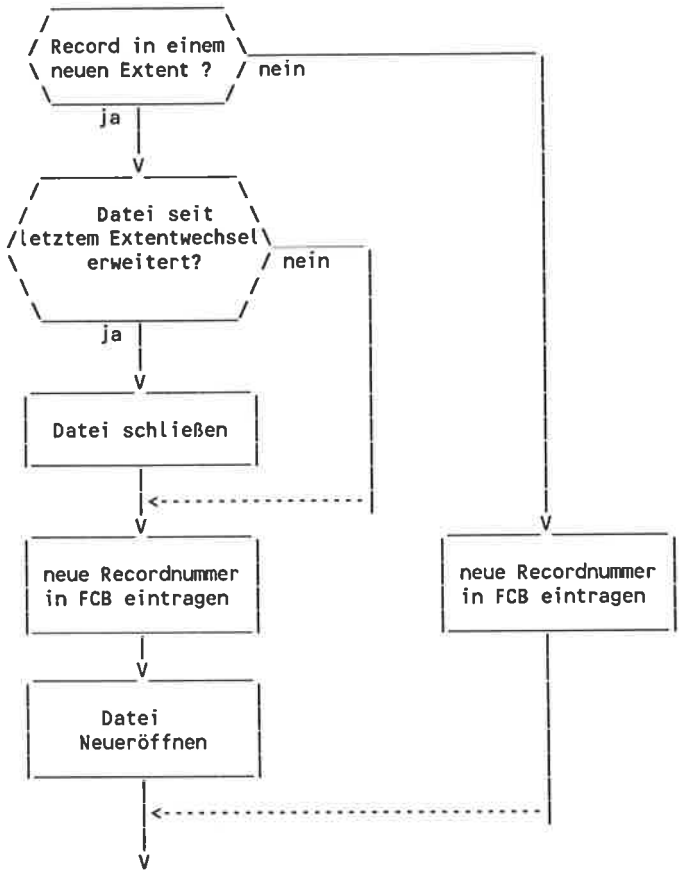


Bild 6-3: *Wahlfreier Zugriff auf eine Datei*

Wenn Sie sehr oft Records aus verschiedenen Extents benötigen, geht sehr viel Zeit durch die häufigen Inhaltsverzeichnizugriffe verloren. Durch einen Trick können Sie diese Zugriffe aber vermeiden: Eröffnen sie doch die fragliche Datei mehrmals mit verschiedenen Dateibeschreibern so, daß jeder Dateibeschreiber auf einen anderen logischen Extent derselben Daei zeigt. Wenn dann der Extent gewechselt werden soll, dann brauchen Sie nur die Dateioperation mit dem entsprechenden schon vorhandenen Dateibeschreiber durchzuführen. Das Schließen und Neueröffnen der Datei ist dann unnötig. In diesem Fall ist allerdings der Verwaltungsaufwand für die Datei wesentlich größer (und damit fehleranfälliger) als bei einer klassischen Bearbeitung. Wegen des Geschwindigkeitsvorteils ist sie trotzdem zur Arbeit mit Overlay-Programmen oder großen Datenbanken mit häufigen Zugriffen auf bestimmte Records gut geeignet.

Arbeiten mit relativen Dateien

Mit den BDOS-Funktionen zur sequentiellen Dateiverarbeitung ist der Direktzugriff auf ganz bestimmte Records der Datei recht schwierig. Unter CP/M 2.2 gibt es deshalb einige Funktionen, die speziell zum relativen Bearbeiten von Dateien geeignet sind.

Diese Routinen erledigen die Umwandlung von der Recordnummer in die interne Darstellung, das Öffnen und das Schließen weiterer Extents automatisch. Sie als Anwender brauchen sich darum überhaupt nicht mehr zu kümmern. Der Direktzugriff auf bestimmte Records der Datei ist damit genauso einfach wie der sequentielle Zugriff. Dafür wird aber die Recordnummer nach dem Lesen oder Schreiben nicht auf den nächsten Record weitergestellt. Wiederholtes Lesen/Schreiben mit einer Direktzugriffsfunktion ohne weitere Maßnahmen wirkt also immer auf denselben Record.

Nur die Routinen zur relativen Dateibearbeitung brauchen einen erweiterten Dateibeschreiber mit den Bytes R0, R1 und R2: In die Bytes R0 und R1 können Sie direkt die Nummer des Records ablegen, der als nächstes gelesen oder beschrieben werden soll. auf diese Weise können wirklich 65536 verschiedene Records adressiert werden. Relative Dateien können so bis zu 8 Megabytes lang werden.

Unter CP/M 2.2 gibt es aber nur eine einzige Dateiart. Dateien, die mit Hilfe der Befehle zur relativen Dateiverwaltung erzeugt wurden, können also ohne weiteres sequentiell gelesen werden. Der umgekehrte Vorgang ist ebenso möglich. Selbst während eines einzigen Bearbeitungsdurchgangs einer Datei kann beliebig zwischen sequentiellem und unmittelbarem/relativem Lesen und Schreiben umgeschaltet werden:

Bei jeder relativen Diskettenoperation muß die Recordnummer in die interne Darstellung umgewandelt und der entsprechende Verzeichniseintrag aufgesucht werden. Als Nebeneffekt wird dabei der Dateibeschreiber auch für sequentielles Lesen vorbereitet. Nach einer relativen Lese-/Schreiboperation kann also unmittelbar eine sequentielle Schreib-/Leseoperation folgen. Weil nach der relativen Dateioperation keine Weiterschaltung des Recordzählers erfolgt, wird bei der ersten sequentiellen Dateioperation allerdings der letzte relativ gelesene Record ein zweites Mal eingelesen. Erst ab dem 2. Mal funktioniert das sequentielle Lesen richtig.

Der umgekehrte Vorgang, die Umrechnung der internen Recordnummer in die Darstellung für die relativen Dateioperationen, erfolgt durch eine spezielle BDOS-Funktion. Die so berechnete Recordadresse zeigt auf den ersten Record nach dem letzten sequentiell gelesenem Record.

Dateien, die mit den Dateibefehlen zur relativen Dateiverwaltung bearbeitet werden, müssen unbedingt mit den Werten 0 in den Bytes EX, S2 und CR geöffnet werden. Sonst funktioniert die Synchronisierung zwischen absoluter und relativer Aufzeichnungsnummer nicht mehr richtig und der Recordzähler 'verhaspelt' sich.

Im Gegensatz zu den sequentiellen Dateioperationen kann zum Schreiben mit relativen Dateioperationen sofort nach dem Erzeugen jede beliebige Recordnummer im Bereich von 0 bis 65535 angegeben werden. Eventuell noch nicht existierende Extents werden hier automatisch erzeugt. Zwischen erstem und letztem Extent der Datei können aber durchaus Verzeichniseinträge fehlen. Im Datenbereich der Diskette belegen ebenfalls nur schon beschriebene Blöcke Speicherplatz. Eine relative Datei kann so besonders große Löcher enthalten. Erweitert man die relative Datei solange, bis alle Löcher geschlossen sind, wird daraus eine ganz normale sequentielle Datei.

Wenn in einen Block nur ein einziger Record geschrieben wurde, tritt ein besonderer Effekt auf: Obwohl nur ein Record auf die Diskette geschrieben wurde, gelten in diesem Fall auch alle anderen Records des Blocks als von der Datei belegt. 7 (bei 1 KByte-Blöcke) weitere eigentlich noch nicht beschriebene Records können so ohne Fehlermeldung gelesen werden, obwohl sie nur unsinnige Daten enthalten. Die Entscheidung, ob der lesbare Record zur Datei gehört oder nicht, müssen Sie als Programmierer treffen. Eine kleine Hilfe gibt eine BDOS-Funktion, die beim Schreiben eines Records gleich den ganzen Block mit Nullen löscht. Ein echtes Unterscheidungskriterium über die Gültigkeit der Daten ist das aber nicht.

Beim Arbeiten mit den BDOS-Routinen zum direkten Zugriff können Sie den Trick mit mehreren Dateibeschreibern für eine Datei prinzipiell ebenfalls verwenden. Genauer beschrieben wurde dieser schon bei 'Arbeiten mit sequentellen Dateien'. Beim Arbeiten mit relativen Dateien ist die Verwaltung aber noch schwieriger, da es wesentlich häufiger vorkommt, daß ein Dateibeschreiber geändert wird. Wenn sie den Geschwindigkeitsvorteil ausnützen wollen, müssen Sie hier besonders vorsichtig sein.

6.5.2 Grundlegende BDOS-Dateifunktionen

Alle hier genannten Dateifunktionen beziehen sich nur auf die Dateien im akuellen Benutzerbereich. Selbst wenn Sie etwa die Entsprechung des Befehls ERA *.* anwenden, werden nur die Dateien eines Users gelöscht. Das gilt auch dann, wenn im Text nur ganz allgemein 'sucht im Inhaltsverzeichnis' steht.

Datei eröffnen

Funktion: *Eine schon vorhandene Datei auf der Diskette zur Bearbeitung vorbereiten.*

Die Funktion sucht im Inhaltsverzeichnis der im 1. Byte des Dateibeschreibers angegebenen Diskette nach dem in den folgenden Bytes genannten Dateinamen. Die Bytes EX und S2 tragen ebenfalls zum Finden des richtigen Eintrages bei. Das Byte S1 im Dateibeschreiber wird automatisch auf Null gesetzt.

Die höherwertigen Bits der einzelnen Namensbytes müssen immer Null sein. Gefunden werden aber trotzdem Dateien mit beliebigen Attributen.

In jedem Namensbytes und in den Bytes EX und S2 dürfen auch Fragezeichen als 'Joker' stehen: Die Inhaltsverzeichnis-Bytes an diesen Positionen verzeichnis gefundene Extent eröffnet, der auf den Namen paßt.

Wenn ein passender Eintrag gefunden wird, wird er in den Dateibeschreiber kopiert. Eventuelle Fragezeichen werden also überschrieben.

Aufruf: c:= 15
de:= FCB-Anfangsadresse

Rückgabe: a:= 0,1,2 oder 3 wenn das Öffnen erfolgreich war,
255 wenn kein passsender Eintrag gefunden wurde.

Beispiel:

```
        lxi  d,fcb          ;Anfangsadresse Dateibeschreiber
        mvi  c,0Fh
        call 0005h
        cpi  04h
        jnc  error          ;Öffnen war nicht erfolgreich
        jmp  ok
fcb:    db   0              ;Bezugslaufwerk
        db   'NAME    TXT'  ;11 Bytes Name
        db   0,0,0          ;EX, S1 und S2
        ds   17             ;beliebiger Inhalt
        db   0              ;Zeiger auf 1. Record stellen
        ds   3              ;nur notwendig, wenn relative
                            ;Dateioperationen verwendet werden.
```

---> Die Datei NAME.TXT ist zur Bearbeitung vorbereitet.

Erzeugen einer Datei

Funktion: *Erzeugen eines neuen Dateinamen im Inhaltsverzeichnis der Diskette.*

Vor dem ersten Öffnen einer neuen Datei muß sie erst einmal erzeugt werden. Diese BDOS-Funktion erstellt eine leere Datei mit dem gewünschten Dateinamen, so daß Sie sie beliebig erweitern können. Beim Erzeugen müssen Sie die Laufwerksnummer und den Dateinamen angeben.

Die Funktion übernimmt keinerlei Kontrolle, ob der angegebene Dateiname auch korrekt ist. Wenn Sie einen unzulässigen Namen (zum Beispiel mit Kleinbuchstaben) verwenden, haben Sie es sehr schwer, auf die Datei erneut zuzugreifen. Besonders gefährlich sind hier Fragezeichen: Beim Versuch eine Datei mit einem solchen Namen zu löschen, werden auch gleich alle anderen ähnlichnamigen Dateien, die anstelle des Fragezeichens ein beliebiges Zeichen haben, gelöscht.

Der neue Dateiname wird einfach in den ersten freien Platz des Inhaltsverzeichnisses eingetragen. Dabei wird nicht überprüft, ob dadurch derselbe Dateiname zweimal im Verzeichnis auftaucht. Bei einem Leseversuch können Sie dann nur die Datei erreichen, die als erstes im Inhaltsverzeichnis auftaucht. Ob das die ältere oder die neue Datei ist, hängt nur vom Zufall ab. Wenn Sie sicher gehen wollen, daß kein Name doppelt auftaucht, müssen Sie vor jedem Erzeugen einer Datei die Löschfunktion für Dateien dieses Namens aufrufen.

Als Nebeneffekt wird beim Erzeugen einer Datei diese gleich richtig eröffnet. Wenn Sie die neue Datei gleich anschließend beschreiben wollen, können Sie das also ohne weitere Vorbereitungen tun.

Aufruf: c:= 22
de:= FCB - Anfangsadresse

Rückgabe: a:= 0,1,2 oder 3 wenn die Operation erfolgreich war,
255 wenn kein Platz im Directory frei war.

Beispiel:

```
        lxi  d,fcb
        mvi  c,16h
        call 0005h
        cpi  04h
        jnc  error
        jmp  ok
fcb:    db   2              ;Datei auf Laufwerk B erzeugen
        db   'NAME    TXT'  ;11 Bytes Namen
        db   0,0,0
```

```
ds   18          ;Rest beliebig
ds   3           ;Zusatzbytes, wenn relative Bear-
                 ;beitung geplant ist.
```

---> Im Inhaltsverzeichnis der Diskette ist jetzt eine Datei mit dem Namen NAME.TXT zu finden.

Datei schließen

Funktion: *Dem Betriebssystem das Ende der Bearbeitung einer Datei mitteilen.*

Durch diese Funktion werden eventuelle Änderungen des Dateibeschreibers, die während der Bearbeitung der Datei entstanden sind, auf die Diskette zurückgespeichert. Nur so gehen die neuen Daten in der Datei beim Diskettenwechsel nicht verloren. Zum Schließen einer Datei müssen Sie unbedingt einen Dateibeschreiber verwenden, der durch das Öffnen der Datei entstanden ist und seitdem manuell nicht mehr verändert wurde. Ein in 'Handarbeit' angefertigter unvollständiger Dateibeschreiber kann zu Schwierigkeiten führen. Eine Ausnahme ist nur für spezielle 'Tricks' erlaubt.

Wenn Sie die bearbeitete Datei nicht erweitert haben, blieb der Dateibeschreiber unverändert. Das Zurückspeichern des Beschreibers durch Schließen erübrigt sich dann. Sie können den Dateibeschreibers also einfach anderweitig verwenden, wenn Sie sicher sind, daß Sie ihn nicht mehr benötigen. Beachten Sie aber, daß eine relative Datei oft erweitert wird, ohne daß Ihnen das auffällt. Das sicherste ist also, jede Datei, auf die irgendwie geschrieben wurde nach der Bearbeitung zu löschen.

Aufruf: c:= 16
de:= Anfangsadresse eines schon initialisierten FCB

Rückgabe: a:= 0,1,2 oder 3 wenn das Schließen erfolgreich war,
255 wenn die zu schließende Datei im Inhaltsverzeichnis nicht gefunden wurde.

Beispiel:

```
lxi  d,fcb
mvi  c,10h
call 0005h
cpi  04h
jnc  error
jmp  ok
```

---> Die Bearbeitung der durch den Dateibeschreiber benannten Datei wurde ordnungsgemäß beendet.

Löschen einer Datei

Funktion: *Löscht die im Dateibeschreiber angegebenen Dateien von der Diskette.*

Im Dateinamen können wie beim Öffnen der Datei Fragezeichen vorkommen. Gelöscht wird dann nicht nur eine Datei, sondern alle Dateien, deren Name zur der angegebenen Maske paßt. Betroffen sind nicht nur die ersten Extents jeder Datei, sondern alle Extents.

Die höherwertigen Bits der einzelnen Namensbytes müssen zurückgesetzt sein. Gefunden werden aber trotzdem Dateien mit beliebigen Attributen.

In einem Arbeitsgang können nur Dateien auf einem einzigen Laufwerk gelöscht werden. Im ersten Byte des Dateibeschreibers darf also kein Fragezeichen stehen.

Die Dateien und Verzeichniseinträge werden nicht wirklich gelöscht: In das Inhaltsverzeichnis wird nur ein Erkennungsbyte eingetragen, das die Datei als ungültig erklärt. Mit gewissen Tricks ist es so manchmal möglich, die Datei zu restaurieren.

Aufruf: c:= 19

de:= Anfangsadresse FCB

Rückgabe: a:= 0,1,2 oder 3 wenn das Löschen erfolgreich war,
255 wenn keine zu löschende Datei gefunden wurde.

Beispiel:

```
        lxi  d,fcb
        mvi  c,13h
        call 0005h
        cpi  04h
        jnc  error          ;keine zu löschende Datei gefunden
        jmp  ok
fcb:    db   1              ;Löschen auf Laufwerk A
        db   'D???????TXT'  ;Dateinamen
        ds   21             ;beliebiger Inhalt
```

---> Jetzt sind alle Dateien vom Typ TXT gelöscht, deren Namen mit dem Buchstaben 'D' beginnen.

Umbenennen einer Datei

Funktion: *Ersetzt alle Vorkommnisse eines bestimmten Namens im Inhaltsverzeichnis durch einen anderen*

Auch hier werden die angegebenen Namen nicht auf Korrektheit überprüft. Hüten Sie sich also vor der Verwendung von Fragezeichen und anderen nicht erlaubten Namen. Von der Namensänderung betroffen sind alle Extents einer Datei und nicht etwa nur der erste. Ein Umbenennen in mehreren Durchgängen ist also unnötig.

Nicht überprüft wird auch, ob der neue Dateiname schon vorher im Inhaltsverzeichnis vorhanden ist. Wie beim Erzeugen einer neuen Datei können Sie auch hier einen doppelten Dateinamen im Verzeichnis erzeugen, wenn Sie nicht aufpassen.

Die höherwertigen Bits der einzelnen Namensbytes müssen zurückgesetzt sein. Gefunden werden trotzdem Dateien mit beliebigen Attributen. Umbenennen lassen sich aber nur Dateien, bei denen das Read-Only-Attribut nicht gesetzt ist.

Der neue Dateiname befindet sich im Dateibeschreiber in den Bytes 17 bis 27. Sie müssen also zum Umbenennen unbedingt einen speziellen Dateibeschreiber anlegen und dürfen nicht etwa einen noch benötigten Dateibeschreiber ändern. Mit Ausnahme der Laufwerksnummer im 1. Byte des Dateibeschreibers können alle anderen Bytes beliebige Werte haben.

Aufruf: c:= 23
de:= Anfangsadresse eines speziellen Dateibeschreibers

Rückgabe: a:= 0,1,2 oder 3 wenn das Umbenennen erfolgreich war,
255 wenn kein passender Name gefunden wurde.

Beispiel:

```
        lxi  d,fcb          ;spezieller Dateibeschreiber
        mvi  c,17h
        call 0005h
        cpi  04h
        jnc  error          ;Dateiname nicht gefunden
        jmp  ok
fcb:    db   1              ;Umbenennen im Laufwerk A
        db   'ALTNAME TXT'  ;Alter Name
        db   0,0,0,0,0      ;nicht benötigte Bytes
        db   'NEUNAME TXT'  ;neuer Name
        db   0,0,0,0,0      ;nicht benötigt
```

---> Die frühere Datei ALTNAME.TXT hat jetzt den Namen NEUNAME.TXT.

Datenpuffer festlegen

Funktion: *Adresse des Puffers bestimmen, den alle folgenden Dateioperationen verwenden.*

Datenübertragungen zwischen Disketten und Speicher erfolgen in Portionen zu 128 Bytes. Durch diese Funktion können Sie den Speicherbereich, in dem diese 128 Bytes abgelegt oder aus dem sie entnommen werden frei bestimmen. Diese Änderung bleibt solange gültig, bis Sie eine neue Adresse angeben oder ein Warmstart ausgeführt wird. Diesen Pufferspeicher nennt man oft auch 'Direct Memory Access'-Bereich oder 'DMA-Bereich'.

Auch bei sequentiellen Lese- oder Schreiboperationen wird die DMA-Adresse nicht verändert. Wenn Sie die Resultate aus aufeinanderfolgenden Leseoperationen hintereinander im Speicher ablegen wollen, müssen Sie die DMA-Adresse nach jedem Lesebefehl manuell um den Wert 128 erhöhen. Dasselbe gilt für das Abspeichern eines längeren Speicherbereiches.

Nach einem Warmstart ist der Bereich zwischen 0080h und 00FFh automatisch als DMA-Bereich eingestellt. Wenn Sie mit diesem Wert zufrieden sind, brauchen Sie beim Programmstart keine neue Adresse zu setzen. Dieser Wert wird auch nach einem Rücksetzen des Diskettensystems automatisch eingestellt.

Aufruf: c:= 26
de:= neue DMA-Adresse

Rückgabe: nichts

Beispiel:

```
lxi  d,dma
mvi  c,1Ah
call 0005h
```

---> Ab jetzt werden die Ergebnisse aus Leseoperationen im Speicherbereich ab der Adresse dma abgelegt. Dasselbe gilt für die Schreibbefehle.

6.5.3 BDOS-Funktionen zur Arbeit mit dem Directory

Unter CP/M liegt das Inhaltsverzeichnis der Diskette nicht irgendwo verborgen: Mit Hilfe einiger BDOS-Funktionen können Sie ganz gezielt eine Übersicht über die Inhalte Ihrer Diskette gewinnen, ohne extra auf einzelne Dateien zugreifen müssen.

Dateimerkmale setzen

Funktion: *Verändern der höchstwertigen Bits in den Bytes von Dateinamen.*

Unter CP/M 2.2 können einzelne Dateien schreibgeschützt oder unsichtbar gemacht werden. Diese Attribute sind in den höchstwertigen Bits der Bytes im Dateinamen codiert: Ein gesetztes Bit 7 im 1. Typbyte des Dateinamens bedeutet, daß die Datei schreibgeschützt ist, ein gesetztes Bit 7 im 2. Typbyte bedeutet, daß die Datei im Inhaltsverzeichnis nicht erscheint. Den höchstwertigen Bits der ersten 4 Buchstaben im Dateinamen dürfen Sie eigene Bedeutungen geben.

Tragen Sie zunächst einen auf der Diskette vorhandenen Dateinamen in einen Dateibeschreiber ein und setzen Sie die Dateiattribute nach Ihren Wünschen. Durch den Aufruf dieser Funktion werden diese Attribute auf die Diskette kopiert. Hüten Sie sich bei der Namensangabe vor Fragezeichen und sonstigen unerlaubten Zeichen.

Beim Eröffnen einer Datei werden die schon gesetzten Dateiattribute zusammen mit den anderen Dateiinformationen in den Dateibeschreiber kopiert. Sie können also ganz einfach testen, welche Attribute eine Datei hat. Insbesondere gilt das auch für die frei definierbaren Attribute in den ersten 4 Namensbytes.

Aufruf: c:= 30
de:= Anfangsadresse FCB

Rückgabe: a:= 0,1,2 oder 3 wenn die Operation erfolgreich war,
255 wenn der zu ändernde Dateiname nicht gefunden wurde.

Beispiel:

```
        lxi  d,fcb
        mvi  c,1Eh
        call 0005h
        cpi  04h
        jnc  error
        jmp  ok
fcb:    db   0              ;Bezugslaufwerk
        db   'NAME    '     ;8 Namensbytes
        db   'T' or 80h     ;Read-Only setzen
        db   'X' or 80h     ;Unsichtbar machen
        db   'T'            ;Letztes Namensbyte
        ds   21             ;beliebiger Inhalt
```

---> Die Datei NAME.TXT ist nach dem Aufruf schreibgeschützt und mit dem DIR-Befehl nicht auflistbar.

Dateigröße ermitteln

Funktion: *Virtuelle Dateigröße feststellen.*

Eine Datei kann auch 'Löcher' besitzen. Diese Funktion betrachtet die Löcher so, als ob sie Daten enthalten würden. Zurückgegeben wird also nicht der wirkliche Speicherbedarf auf der Diskette, sondern die Nummer des ersten Records hinter dem Dateiende. Da der erste Record die Nummer 0 besitzt, ist das gleichzeitig die (virtuelle) Zahl der in der Datei enthaltenen Records. Die Funktion liefert nur dann ein sinnvolles Ergebnis, wenn der im Dateibeschreiber angegebene Name auch wirklich auf der Diskette existiert.

Die Dateigröße wird in den Bytes R0 und R1 des erweiterten Dateibeschreibers zurückgegeben. Enthält das Byte R2 den Wert 1 hat die Datei schon die Maximalgröße von 8 MegaByte und kann nicht mehr erweitert werden. Ein umittelbar folgender Direktzugriffs-Schreibbefehl schreibt also in den Record direkt hinter dem Dateiende.

So können Sie auch ganz einfach mit sequentiellen Schreibbefehlen eine bestehende Datei fortsetzen, ohne erst die Datei von Anfang an einzulesen:

1. Nach dem Feststellen der Dateigröße schreiben Sie ohne weitere Vorbereitung einen Record mit dem Befehl für unmittelbaren Zugriff in die Datei. Dieser Record wird unmittelbar an das Dateiende angehängt. Die Daten in diesem Record sind dann zwar sinnlos, aber das macht in diesem Fall nichts.
2. Durch den Schreibbefehl ist der Dateibeschreiber auch für das sequentielle Schreiben vorbereitet, wobei der aktuelle Record der ist, der gerade mit unsinnigen Daten beschrieben wurde.
3. Jetzt können Sie mit dem sequentiellem Schreiben beginnen. Der erste Schreibzugriff überschreibt den unsinnigen Record und dann geht's ganz normal weiter.

Zum Berechnen der Dateigröße muß der FCB so vorbereitet sein, als ob Sie die Datei eröffnen wollen. Wenn die Datei schon eröffnet ist, dürfen Sie selbstverständlich den schon vorhandenen FCB verwenden. Beachten Sie aber, daß durch die Funktion die Adresse des aktuellen Records verlorengeht. Sie können also die Datei danach nicht mehr sequentiell an der alten Stelle weiter beschreiben.

Aufruf: c:= 35
de:= Anfangsadresse eines erweiterten FCB

Rückgabe: nichts

Beispiel:

```
lxi  d,fcb     ;Der FCB muß den Dateinamen enthalten
mvi  c,23h
call 0005h
lhld fcb+21h
```

---> Register HL enthält die Nummer des ersten nicht mehr zur Datei gehörenden Records.

Ersten Inhaltsverzeichniseintrag suchen

Funktion: *Lädt den Inhalt des ersten zum angegebenen Namen passenden Inhaltsverzeichniseintrages in den Speicher*

Zum Dateinamen im Dateibeschreiber wird der erste passende Inhaltsverzeichniseintrag gesucht. Zunächst wird der Record, in dem der gefundene Eintrag liegt, in den DMA-Bereich kopiert. Da jeder Inhaltsverzeichniseintrag 32 Bytes lang ist, finden in einem 128 Byte-Record 4 Stück Platz. Ein Eintrag kann also an 4 Positionen liegen, die mit den Nummern 0 bis 3 bezeichnet sind. Diese Nummern gibt die Routine als Ergebnis zurück.

Die Namensbytes dürfen auch Fragezeichen enthalten, so daß aus mehreren möglichen Dateinamen der erste vorkommende ausgewählt wird. Die Werte in den beiden Bytes EX und S2 geben die Nummer des Extents an, der gesucht wird - Sie können also beim Suchen die ersten Extents einer Datei gezielt umgehen. Sie können aber auch Fragezeichen in die Extentbytes setzen: Dann wird ein beliebiger Extent der gesuchten Datei in den Speicher geholt.

In der Regel gibt das erste Byte im FCB das Laufwerk an, auf dem die Datei gesucht wird. Nur in dieser BDOS-Funktion dürfen Sie den Laufwerkscode durch ein Fragezeichen ersetzen. Gesucht kann dann nicht mehr auf einem anzugebenden Laufwerk werden, sondern nur noch auf dem Bezugslaufwerk. Dafür werden aber Dateien aus allen Userbereichen gefunden (nicht nur aus dem aktuellem, wie gewöhnlich) und sogar gelöschte Inhaltsverzeichniseinträge. Wenn das erste Byte des Dateibeschreibers kein Fragezeichen enthält, wird das Byte S2 automatisch auf den Wert 0 gesetzt.

Aufruf: de:= Anfangsadresse FCB
c:= 17

Rückgabe: a:= 0,1,2 oder 3 wenn ein passender Directoryeintrag gefunden wurde. Dann gibt die Zahl die Position des Inhaltsverzeichniseintrages im Datenpuffer an.
255 wenn kein Eintrag gefunden werden konnte.

Beispiel:

```
        lxi  d,fcb
        mvi  c,11h
        call 0005h
        cpi  FFh
        jz   error
        add  a!add a!add a!add a!add a
        mov  e,a!mvi d,00h
        lxi  h,dma
        dad  d
        jmp  ok
fcb:    db   0                  ;Laufwerk 0
        db   '??????????????'   ;14 Fragezeichen
        ds   18                 ;beliebige Werte
```

---> HL enthält die Position, an der der erste Eintrag des Inhaltsverzeichnisses im Speicher liegt. Gefunden wurden in diesem Fall nur gültige Einträge aus dem aktuellen Userbereich, unabhängig davon, in welchem Userbereich er liegt und ob er gelöscht ist oder nicht.

Weiteren Inhaltsverzeichniseintrag suchen

Funktion: *Sucht einen weiteren Directoryeintrag, der zur Namensangabe bei der letzten Dateioperation paßt.*

Diese BDOS-Funktion sucht den nächsten Verzeichniseintrag, der zu dem bei der letzten Dateioperation verwendeten Verzeichniseintrag paßt. Sie stützt sich also nicht auf einen eventuell hier angegebenen weiteren Dateibeschreiber, sondern auf die im BDOS gespeicherten 'Überreste' der letzten Dateioperation. In der Regel ist die Anwendung dieser BDOS-Funktion nur unmittelbar nach der Funktion 'Suchen nach dem ersten Verzeichniseintrag' sinnvoll.

Das Ergebnis erhalten Sie in der gleichen Form wie bei der Funktion 'Suchen nach dem ersten Eintrag': Der Record mit dem gefundenen Verzeichniseintrag wird in den DMA-Bereich kopiert, wobei das Register A die Lage des Eintrages in diesem Bereich enthält.

Auf diese Weise können Sie sich eine vollständige Kopie des Inhaltsverzeichnisses im RAM anfertigen: Starten Sie die Suche mit 15 Fragezeichen im Dateibeschreiber und dem Suchen des ersten Eintrages. Danach lesen Sie solange weitere Verzeichniseinträge, bis ein Fehler auftritt. Nach jeweils 4 Einträgen beginnt ein neuer Record im Inhaltsverzeichnis. Sie sollten also nach jedem 4. Lesevorgang den DMA-Bereich um 128 Bytes verschieben, um diesen ebenfalls in einen neuen Bereich zu kopieren.

Aufruf: c:= 18

Rückgabe: a:= 0,1,2 oder 3 wenn ein passender Eintrag gefunden wurde. Das A-Register enthält dann die Lage des Verzeichniseintrages im Datenpuffer.
255 wenn kein weiterer Verzeichniseintrag gefunden wurde.

Beispiel:

```
mvi  c,12h
call 0005h
```

---> Analog zur Funktion 11h enthält der Datenpuffer einen Ausschnitt aus dem Inhaltsverzeichnis und das Register A die Lage des gefundenen Eintrages im Ausschnitt.

6.5.4 Beispiele zum Umgang mit dem Directory

Das folgende Programm zeigt, wie die Anwendung der Directory-BDOS-Funktionen in der Praxis aussehen könnte. Dabei können Sie auch gleich Ihr Wissen über die Verwaltung der Dateien vervollständigen:

Die EX-DIR-Programme geben nicht nur die Namen der im Inhaltsverzeichnis aufgeführten Dateien aus, sondern alle Informationen, die überhaupt über Dateien enthalten sind (Siehe Bild 6-4).

```
A>ex-dir-1
Vollständige Directoryinformation

00 MOVCPM  COM 00 00 00 4C 01 02 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 COPY    COM 00 00 00 0D 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 FORMAT  COM 00 00 00 07 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 FILECOPYCOM 00 00 00 16 06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                                  .
                                  .
                                  .
```

Bild 6-4: *Wirkung von EX-DIR-1.COM*

Die 11 Namensbytes werden wie beim gewöhnlichen DIR-Befehl in Klarschrift ausgegeben. Zusätzlich sehen Sie aber die anderen 21 Bytes des Verzeichniseintrages in hexadezimaler Schreibweise. Sie können also genau feststellen, welche Blöcke von der Datei belegt sind oder wie lang die Datei ist.

Gezeigt werden sowohl gelöschte als auch Einträge aller Benutzerbereiche. Diese unterscheiden sich in der ersten Zahl einer Zeile: Gelöschte Einträge haben an dieser Stelle den Wert E5h, ansonsten steht da die Benutzernummer, zu der die Datei gehört. Verschiedene Extents zur selben Datei

unterscheiden sich in den Bytes EX und S2. Wenn in den 3 Typbytes des Dateinamens Graphikzeichen zu sehen sind, ist ein Dateiattribut gesetzt. Das Graphikzeichen entstcht durch die Addition des Wertes 80h zum normalen Ascii-Code.

Damit Sie sehen, daß Sie die BDOS-Funktionen nicht nur in Maschinensprache nützlich anwenden können, ist das Programm EX-DIR auch in der Turbo-Pascal-Version angegeben. So können Sie ohne Maschinenspracheкenntnisse das Inhaltsverzeichnis einer Diskette in ein Programm integrieren.

Programm EXDIR1: die Assemblerversion

Wenn dieses Programm als COM-Datei auf Ihrer Diskette vorliegt, können Sie es einfach durch die Eingabe von 'EXDIR1' starten. Danach erscheinen auf dem Bildschirm alle Directoryeinträge der Diskette - egal ob sie belegt sind oder nicht.

Beachten Sie, daß im Assemblerlisting der Dateibeschreiber nur beim Suchen des ersten Verzeichniseintrages benötigt wird. Ab dem 2. Durchlauf der Schleife wird die FCB-Adresse vor Suchoperationen nicht mehr ins DE-Register übertragen. Beachten Sie auch, daß dazu passend am Ende der Schleife auch der Code für 'Suchen eines Eintrages' geändert wird.

Nach dem Lesen eines Eintrages enthält der Datenpuffer (der sich ohne besondere Maßnahmen an der Adresse 0080h befindet) den gesuchten und 3 weitere Verzeichniseinträge. Das Register A gibt an, welcher der Einträge der gesuchte ist, wobei der erste die Nummer 0 besitzt. Durch Multiplikation mit 32 ergibt sich die Anfangsadresse des gesuchten Eintrages. 'ADD A' verdoppelt den Wert des Registers A, so daß die fünfmalige Wiederholung das Gewünschte bewirkt.

Vor dem eigentlichen Programmstart wird der Maschinenstapel in einen ausreichend großen lokalen Bereich übertragen. Das sollten Sie in jedem Ihrer eigenen Programme ebenfalls machen, wenn Sie sich vor unliebsamen Überraschungen schützen wollen. Ein Trick ist jedoch, daß die alte Stackpointeradresse noch gespeichert wird: Vor dem Programmende kann das Programm so die Lage des alten Stapels rekonstruieren und unmittelbar durch einen RET-Befehl in den CCP zurückspringen. Dadurch wird ein lästiger Warmstart eingespart. Diese Vorgehensweise ist jedoch nur dann erlaubt, wenn in jedem Fall sichergestellt ist, daß der CCP beim Programmablauf nicht überschrieben wird.

```
                ;Ausgeben der gesamten Inhaltsverzeichnisinformation
                ;       der Diskette
0005 =          bdos    equ     0005h
0080 =          dma     equ     0080h
0011 =          firdir  equ     11h     ;ersten Directoryeintrag suchen
0012 =          nexdir  equ     12h     ;nächsten Directoryeintrag suchen
0009 =          txtaus  equ     09h     ;Zeichenkette an Bildschirm senden
0002 =          chraus  equ     02h     ;Zeichen an Bildschirm ausgeben
0100            org     0100h
                ;
                ;****** lokalen Stapel anlegen
0100 210000             lxi     h,0000h
0103 39                 dad     sp
0104 221602             shld    oldstk  ;alten Stackpointer retten
0107 211602             lxi     h,oldstk;neuer Stackpinter
010A F9                 sphl
                ;
                ;****** Meldetext ausgeben
010B 118C01             lxi     d,meld
010E 0E09               mvi     c,txtaus
0110 CD0500             call    bdos
                ;
                ;****** Ersten Directoryeintrag suchen
0113 11B201             lxi     d,fcb
0116 0E11               mvi     c,firdir
0118 CD0500     next:   call    bdos
011B FEFF               cpi     0FFh    ;kein Eintrag gefunden
011D CA6C01             jz      ende    ;
0120 87                 add     a       ;'a'ter Eintrag im Buffer gültig
0121 87                 add     a       ;mit 32 multiplizieren
0122 87                 add     a
0123 87                 add     a
0124 87                 add     a
0125 5F                 mov     e,a
0126 1600               mvi     d,00    ;de=relativer Start des Eintrag
0128 218000             lxi     h,dma   ;Start Buffer
012B 19                 dad     d       ;hl=absoluter Eintragstart
                ;
                ;****** gefundenen Eintrag ausgeben
                ;(Usernummer/gelöschter Eintrag)
012C 7E                 mov     a,m     ;Usernummer
012D 23                 inx     h
012E E5                 push    h
012F CD7101             call    hexaus  ;hexadezimal ausgeben
0132 3E20               mvi     a,20h
0134 CD8501             call    zeich   ;Zeichen ausgeben
0137 E1                 pop     h
                ;(Namen der Datei)
0138 060B               mvi     b,0bh   ;11 Namensbytes
013A 7E         loop1:  mov     a,m
013B 23                 inx     h       ;Namensbyte
013C E5                 push    h
013D C5                 push    b
013E CD8501             call    zeich   ;ausgeben
0141 C1                 pop     b
0142 E1                 pop     h
0143 05                 dcr     b
0144 C23A01             jnz     loop1   ;nächstes Namensbyte
```

```
                ;(Rest hexadezimal ausgeben)
0147 0614               mvi     b,14h   ;20 Bytes ausgeben
0149 C5         loop2:  push    b
014A E5                 push    h
014B 3E20               mvi     a,20h   ;Leerzeichen
014D CD8501             call    zeich
0150 E1                 pop     h
0151 7E                 mov     a,m     ;Byte aus Directory
0152 23                 inx     h
0153 E5                 push    h
0154 CD7101             call    hexaus  ;Hexadezimal ausgeben
0157 E1                 pop     h
0158 C1                 pop     b
0159 05                 dcr     b
015A C24901             jnz     loop2   ;nächstes Byte
                ;(Zeilenabschluß)
015D 3E0D               mvi     a,0dh   ;Wagenrücklauf
015F CD8501             call    zeich
0162 3E0A               mvi     a,0ah   ;Zeilenvorschub
0164 CD8501             call    zeich
                ;
                ;****** nächsten Eintrag suchen
0167 0E12               mvi     c,nexdir
0169 C31801             jmp     next
                ;
                ;****** Programmende
016C 2A1602     ende    lhld    oldstk  ;alten Stackpointer restaurieren
016F F9                 sphl
0170 C9                 ret             ;Rücksprung in den CCP
                ;
                ;****** Unterprogramm 1: Byte hexadezimal ausgeben
0171 F5         hexaus: push    psw
0172 0F                 rrc             ;höherwertiges Nibble
0173 0F                 rrc             ;auf Bits 0-3
0174 0F                 rrc
0175 0F                 rrc
0176 CD7A01             call    nibble
0179 F1                 pop     psw     ;niederwertiges Nibble
017A E60F       nibble: ani     0Fh
017C C630               adi     30h     ;Ascii-Codierung
017E FE3A               cpi     3ah     ;A bis F?
0180 DA8501             jc      zeich
0183 C607               adi     07h     ;Korrektur
                ;
                ;****** Unterprogramm 2: Zeichen unverändert ausgeben
0185 5F         zeich:  mov     e,a
0186 0E02               mvi     c,chraus
0188 CD0500             call    bdos    ;nicht mit 'JMP BDOS' abkürzen!
018B C9                 ret
                ;
                ;****** Meldetext
018C 566F6C6C73meld:    db      'Vollständige Directoryinformation'
01AD 0D0A0D0A24         db      13,10,13,10,'$'
                ;
                ;****** File-Contoll-Block
01B2 3F3F3F3F3Ffcb:     db      '??????????????'
01C1                    ds      15h
```

```
                ;
                ;****** lokaler Stapel
01D6                    ds      64      ;müßte reichen
0216            oldstk: end
```

Programm EXDIR2: die Turbo-Pascal-Version

Nachdem die Pascal-Version EXDIR2 als COM-Datei vorliegt, können Sie sie genauso starten wie die Assemblerversion. Auch die Bildschirmausgabe der beiden Programme ist völlig identisch und sie sind sogar genau gleich schnell.

Genau wie in einem Assemblerprogramm mit 'call bdos' können Sie auch in einem Pascal-Programm beliebige BDOS-Funktionen aufrufen. Dafür gibt es in Pascal die Funktionen/Prozeduren mit den Namen *BDOS* und *BDOSHL*. Der erste dabei zu übergebende Parameter entspricht dem Wert im Register C beim Assemblerbefehl, also der Funktionsnummer. Der (nicht unbedingt notwendige) 2. Parameter wird ins DE-Register übertragen. Der Assembler-Aufruf

```
MVI C,26!LXI D,10000!CALL 0005!
```

entspricht in Pascal also der Befehlsfolge

```
BDOS(26,1000)
```

Liefert die BDOS-Funktion ein Ergebnis, so erhalten Sie dieses, indem Sie die Pascal-Funktion einer Variablen zuweisen. Dazu gibt es 2 Varianten: *BDOS* gibt den 8-Bit-Wert im Register A zurück und *BDOSHL* den 16-Bit-Wert im Register HL.

Auch im Programm EXDIR2 wurden die BDOS-Operationen genauso verwendet, wie es in den Erklärungen zu EXDIR2 beschrieben wurde. Dazugekommen ist nur der Aufruf *BDOS(26,...)*. Er startet die BDOS-Operation zum legen des Datenpuffers in einen selbstgewählten Speicherbereich.

Enthält die Variable *fcb* etwa den Dateibeschreiber, dann müssen Sie der BDOS-Funktion zum Suchen eines Verzeichniseintrages die Adresse dieser Variablen übergeben. Diese Adresse können Sie mit der Funktion *ADDR(fcb)* bestimmen. Bei einer Zeichenkettenvariable gibt das erste Byte jedoch nicht das erste Datenbyte, sondern die Länge der Zeichenkettenvariable an. Erst im 2. Byte beginnt der sichtbare Inhalt der Zeichenkette. Beim BDOS-Aufruf müssen Sie also durch *ADDR(fcb)+1* dieses Längenbyte überspringen. Für die Variable *DMA*, die den Datenpuffer enthält, gilt dasselbe. Denkbar wäre aber auch, den Variablen durch eine Deklarierung mit *Absolute* einem selbstgewählten Speicherbereich zuzuordnen.

Die Variablen DMA und FCB wurden in diesem Programm nicht mit der *concat*-Prozedur gefüllt, sondern mit *fillchar*. Bei vielmaliger Wiederholung ein- und desselben Nicht-Ascii-Wertes führt das zu einem Geschwindigkeitsvorteil.

```
program dirleser;
{$c+}  (* Programm kann nach jeder Ausgabe abgebrochen werden *)

var fcb:string[36];       (* Raum fuer Dateibeschreiber *)
    dma:string[128];      (* 128-Byte Pufferspeicher *)
    dir:string[32];       (* Gefundener Verzeichniseintrag *)
    rc,i:byte;            (* 'rc' = Returncode 0,1,2,3 oder 255 *)

procedure hex(z:char);    (* Zeichencode hexadezimal ausgeben *)
var a,b:byte;
begin
  a:=ord(z) shr 4+48;     (* Ascii-Code fuer 0..9 ist 48..57 *)
  if a>57 then a:=a+7;    (* Korrektur fuer Buchstaben A..F: 65..70 *)
  b:=ord(z) and 15+48;    (* niederwertige Stelle *)
  if b>57 then b:=b+7;
  write(chr(a),chr(b),' ')
end;

begin
  writeln('Ausgeben der gesamten Directoryinformation');
  fillchar(dma,129,0);              (* Pufferspeicher loeschen *)
  fillchar(dma,1,128);
  fillchar(fcb,37,0);               (* FCB mit Fragezeichen fuellen *)
  fillchar(fcb,14,63);
  fillchar(fcb,1,36);
  bdos(26,addr(dma)+1);             (* Funktion 'SETDMA' *)
  rc:=bdos(17,addr(fcb)+1);         (* Funktion 'SEARCH FOR FIRST' *)
  while rc<>255 do
    begin
      dir:=copy(dma,rc*32+1,32);    (* Gesuchter Directoryeintrag *)
      hex(copy(dir,1,1));           (* Usernummer hexadezimal *)
      write(copy(dir,2,11),' ');    (* Dateiname *)
      for i:= 13 to 31 do
        hex(copy(dir,i,1));         (* restliche Bytes hexadezimal *)
      writeln;
      rc:=bdos(18,addr(fcb)+1)      (* Funktion 'SEARCH FOR NEXT' *)
    end;
end.
```

6.5.5 Sequentielles Lesen und Schreiben

Speziell zum sequentiellen Lesen und Schreiben gibt es nur 2 BDOS-Funktionen. Da mit diesen das wahlfreie Ansteuern eines bestimmten Records sehr umständlich ist, eignen sie sich wohl vorwiegend zum Arbeiten mit Textdateien oder ähnlichen. Bei diesen Dateitypen ist sowieso nur ein geradliniges Durcharbeiten von vorne bis hinten sinnvoll.

Sequentielles Lesen

Funktion: *Liest den aktuellen Record aus der Datei und stellt den Recordzeiger auf den nächsten Record.*

Der aktuelle 128-Byte-Record wird von der Diskette in den Datenpuffer kopiert. Danach wird der Dateibeschreiber so verändert, daß der nächste sequentielle Lesebefehl auf den darauffolgenden Record zugreift. Ohne weitere Maßnahmen werden so alle Records der Reihe nach bearbeitet.

Der Dateibeschreiber, den der Lesebefehl benötigt, muß einem gültigen Inhaltsverzeichniseintrag entsprechen. Das ist nur gewährleistet, wenn mit diesem Dateibeschreiber eine erfolgreiche Eröffnungsprozedur durchgeführt wurde. Wenn Sie sich nicht daran halten, werden Sie nur zufällige Daten von der Diskette lesen.

Schon beim Eröffnen kann eine Recordnummer und eine Extentnummer vorgegeben werden. In diesem Fall wird als erstes der angegebene Record gelesen und erst danach sequentiell fortgefahren. So ist in beschränktem Maße auch ein wahlfreier Zugriff möglich.

Das EX-Byte gibt die niederwertigen Bits der logischen Extentnummer an und kann Werte zwischen 0 und 31 annehmen. Obwohl das S2-Byte im Inhaltsverzeichniseintrag die höherwertigen 5 Bits des Extents angibt, wird das entsprechende Byte im Dateibeschreiber vor jeder Leseoperation zu Null gesetzt. Mit diesem Lesebefehl kann also nur auf die ersten 32 logischen Extents einer Datei zugegriffen werden. Damit ist 32*16 KByte = 512 KByte die maximale Länge einer sequentiellen Datei. Im Normalfall dürfte das völlig ausreichend sein, denn längere Dateien sind sowieso nur mit wahlfreiem Zugriff in zumutbarer Zeit zu bearbeiten.

Sofern die Datei korrekt eröffnet war, tritt ein Fehler nur auf, wenn das Dateiende überschritten wurde oder versucht wurde, Daten aus einem 'Loch' in der Datei zu lesen.

Aufruf: c:= 20
de:= Anfangsadresse FCB

Rückgabe a:= 0, wenn das Lesen erfolgreich war,
ungleich 0, wenn das Lesen nicht erfolgreich war.

Beispiel:

```
lxi  d,fcb     ;korrekt eröffneter Dateibeschreiber
mvi  c,14h
call bdos
ana  a
jnz  error
```

---> Der Datenpuffer enthält einen Record aus der im Dateibeschreiber genannten Datei.

Sequentielles Schreiben

Funktion: *Beschreibt den aktuellen Record mit Daten und stellt den Recordzeiger auf den nächsten Record.*

Der Inhalt des Datenpuffers wird in den gerade aktuellen Record auf der Diskette kopiert. Danach wird der Dateibeschreiber so verändert, daß der nächste sequentielle Schreibbefehl auf den darauffolgenden Record wirkt. Ohne weitere Maßnahmen werden so alle Records der Reihe nach bearbeitet.

Für den Aufbau des Dateibeschreibers gelten beim sequentiellen Schreiben dieselben Regeln wie beim sequentiellen Lesen. Insbesonders muß auch hier der Dateibeschreiber korrekt geöffnet sein. Wenn Sie darauf nicht achten, schreibt die BDOS-Operation in einen zufälligen Record auf der Diskette. Wenn sie nicht gerade mit einer leeren Diskette arbeiten, werden dabei wichtige Daten zerstört. Die Beschränkung auf 512 KByte als maximale Dateilänge gilt auch für den Schreibbefehl.

Sind im gerade aktuellen Record schon Daten enthalten, werden diese bei einer Schreiboperation ohne Warnung überschrieben. Liegt der aktuelle Record hinter dem Dateiende oder in einem Loch der Datei, so wird die Datei ergänzt. Zusätzlich können sich Lese- und Schreiboperationen beliebig abwechseln. So können Sie sowohl eine Datei verlängern/neu erstellen als auch überschreiben oder selektiv einige schon bestehende Records lesen und andere überschreiben.

Mit dieser BDOS-Operation können Sie ebenfalls unter bestimmten Bedingungen wahlfreie Records der Datei ansteuern. Einen neuen physischen Extent können Sie jedoch nicht erstellen. Es ist nur möglich, Records in schon bestehenden Extents zu erzeugen/überschreiben. Eine Ausnahme gibt es nur unmittelbar am Ende eines physischen Extents: Wenn Ihre Disketten in 16-KByte-Extents organisiert sind, wird nach dem Schreiben des Rec-

ords mit der Nummer 127 automatisch der Extent für die Nummern 128 bis 255 erstellt. So können Sie sich langsam zu höheren Extentnnummern vortasten. Wenn Ihre Disketten etwa in 64 KByte langen physischen Extents organisiert sind, tritt dieser Effekt erstmals beim Record mit der Nummer 511 auf.

Sofern die Datei korrekt eröffnet ist, tritt ein Fehler nur auf, wenn ein noch nicht bestehender Extent adressiert wurde oder die Diskette voll ist.

Aufruf: c:= 21
de:= Anfangsadresse FCB

Rückgabe a:= 0, wenn das Schreiben erfolgreich war,
ungleich 0, wenn die Daten nicht geschrieben werden konnten.

Beispiel:

```
lxi  d,fcb          ;korrekt eröffneter FCB
mvi  c,15h
call 0005h
ana  a
jnz  error
```

---> Die Daten im Datenbuffer wurden in den aktuellen Record der im FCB genannten Datei übertragen.

6.5.6 Beispiele zum Arbeiten mit sequentiellen Dateien

Auf Ihrer CP/M-Systemdiskette befindet sich eine Datei mit dem Namen DUMP.ASM. Das ist das Assemblerlisting eines Programms, das eine beliebige Datei von der Diskette liest und hexadezimal auf dem Bildschirm ausgibt. Dieses Programm sollten Sie sich ruhig einmal anschauen, wenn Sie wissen wollen wie man in der Praxis eine Datei von der Diskette liest. Beispiele zum Schreiben in eine Datei sehen Sie im folgenden. Diese Programme sind auf jedem beliebigen Computer mit CP/M 2.2-Betriebssystem lauffähig und nicht etwa nur auf einem CPC 464 oder CPC 664.

Eine sequentielle Datei erzeugen

Dieses Programm erstellt oder erweitert eine Ascii-Datei mit dem Namen TEST.TXT. Gibt es auf der Diskette noch keine Datei mit diesem Namen, so wird sie neu erstellt und mit Ascii-Zeichen gefüllt: In alle 128 Bytes des ersten Records werden Leerzeichen (Code 20h) geschrieben, in den zweiten Record Ausrufezeichen (Code 21h) und so weiter. In den letzten Record wird das Zeichen '~' beziehungsweise 'ß' mit dem Code 7Eh geschrieben.

Nach dem Programmende befinden sich also 95 Records in der Datei. Mit 'TYPE TEST.TXT' können Sie sich dieses Ergebnis anschauen.

Befand sich schon vor dem Programmstart eine Datei TEST.TXT auf der Diskette, wird die alte Datei nicht überschrieben, sondern die neuen Daten werden am Ende angehängt. Bei zweimaligem Programmstart befinden sich also 190 Records mit dem Ascii-Zeichensatz in der Datei. An andere Dateien werden die 90 Records natürlich genauso angehängt. Beachten Sie aber, daß der Type-Befehl und die meisten anderen Programme, die Textdateien verarbeiten, einen eventuellen Code 1Ah in einer Datei als vorzeitiges (softwaremäßiges) Dateiende interpretieren. Die zusätzlichen Daten können in so einem Fall nicht ausgetippt werden.

Wie üblich wird im Programm zunächst ein lokaler Stapel angelegt. Danach wird mit Hilfe der BDOS-Funktion 11h 'Suchen nach dem ersten Eintrag' getestet, ob sich die Datei TEST.TXT schon auf der Diskette im Bezugslaufwerk befindet. Wenn nein, kann die Datei bedenkenlos neu erzeugt werden.

Befindet sich TEST.TXT aber schon auf der Diskette würde ein weiteres Erzeugen zwei Dateien unter demselben Namen im Inhaltsverzeichnis bewirken. Das sollten Sie ruhig einmal ausprobieren, indem Sie diese Sicherheitsabfrage weglassen.

In diesem Fall wird also die Datei nur geöffnet. Die BDOS-Funktion 33h ermittelt die Nummer des ersten nicht mehr von der Datei belegten Records und legt diese in den Bytes R0 und R1 des Dateibeschreibers ab. Die Funktion 'Wahlfreien Record schreiben' verlangt in genau diesen Bytes die Angabe der Recordnummer, in den geschrieben werden soll. Als Trick wird jetzt genau diese Funktion (mit der Nummer 22h) ausgeführt, obwohl sich im Datenpuffer noch unsinnige Daten befinden. Als Nebeneffekt wird dabei der Dateibeschreiber so initialsisiert, daß auch sequentieller Schreibbefehle erlaubt sind. Der erste sequentielle Schreibbefehl überschreibt dann genau diesen unsinnigen Record, unmittelbar hinter dem Dateiende. Die alte Datei kann dann mit derselben Routine genauso fortgesetzt werden wie die neu erzeugte Datei.

Danach wird in einer Schleife der Datenpuffer mit einem Zeichen aufgefüllt und in die Datei geschrieben. Nach dem letzten Record darf nicht vergessen werden, die Datei wieder zu schließen. Sonst sind die neuen Daten im Inhaltsverzeichnis nicht registriert und würden verlorengehen.

Nach jedem Diskettenbefehl sollten Sie überprüfen, ob auch kein Fehler aufgetreten ist. Nur so ersparen Sie sich viel Ärger. Im Assemblerlisting wurde das auch konsequent durchgeführt. Insbesondere gilt das für den

Dateiöffnungsbefehl: Eigentlich müßte das Öffnen immer funktionieren, weil unmittelbar vorher überpüft wurde, ob die zu öffnende Datei auch vorhanden ist. Der Test ist aber eine Sicherheitsmaßnahme: Wenn das Öffnen aufgrund eines Programmierfehlers nicht funktionierte, wird so vermieden, daß auf der gesamten Diskette unsinnige Daten verteilt werden. Außerdem gewinnen Sie so einen zusätzlichen Hinweis, wo der Fehler steckt.

Das Listing von WRITESEQ.ASM

```
                ;Die Datei 'TEST.TXT' sequentiell beschreiben
                ;  (jeder Record wird mit einem anderen Buchstaben gefüllt)
                ;
0005 =          bdos    equ     0005h
0080 =          dma     equ     0080h
005C =          fcb     equ     005Ch
0009 =          type    equ     09h             ;Zeichenkette auf Bildschirm
000F =          open    equ     0Fh             ;Datei öffnen
0010 =          close   equ     10h             ;Datei schließen
0011 =          search  equ     11h             ;Directoryeintrag suchen
0015 =          write   equ     15h             ;sequentielles Schreiben
0016 =          make    equ     16h             ;Datei erzeugen
0022 =          rndwr   equ     22h             ;Wahlfreies Schreiben
0023 =          size    equ     23h             ;Dateigröße errechnen
0100                    org     0100h
                ;
                ;****** Initialisierung: Lokalen Stapel anlegen
0100 210000             lxi     h,0
0103 39                 dad     sp              ;hl enthält alten Stapel
0104 228102             shld    oldstk          ;Zwischenspeichern
0107 318102             lxi     sp,oldstk       ;neuer Stapel
                ;
                ;****** Datei 'TEST.TXT' suchen
010A 111D02             lxi     d,efcb          ;erweiterter Dateibeschreiber
010D 0E11               mvi     c,search        ;ersten Eintrag suchen
010F CD0500             call    bdos
0112 3C                 inr     a               ;Vergleich mit 'FFh'
0113 C22501             jnz     found
                ;
                ;****** Datei 'TEST.TXT' nicht gefunden
0116 111D02             lxi     d,efcb
0119 0E16               mvi     c,make          ;Datei neu erzeugen
011B CD0500             call    bdos
011E 3C                 inr     a               ;Vergleich mit 'FFh'
011F CA8001             jz      full            ;Directory voll
0122 C34601             jmp     work
                ;
                ;****** Datei 'TEST.TXT' schon vorhanden
0125 111D02     found:  lxi     d,efcb
0128 0E0F               mvi     c,open          ;Datei eröffnen
012A CD0500             call    bdos
012D 3C                 inr     a               ;Vergleich mit 'FFh'
012E CA7A01             jz      nodat           ;Nicht gefunden (dürfte nicht vor-
0131 111D02             lxi     d,efcb          ;kommen, aber sicher ist sicher
0134 0E23               mvi     c,size          ;Dateigröße errechnen
```

```
0136 CD0500             call    bdos            ;Größe steht in R0 und R1
0139 111D02             lxi     d,efcb
013C 0E22               mvi     c,rndwr         ;Wahlfreies Schreiben in durch
013E CD0500             call    bdos            ;R0 und R1 gegebenen Record
0141 E607               ani     a               ;Nur 0 ist ok.
0143 C28001             jnz     full            ;könnte genauer analysiert werd
                ;(jetzt kann die Datei sequentiell fortgesetzt werden)
                ;
                ;****** Datei mit neuen Daten füllen bzw. fortsetzen
0146 3E20       work:   mvi     a,' '           ;Zeichen für 1. Record
0148 F5         record: push    psw             ;Zeichen speichern
                ;(Datenpuffer mit gewähltem Zeichen füllen)
0149 218000             lxi     h,dma           ;Anfangsadresse Datenpuffer
014C 0680               mvi     b,80h           ;Länge Datenpuffer
014E 77         fill:   mov     m,a
014F 23                 inx     h
0150 05                 dcr     b
0151 C24E01             jnz     fill
                ;(Datenpuffer sequentiell auf Diskette schreiben)
0154 111D02             lxi     d,efcb
0157 0E15               mvi     c,write         ;aktuelle Recordnummer wird
0159 CD0500             call    bdos            ;automatisch erhöht
015C E607               ani     a               ;Nur 0 ist ok.
015E C28001             jnz     full            ;Fehler
                ;(eventuell weiteren Record beschreiben)
0161 F1                 pop     psw             ;Zeichen im letzten Record
0162 3C                 inr     a               ;nächstes Zeichen in nächsten
0163 FE7F               cpi     7Fh             ;alle Asciizeichen schon durch?
0165 C24801             jnz     record
                ;
                ;****** Datei schließen
0168 111D02             lxi     d,efcb
016B 0E10               mvi     c,close         ;Datei schließen
016D CD0500             call    bdos
0170 3C                 inr     a               ;Test auf 'FFh'
0171 CA7A01             jz      nodat           ;dürfte nicht passieren
                ;
                ;****** Korrekte Programmbeendigung
0174 118D01             lxi     d,mess1         ;Meldung: Programmende
0177 C38301             jmp     mess
                ;
                ;****** Fehler 1: Datei nicht gefunden
017A 11C901     nodat:  lxi     d,mess2         ;Adresse 1. Botschaft
017D C38301             jmp     mess
                ;
                ;****** Fehler 2: Diskette oder Directory voll
0180 11EE01     full:   lxi     d,mess3         ;Adresse 2. Botschaft
0183 0E09       mess:   mvi     c,type          ;Zeichenkette ausgeben
0185 CD0500             call    bdos
                ;
                ;****** Programm beenden
0188 2A8102             lhld    oldstk          ;alten Stackpointer zurück
018B F9                 sphl                    ;restaurieren
018C C9                 ret                     ;Programmende ohne Warmstart
                ;
                ;****** Texte der Fehlermeldungen
018D 416C6C6573mess1:   db      'Alles in Ordnung: Datei ''TEST.TXT'' erzeugt '
01B8 6F64657220         db      'oder erweitert',13,10,'$'
```

```
01C9 2A2A2A2A2Amess2:   db      '***** Fehler: Datei nicht gefunden',13,10,'$'
01EE 2A2A2A2A2Amess3:   db      '***** Fehler: Diskette oder Verzeichnis voll'
021A 0D0A24             db      13,10,'$'

                ;
                ;****** Im Programm verwendeter Dateibeschreiber
021D 00         efcb:   db      0               ;Schreiben auf Bezugslaufwerk
021E 5445535420         db      'TEST    TXT'   ;11 Zeichen Dateiname
0229 000000             db      0,0,0           ;erste benötigte Extentnummer
022C                    ds      17              ;beliebiger Inhalt
023D 00                 db      0               ;1. benötigter Record im Extent
023E                    ds      3               ;erweiterter FCB: R0, R1, R2
                ;
                ;****** Lokaler Stapelbereich
0241                    ds      64              ;Platz für 32 Stapeleinträge
0281            oldstk: ds      2               ;Zwischenspeicher: alte Positio
0283                    end
```

Eine sequentielle Datei wahlfrei beschreiben

Die Funktionsweise des Programms RAND1.ASM

Der Anfang des RAND1-Programms entspricht im Aufbau dem Programm WRITESEQ.ASM. Weggelassen wurde nur das Aufsuchen des Dateiendes, wenn TEST.TXT schon vorhanden ist. Das wäre auch sinnlos, denn später wird sowieso auf beliebige Records zugegriffen. Im Programmteil zwischen dem Öffnen und Schließen der Datei wird mit Hilfe der BDOS-Funktionen 'Zeichenkette ausgeben' und '1 Zeichen von der Tastatur einlesen' ein einfaches Auswahl-Menü realisiert. Abhängig von Ihrer Auswahl wird die Bearbeitung an den Stellen 'Record in die Datei schreiben' und 'Record aus der Datei lesen' fortgesetzt.

Beide Programmteile benützen das Unterprogramm 'Dezimalzahl einlesen'. In diesem Unterprogramm werden in einer Schleife einzelne Zeichen von der Tastatur eingelesen und sofort analysiert. Wurde keine Ziffer zwischen 0 und 9 eingegeben, wird das Unterprogramm beendet. Im anderen Fall handelt es sich um die (bisher) niederwertigste Stelle der Zahl. Der schon vorher eingegebene Wert ist dann höherwertig und wird mit 10 multipliziert und zur neuen Ziffer addiert. Das Gesamtergebnis wird dem Hauptprogramm im Register HL zurückgegeben. Beim Versuch, eine größerer Zahl als 65535 einzugeben, ergibt sich keine Fehlermeldung. Die überschüssigen Bits (der binären Darstellung) werden einfach ignoriert.

Im Programmteil 'Record in die Datei schreiben' wird zunächst mit Hilfe der BDOS-Funktion Nr. 10 eine ganze Zeichenkette von der Tastatur in einen Eingabepuffer gelesen. Der Datenbereich des Puffers ist mit dem Diskettenpuffer identisch. An der 1. Stelle des Puffers steht 128 Bytes (= 1 Record) für die maximale Eingabelänge. Im 2. Byte steht nach der Eingabe

die tatsächlich benötigte Länge. Anhand dieser Information werden die überschüssigen Bytes, die sinnlose Werte enthalten, mit Nullen gelöscht.

Im Programmteil 'Record aus der Datei lesen' wird jedes Zeichen des erhaltenen Datenbereiches einzeln auf dem Bildschirm ausgegeben. Dabei werden automatisch die Bits 7 ausgeblendet und Steuerzeichen durch unschädliche Ascii-Zeichen ersetzt. So wird selbst bei sinnlosen Daten ein Absturz des Computers vermieden. Ist nach dem Lesen des Records aus der Datei das Carry-Bit gesetzt, war der Lesevorgang nicht erfolgreich. Die Ausgabe der Daten auf den Bildschirm wird in diesem Fall übersprungen.

Das eigentliche Schreiben und Lesen in/aus einem Record der Datei erfolgt in weiteren Unterprogrammen am Ende des Gesamtprogrammes. Die Programmteile, die die Ausgaben auf dem Bildschirm und das Einlesen über die Tastatur erledigen, übergeben diesen Unterprogrammen nur die Nummer des Records, der behandelt werden soll. Wie die Unterprogramme das Schreiben/Lesen im Detail erledigen ist für die aufrufenden Programme völlig uninteressant. Im Programm RAND1.ASM wurden dazu trotzdem Befehle zum sequentiellen Lesen/Schreiben verwendet. Der wahlfreie Zugriff kommt durch geschicktes Verändern des Dateibeschreibers vor jedem Zugriff zustande.

Ein logischer Extent beinhaltet 128 Bytes. Wenn ein neuer Record innerhalb dieses Extents adressiert werden soll, braucht nur die Nummer des neuen Records in das Byte CR des Dateibeschreibers übertragen werden. Wenn aber der neue Record in einem anderen Extent liegt, genügt die Änderung nur des EX-Bytes im Dateibeschreiber nicht. Mit geändert werden müssen auch die Nummern der reservierten Records im neuen Extent. Die einzige ohne großen Aufwand mögliche Methode dazu ist das Neueröffnen der Datei nach der Änderung des EX-Bytes. Dabei werden nicht etwa die Blockreservierungen für den ersten Extent in den Dateibeschreiber kopiert, sondern gleich die für den aktuellen Extent.

Wenn im alten Extent neue Records auf die Diskette geschrieben wurden, sind die dadurch notwendigen Blockreservierungen nur im Dateibeschreiber, nicht aber auf der Diskette enthalten. Bei dieser Neueröffnung gehen aber die Daten im alten Dateibeschreiber verloren. Vor der Neueröffnung muß also in diesem Fall die Datei mit den Daten des alten Dateibeschreibers geschlossen werden. Genau so wird auch im Programm RAND1.ASM vorgegangen. Mit dieser Methode können nur die Extents mit den Nummern 0 bis 31 behandelt werden. Das ist nicht nur eine Einschränkung dieses Programms, sondern eine Einschränkung des CP/M-Betriebssystemes. Eine Änderung des S2-Bytes (um die übrigen Extents anzusprechen) wäre völlig sinnlos.

Nicht gezielt erzeugen können Sie neue Extents der Datei. Beim Versuch, einen nicht vorhandenen Extent zu eröffnen, erhalten Sie nur eine Fehlermeldung. Die fehlerhafte Extentnummer bleibt aber im Dateibeschreiber erhalten. Beim nachfolgenden Versuch, einen weiteren (richtigen) Extent zu öffnen, müßte erst der falsche Extent geschlossen werden - was natürlich ebenfalls unmöglich ist. Alle Schreib- und Leseoperationen in der Datei wären dann blockiert. Um das zu vermeiden, wird in diesem Fall der immer vorhandene Extent mit der Nummer 0 eröffnet. Sollte das auch nicht gelingen wird das Programm RAND1.ASM abgebrochen. Wird 'nur' ein falscher Record verlangt, so wird das Programm fortgesetzt um noch eine Korrektur zu ermöglichen.

Wenn Sie Zeit haben, können Sie das Programm noch verbessern:

- Wenn vor dem Versuch, einen neuen Extent zu eröffnen der alte Dateibeschreiber irgendwo zwischengespeichert wird, können Sie ihn im Fehlerfall ohne Eröffnen des Records Nummer 0 restaurieren. Damit wird ein Diskettenzugriff eingespart.
- Das vorsorgliche Schließen des alten Records kann entfallen, wenn dieser nicht geändert wurde.
- Beim erstmaligen Eröffnen der Datei kann der Test auf Vorhandensein entfallen, wenn die Datei zunächst nur auf 'gut Glück' eröffnet wird und nur im Fehlerfall neu erzeugt wird.
- Es könnten mehrere verschiedene Fehlerursachen unterschieden werden.

Diese Methoden sind aber wegen des höheren Verwaltungsaufwandes (Programmier-) fehleranfälliger und nur zu empfehlen, wenn Sie das letzte aus Ihrem Computer herausholen wollen. Natürlich können Sie in RAND1.ASM zum Experimentieren auch absichtlich Fehler einbauen und deren Auswirkungen studieren. Besonders dankbar ist hier das 'Vergessen' des Eröffnens einer Datei.

Experimentieren mit dem Programm RAND1.ASM

Wenn RAND1 als COM-Datei vorliegt, können Sie es wie jedes CP/M-Programm nur durch die Namensangabe starten. Danach erscheint ein einfaches Menü und Sie können durch die Nummern 1 bis 3 verschiedene Aktionen anwählen. Bei den Experimenten sollten Sie besonders darauf achten, wann das Diskettenlaufwerk zu laufen beginnt und wann Schreib-/Lesefehler gemeldet werden. Dadurch können Sie sicher Ihr Verständnis der Diskettenverwaltung noch vertiefen. Versuchen Sie dabei ruhig auch, noch nicht geschriebene Records zu lesen oder einen Record mit der

Nummer 1000000 zu adressieren. Passieren kann dabei nichts. Einige besondere Erscheinungen werden jetzt besprochen.

Die Fehlermeldung 'Extent noch nicht vorhanden' bedeutet, daß Sie eine zu große Recordnummer eingegeben haben. Diese Grenze ist aber nicht endgültig. Nach einigen Vorbereitungen können Sie dann den gewünschten Record doch noch beschreiben. Beim Schreiben bedeutet 'Fehler beim Lesen/Schreiben aufgetreten', daß Ihre Diskette voll ist. Tritt dieser Fehler beim Lesen auf, wurde versucht, einen noch nicht geschriebenen Record einzulesen.

Wenn vor dem Programmstart die Datei TEST.TXT noch nicht vorhanden war, können Sie das folgende Experiment durchführen: Schreiben Sie zunächst irgendetwas in den Record Nummer 7. Obwohl das dann der einzige wirklich in der Datei enthaltene Record ist, können Sie danach auch die Records 0 bis 6 lesen, die aber nur sinnlose Daten enthalten. Das liegt daran, daß beim Schreiben des Records gleich ein ganzer Block reserviert wurde und alle in diesem Block enthaltenen Daten als zu der Datei gehörig betrachtet werden. Bei einer Blocklänge von 1 KByte sind davon jeweils 8 Records betroffen, bei größeren Blöcken entsprechend mehr. In einem eventuellen Anwenderprogramm ist also eine zusätzliche Routine nötig, die herausfindet, welche Records nun wirklich gültig sind.

In der Regel werden die ungültigen Records irgendwelche Reste aus schon gelöschten Dateien enthalten. Manchmal werden Sie aber auch einen Ausschnitt aus dem Inhaltsverzeichnis der Diskette erhalten. Das ist dann aber nicht wirklich das Inhaltsverzeichnis, sondern nur eine Kopie davon: Auf die Diskette wird (bei der Schneider 3"-Floppydisk) nur in Paketen von 512 Bytes geschrieben. Wenn Sie nur einen einzigen 128-Byte-Record ändern möchten, wird normalerweise zunächst ein ganzer 512-Byte-Sektor gelesen, die 128 Bytes ersetzt und der Sektor zurückgespeichert. Wenn mit diesem Record aber ein bisher nicht vorhandener Block adressiert wurde, enthält dieser Sektor nur unsinnige Daten. Das CP/M-Betriebssystem unterläßt in diesem Fall automatisch das vorbereitende Lesen, so daß die alten Daten im Sektorpuffer erhalten bleiben. Da am häufigsten auf die Directoryspur der Diskette zugegriffen wird, wird das in der Regel ein Ausschnitt aus dem Inhaltsverzeichnis sein.

Beschreiben Sie jetzt den Record mit der Nummer 120. Das gelingt einwandfrei. Das anschließende Lesen des Records mit der Nummer 90 mißlingt aber. Eine Datei kann also wirklich 'Löcher enthalten'. Bei einer rein sequentiellen Dateibehandlung können nur die Records bis zum ersten Loch gelesen werden. Wenn schon der erste Block der Datei ein Loch ist, scheint die Datei bei einem Befehl wie TYPE völlig leer zu sein.

Wenn Ihre Disketten in physikalischen Extents zu je 16 KByte organisiert sind, können Sie das folgende Experiment durchführen: Versuchen Sie, den Record mit der Nummer 130 (liegt im 2. Extent der Datei) zu beschreiben. Sie werden die Fehlermeldung 'Extent noch nicht vorhanden' erhalten. Ein neuer Extent wird also nicht zu beliebigen Zeiten automatisch erzeugt. Jetzt sollten Sie den Record mit der Nummer 127 beschreiben. Dieser liegt gerade am Ende eines physikalischen Extents. Das besondere ist, daß durch das sequentielle Schreiben dabei automatisch der nächste Extent eröffnet wird. Danach gelingt das Beschreiben des Records mit der Nummer 130 einwandfrei. Dasselbe wiederholt sich an der nächsten Extentgrenze zwischen den Nummer 255/256. Nach und nach können Sie sich so bis zu beliebigen Recordnummern hochhangeln. Diese zusätzlichen Extents bleiben auch nach einem Abbruch und Neustart des Programms erhalten. Sie brauchen diese Prozedur also nur ein einziges Mal durchzuführen. Wenn Ihre Disketten (wie zum Beispiel mit dem Vortex-Floppydiskontroller) in 64 KByte langen physischen Extents verwaltet werden, tritt dieser Effekt erstmals an der Grenze zwischen den Records Nr. 511 und 512 auf.

Aufgrund der Beschränkung von sequentiellen Dateien auf die logische Länge von 512 KByte können Sie mit diesem Programm maximal 4096 Records pro Datei verwalten. Obwohl es so aussieht, als ob Sie nach den oben geschilderten Vorbereitungen beliebige Recordnummern eingeben können, werden alle im Bereich zwischen 0 und 4095 abgelegt. Der Record mit der Nummer 5000 würde also den Record mit der Nummer 5000-4096 = 904 überschreiben. Das können Sie ganz leicht durch beschreiben des Records Nr. 4096 mit einem leicht merkbaren Text und anschließendem Lesen des Records Nr. 0 feststellen.

Achten Sie beim Experimentieren mit dem RAND1-Programm auch auf das Verhalten des Diskettenlaufwerkes: Sie werden feststellen, daß der Sektorpuffer erst auf die Diskette zurückgeschrieben wird, wenn wirklich ein neuer Sektor adressiert wird. Die Records mit den Nummern 124 bis 127 liegen in einem Sektor. Solange Sie in diesem Bereich bleiben, können Sie beliebig Records aus diesem Sektor lesen und schreiben, ohne daß das Diskettenlaufwerk anspringt. Erst bei der Anwahl eines neuen Sektors wird der alte zurückgespeichert. Der Leseversuch eines nicht vorhandenen Records (im selben Extent) gilt dabei nicht als Sektorwechsel. Auch die Meldung 'Fehler beim Lesen/Schreiben aufgetreten' erhalten Sie, ohne daß sich das Laufwerk zu drehen beginnt.

Solange Sie in einem Extent bleiben, werden die Records auf der Diskette direkt angefahren. Nur bei einem Extentwechsel merken Sie durch die lauten Geräusche der Floppydisk, daß zwischendurch das Inhaltsverzeichnis angesteuert wird. Der Zugriff auf einen beliebigen Record der Datei dauert

so maximal 2 Sekunden (wenn der Motor erst anlaufen muß). Bei laufendem Motor und ohne zwischenzeitlichen Verzeichniszugriffen läuft das Schreiben/Lesen aber wesentlich schneller ab.

Das Listing von RAND1.ASM

```
                ;Die Datei 'TEST.TXT' mit Befehlen zum sequentiellem Lesen
                ;  wahlfrei beschreiben und lesen
                ;
                ;****** Konstanten: Nummern der CP/M-Funktionen
0005 =          bdos    equ     0005h
0001 =          rdcha   equ     01h             ;Zeichen von der Tastatur lesen
0002 =          wrcha   equ     02h             ;Zeichen an Bildschirm ausgeben
0009 =          type    equ     09h             ;Zeichenkette auf Bildschirm
000A =          rdbuf   equ     0Ah             ;Zeichenkette von der Tastatur
000F =          open    equ     0Fh             ;Datei öffnen
0010 =          close   equ     10h             ;Datei schließen
0011 =          search  equ     11h             ;Directoryeintrag suchen
0014 =          read    equ     14h             ;sequentielles Lesen
0015 =          write   equ     15h             ;sequentielles Schreiben
0016 =          make    equ     16h             ;Datei erzeugen
001A =          setdma  equ     1Ah             ;Pufferspeicherlage wählen
0021 =          randrd  equ     21h             ;wahlfreies Lesen
0022 =          randwr  equ     22h             ;wahlfreies Schreiben
0100            org     0100h
                ;
                ;****** Initialisierung: Lokalen Stapel anlegen
0100 210000             lxi     h,0
0103 39                 dad     sp              ;hl enthält alten Stapel
0104 22DC03             shld    oldstk          ;Zwischenspeichern
0107 31DC03             lxi     sp,oldstk       ;neuer Stapel
                ;
                ;****** Datei 'TEST.TXT' suchen
010A 111C03             lxi     d,dma           ;Pufferspeicher festlegen
010D 0E1A               mvi     c,setdma
010F CD0500             call    bdos
0112 11F602             lxi     d,efcb          ;erweiterter Dateibeschreiber
0115 0E11               mvi     c,search        ;ersten Eintrag suchen
0117 CD0500             call    bdos
011A 3C                 inr     a               ;Vergleich mit 'FFh'
011B C22D01             jnz     found
                ;
                ;****** Datei 'TEST.TXT' nicht gefunden
011E 11F602             lxi     d,efcb
0121 0E16               mvi     c,make          ;Datei neu erzeugen
0123 CD0500             call    bdos
0126 3C                 inr     a               ;Vergleich mit 'FFh'
0127 CA7501             jz      full            ;Directory voll
012A C33901             jmp     menuc
                ;
                ;****** Datei 'TEST.TXT' schon vorhanden
012D 11F602     found:  lxi     d,efcb
0130 0E0F               mvi     c,open          ;Datei eröffnen
0132 CD0500             call    bdos
0135 3C                 inr     a               ;Vergleich mit 'FFh'
0136 CA6F01             jz      nodat           ;(dürfte nicht vorkommen)
```

```
                        ;
                        ;****** Datei mit neuen Daten füllen bzw. fortsetzen
0139 118201     menue:  lxi     d,help          ;Menü ausgeben
013C 0E09               mvi     c,type
013E CD0500             call    bdos
0141 0E01               mvi     c,rdcha         ;Zeichen von Tastatur abwarten
0143 CD0500             call    bdos
0146 F5                 push    psw             ;Zeichen speichern
0147 117904             lxi     d,crlf          ;neue Zeile
014A 0E09               mvi     c,type
014C CD0500             call    bdos
014F F1                 pop     psw             ;Zeichen zurück
0150 D631               sui     '1'             ;Code für Record lesen
0152 CADE03             jz      lesen
0155 3D                 dcr     a               ;'2' ist Code für Record schrei
0156 CA1504             jz      schrib
0159 3D                 dcr     a               ;'3' ist Code für Programmende
015A C23901             jnz     menue           ;falsche Eingabe -> wiederholen
                        ;
                        ;****** Datei schließen
015D 11F602             lxi     d,efcb
0160 0E10               mvi     c,close         ;Datei schließen
0162 CD0500             call    bdos
0165 3C                 inr     a               ;Test auf 'FFh'
0166 CA6F01             jz      nodat           ;dürfte nicht passieren
                        ;
                        ;****** Korrekte Programmbeendigung
0169 111A02             lxi     d,mess1         ;Meldung: Programmende
016C C37801             jmp     mess
                        ;
                        ;****** Fehler 1: Datei nicht gefunden
016F 115302     nodat:  lxi     d,mess2         ;Adresse 1. Botschaft
0172 C37801             jmp     mess
                        ;
                        ;****** Fehler 2: Diskette oder Directory voll
0175 11C702     full:   lxi     d,mess3         ;Adresse 2. Botschaft
0178 0E09       mess:   mvi     c,type          ;Zeichenkette ausgeben
017A CD0500             call    bdos
                        ;
                        ;****** Programm beenden
017D 2ADC03             lhld    oldstk          ;alten Stackpointer zurück
0180 F9                 sphl                    ;restaurieren
0181 C9                 ret                     ;Programmende ohne Warmstart
                        ;
                        ;****** Texte in der Hauptprogrammschleife
0182 0D0A0D0A46help:    db      13,10,13,10,'Folgende Aktionen stehen zur '
01A3 4175737761         db      'Auswahl: ',13,10
01AE 09313A2052         db      9,'1: Record aus der Datei lesen',13,10
01CE 09323A2052         db      9,'2: Record in die Datei schreiben',13,10
01F1 09333A2050         db      9,'3: Programmende',13,10
0203 2020202042         db      '    Bitte wählen Sie: $'
021A 416C6C6573mess1:   db      'Alles in Ordnung: Datei ''TEST.TXT'' korrekt '
0245 6765736368         db      'geschlossen',13,10,'$'
0253 2A2A2A2A2Amess2:   db      '***** Fehler: Datei nicht gefunden',13,10
0277 4163687475         db      'Achtung: dürfte eigentlich nicht auftreten, '
02A3 7761687273         db      'wahrscheinlich ein Programmfehler',13,10,'$'
02C7 2A2A2A2A2Amess3:   db      '***** Fehler: Diskette oder Verzeichnis voll'
02F3 0D0A24             db      13,10,'$'
```

```
                ;
                ;****** Im Programm verwendeter Dateibeschreiber
02F6 00         efcb:   db      0               ;Schreiben auf Bezugslaufwerk
02F7 5445535420         db      'TEST    TXT'   ;11 Zeichen Dateiname
0302 00         efcbex: db      0               ;Extent-Nr, niederwertig
0303 00                 db      0
0304 00         efcbs2: db      0               ;Extent-Nr, höherwertig
0305                    ds      17              ;beliebiger Inhalt
0316 00         efcbcr: db      0               ;1. benötigter Record im Extent
0317 000000     rand:   db      0,0,0           ;erweiterter FCB: R0, R1, R2
                ;
                ;****** Vom Programm benutzte Zwischenspeicher
031A 80         puffer: db      128             ;maximale Eingabelänge
031B                    ds      1               ;Tatsächlich benötigte Länge
031C            dma:    ds      128             ;Raum für Eingaben
039C                    ds      64              ;Platz für Maschinenstapel
03DC            oldstk: ds      2               ;Speicher: alter Stapelzeiger
                ;
                ;****** Wahlfreien Record von der Diskette lesen
                ;(Record lesen und Meldungen ausgeben)
03DE CD4804     lesen:  call    inpnr           ;Zu lesende Recordnummer eingeben
03E1 CD0B05             call    diskin          ;gewählten Record lesen
03E4 DA3901             jc      menue           ;Ende bei Fehler
03E7 11BF04             lxi     d,ergmes        ;Meldung ausgeben
03EA 0E09               mvi     c,type
03EC CD0500             call    bdos
                ;(Gelesenen Record ausgeben)
03EF 211C03             lxi     h,dma           ;Hier steht das Gelesene
03F2 0680               mvi     b,80h           ;128 Zeichen
03F4 7E         nxtcha: mov     a,m             ;Zeichen aus Puffer
03F5 23                 inx     h
03F6 E67F               ani     7Fh             ;Bit 7 ausblenden
03F8 FE7F               cpi     7Fh             ;Steuerzeichen?
03FA CA0204             jz      steuer
03FD FE20               cpi     20h             ;Steuerzeichen?
03FF D20404             jnc     keinst
0402 3E2E       steuer: mvi     a,2Eh           ;Durch Punkt ersetzen
0404 5F         keinst: mov     e,a
0405 C5                 push    b
0406 E5                 push    h
0407 0E02               mvi     c,wrcha
0409 CD0500             call    bdos            ;anzeigen
040C E1                 pop     h
040D C1                 pop     b
040E 05                 dcr     b
040F C2F403             jnz     nxtcha
0412 C33901             jmp     menue
                ;
                ;****** Wahlfreien Record auf Diskette schreiben
0415 CD4804     schrib: call    inpnr           ;Zu beschreibende Nummer eingeb
0418 E5                 push    h               ;Recordnummer speichern
0419 11D604             lxi     d,getmes
041C 0E09               mvi     c,type
041E CD0500             call    bdos
0421 111A03             lxi     d,puffer
0424 0E0A               mvi     c,rdbuf         ;Text von Tastatur einlesen
0426 CD0500             call    bdos            ;Steht jetzt im DMA-Bereich
0429 3A1B03             lda     puffer+1        ;Länge der Eingabe
```

```
042C 5F              mov     e,a
042D 1600            mvi     d,0
042F 211C03          lxi     h,dma           ;Anfang Eingabepuffer
0432 19              dad     d               ;Anfang des nicht benötigen
0433 3E80            mvi     a,80h           ;Pufferlänge
0435 93              sub     e               ;a:= noch freier Bereich
0436 CA4104          jz      clend           ;nichts mehr frei
0439 47              mov     b,a             ;Schleifenzahl
043A AF              xra     a               ;a:= Null
043B 77      clear:  mov     m,a             ;nicht benötigten Bereich
043C 23              inx     h               ;löschen
043D 05              dcr     b
043E C23B04          jnz     clear
0441 E1      clend:  pop     h               ;Recordnummer restaurieren
0442 CD0605          call    diskot          ;Record auf Diskette schreiben
0445 C33901          jmp     menue
             ;
             ;****** Dezimale Zahl von der Tastatur einlesen
0448 117C04  inpnr:  lxi     d,gibnum        ;Meldung ausgeben
044B 0E09            mvi     c,type
044D CD0500          call    bdos
0450 210000          lxi     h,0             ;Startwert
0453 E5      ziff:   push    h
0454 0E01            mvi     c,rdcha         ;Ziffer einlesen
0456 CD0500          call    bdos
0459 E1              pop     h
045A D630            sui     '0'             ;Konvertierung
045C FE0A            cpi     0Ah
045E D26E04          jnc     endzah          ;keine Ziffer -> Ende
0461 29              dad     h               ;bisherige Zahl mit 10
0462 54              mov     d,h             ;multiplizieren
0463 5D              mov     e,l             ;de -> Zahl * 2
0464 29              dad     h
0465 29              dad     h               ;hl -> Zahl * 8
0466 19              dad     d               ;hl -> Zahl * 10
0467 5F              mov     e,a
0468 1600            mvi     d,0             ;de -> letzte dezimale
046A 19              dad     d               ;zum bisherigen addieren
046B C35304          jmp     ziff            ;nächste Ziffer
046E E5      endzah: push    h
046F 117904          lxi     d,crlf          ;neue Zeile
0472 0E09            mvi     c,type
0474 CD0500          call    bdos
0477 E1              pop     h
0478 C9              ret
             ;
             ;****** Meldungen bei der Ein- und Ausgabe
0479 0D0A24  crlf:   db      13,10,'$'
047C 0D0A426974gibnum: db    13,10,'Bitte dezimale Nummer des zu '
049B 626568616E      db      'behandelnden Datensatzes eingeben: $'
04BF 47656C6573ergmes: db    'Gelesener Datensatz:',13,10,'$'
04D6 4269747465getmes: db    'Bitte geben Sie den zu schreibenden Text ein:'
0503 0D0A24          db      13,10,'$'
             ;
             ;****** Durch Register hl adressierten Record auf Diskette
0506 0E15    diskot: mvi     c,write
0508 C30D05          jmp     diskrw
```

```
                        ;
                        ;****** Durch Register hl adressierten Record lesen
050B 0E14               diskin: mvi     c,read
050D C5                 diskrw: push    b
                        ;(interne Darstellung der Extentnummer berechnen)
050E 7D                         mov     a,l             ;Recordnummer, low-Byte
050F 07                         rlc                     ;Bit 7-> Carry
0510 7C                         mov     a,h             ;Recordnummer, high-Byte
0511 17                         ral                     ;Carry -> Bit 0
0512 E61F                       ani     1Fh             ;-> EX-Byte
0514 4F                         mov     c,a             ;Zwischenspeichern
0515 7D                         mov     a,l             ;Recordnummer
0516 E67F                       ani     7Fh             ;Nummer im laufenden Extent
0518 321603                     sta     efcbcr          ;in den Dateibeschreiber
                        ;(feststellen, ob neuer Extent adressiert)
051B 3A0203                     lda     efcbex          ;alte Extentnummer
051E B9                         cmp     c               ;Vergleich mit neuer
051F CA4005                     jz      oldext
                        ;(neuen Extent anwählen)
0522 C5                 neuext: push    b
0523 11F602                     lxi     d,efcb          ;Schließen des alten Extents
0526 0E10                       mvi     c,close
0528 CD0500                     call    bdos
052B C1                         pop     b
052C 3C                         inr     a               ;'A' darf nicht 'FFh' sein
052D CA6F01                     jz      nodat           ;dürfte nicht vorkommen
0530 79                         mov     a,c
0531 320203                     sta     efcbex          ;Extentnummer
0534 11F602                     lxi     d,efcb          ;neuen Extent öffnen
0537 0E0F                       mvi     c,open
0539 CD0500                     call    bdos
053C 3C                         inr     a               ;'A' darf nicht 'FFh' sein
053D CA5305                     jz      adrerr          ;nicht vorhandener Extent adr.
                        ;(Record im gewählten Extent (alt oder neu) schreiben/ lesen)
0540 C1                 oldext: pop     b               ;enthält Code, ob lesen/schreib
0541 11F602                     lxi     d,efcb
0544 CD0500                     call    bdos
0547 A7                         ana     a               ;muß Null sein
0548 C8                         rz                      ;alles in Ordnung
                        ;
                        ;****** Fehler beim Schreiben/Lesen aufgetreten
0549 117105                     lxi     d,errmsg        ;Fehlermeldung
054C 0E09                       mvi     c,type
054E CD0500                     call    bdos
0551 37                         stc                     ;Flag: Fehler
0552 C9                         ret
                        ;
                        ;****** Fehler beim Anlegen eines neuen physischen Extents
                        ;(Meldung ausgeben)
0553 11A305             adrerr: lxi     d,exterr
0556 0E09                       mvi     c,type
0558 CD0500                     call    bdos
                        ;(Extent 0 öffnen, sonst sind alle nachfolgenden
                        ;   Dateioperationen blockiert)
055B AF                         xra     a               ;Null
055C 320203                     sta     efcbex          ;Extentbytes löschen
055F 320403                     sta     efcbs2
0562 11F602                     lxi     d,efcb
```

```
0565 0E0F               mvi     c,open          ;Extent 0 öffnen
0567 CD0500             call    bdos
056A 3C                 inr     a               ;darf nicht 'FFh' sein
056B CA6F01             jz      nodat           ;dürfte nicht vorkommen
056E C1                 pop     b               ;dieses Byte nicht benötigt
056F 37                 stc                     ;Flag für Fehler
0570 C9                 ret                     ;Zurück ins Hauptprogramm
                ;
                ;****** Fehlermeldungen
0571 2A2A2A2A2Aerrmsg:  db      '***** Fehler beim Schreiben/Lesen '
0593 4175666765         db      'Aufgetreten !',13,10,'$'
05A3 2A2A2A2A2Aexterr:  db      '***** Fehler: neuer Extent ist noch nicht '
05CD 766F726861         db      'vorhanden!',13,10,'$'
                ;
05DA                    end
```

6.5.7 Operationen zum wahlfreien Lesen/Schreiben

Im letzten Beispiel sahen Sie, daß der wahlfreie Zugriff auf bestimmte Records mit den Routinen zum sequentiellen Bearbeiten einer Datei ganz schön umständlich ist. CP/M 2.2 enthält deshalb vier BDOS-Funktionen, die speziell den Umgang mit dieser Art von Dateien erleichtern. Außerdem können mit diesen Funktionen nicht nur 512 KByte sondern 8 Megabytes direkt adressiert werden. Das entspricht den Records mit den Nummern 0 bis 65535. Diese Funktionen begnügen sich aber nicht mit dem Standard-Dateibeschreiber mit 33 Bytes, sondern benötigen einen 36 Bytes langen Dateibeschreiber. Die zusätzlichen Bytes sind für die wahlfreie Verwaltung zuständig.

Wahlfreien Record lesen

Funktion: *Kopiert einen beliebigen Datensatz aus einer Datei in den Datenpuffer.*

Zum Funktionieren benötigt diese Funktion einen Dateibeschreiber, der durch das erfolgreiche Eröffnen einer Datei mit der gewöhnlichen Eröffnungsfunktion erzeugt wurde. Hier müssen allerdings die Bytes EX und S2 vor dem Eröffnen auf Null gesetzt werden, sonst funktioniert die Recordzählung nicht richtig. Die versehentliche Verwendung eines ungültigen Dateibeschreibers hat (wie immer) verheerende Folgen.

Die Nummer des zu lesenden Records muß vor der Leseoperation in die Bytes R0 und R1 des erweiterten Dateibeschreibers übertragen werden. Das Byte R2 muß den Wert 0 besitzen. Das Eröffnen und Schließen der richtigen Extents wird von der Funktion automatisch übernommen. Weitere Vorbereitungen sind zum wahlfreien Lesen also nicht nötig. Beim Lesen bleiben die Bytes R0 und R1 unverändert. Sie können also danach entweder auf denselben Record schreiben, ihn noch einmal lesen oder durch Inkrementierung der Bytes R0 und R1 den nächsten Datensatz adressieren.

Nach dem Lesen ist der Dateibeschreiber genauso vorbereitet, als ob ein sequentieller Zugriff auf den letzten Record erfolgt wäre. Nach einem wahlfreien Zugriff kann also ohne weitere Vorbereitungen zum sequentiellen Lesen übergegangen werden. Der letzte wahlfrei gelesene Record wird dann allerdings doppelt gelesen.

Erhält man als Ergebnis der 'Read Random'-Funktion im Register A eine Null, so war das Lesen erfolgreich. Alle anderen Werte sind Fehlermeldungen. Die Werte bedeuten dabei im einzelnen:

01 Es wurde versucht, einen noch nicht geschriebener Datensatz zu lesen: Das ist ein 'Loch' in einem vorhandenen Extent.
02 (nicht verwendet)
03 Der aktuelle Record kann nicht geschlossen werden: Unter normalen Umständen dürfte diese Fehlermeldung nicht auftreten. Denkbar wäre diese höchstens, wenn die schon geöffnete Datei an einer anderen Programmstelle gelöscht wird, so daß der zu schließende Extent nicht mehr gefunden wird.
04 Ein nicht vorhandener Extent wurde adressiert: Das ist die verschärfte Form des Fehlers 01: Nicht nur der einzelne Block, in dem der gesuchte Record liegen soll, fehlt, sondern der gesamte Extent.
05 (nicht verwendet)
06 Ein Record hinter der 8 Megabyte-Marke wurde adressiert. Das ist dann der Fall, wenn das R2-Byte einen Wert ungleich Null besitzt.

Aufruf: c:= 33
de:= Adresse erweiterter Dateibeschreiber

Rückgabe: a:= 0, wenn das Lesen erfolgreich war,
1,3,4 oder 6 bei nicht erfolgreichem Lesen.

Beispiel:

```
lxi  h,10000
shld efcbr0        ;hl in den erweiterten FCB eintragen
lxi  d,efcb        ;Adresse FCB
mvi  c,21h
call 0005h
ana  a
jnz  error
```

---> Der Datenpuffer enthält den Inhalt des Records Nummer 10000 der durch den Dateibeschreiber angegebenen Datei.

Wahlfreien Record schreiben

Funktion: *Kopieren des Datenpuffers in einen beliebigen Record der Datei.*

Die Funktion zum wahlfreien Schreiben eines Records in eine Datei wirkt analog zur Funktion zum Lesen aus der Datei. Genau wie bei dieser erfolgt die Übergabe der Recordnummer in den Bytes R0 und R1 und das Byte R2 muß den Wert 0 besitzen. Nach dem Schreibbefehl ist der Dateibeschreiber ebenfalls zum sequentiellen Lesen/Schreiben vorbereitet und zeigt auf den gerade beschriebenen Record.

Wenn ein Record in einem noch nicht bestehenden Extent adressiert wird, wird dieser Extent automatisch angelegt. Sie können also ohne lange Vor-

bereitungen gleich nach dem Erzeugen der Datei in den Record mit der Nummer 65535 schreiben, ohne daß Fehlermeldungen auftreten. Im Inhaltsverzeichnis finden sich dann nicht etwa alle 512 Einträge für die dazwischenliegenden (leeren) Extents, sondern wirklich nur die beiden benötigten: Der Extent mit der Nummer 0 ist automatisch in jeder Datei enthalten. Auch auf kleinen Disketten mit wenig Raum für Inhaltsverzeichniseinträge können so mehrere Dateien mit Recordnummern bis 65535 enthalten sein. Dasselbe gilt sinngemäß für das Anlegen eines neuen Blocks.

Nach der Operation steht auch hier im Register A eine Null, wenn das Schreiben erfolgreich war. Bei einem Fehler kann das Register A die folgenden Werte enthalten:

01 (nicht verwendet)
02 (nicht verwendet)
03 Ein Extent kann nicht geschlossen werden. Wie beim wahlfreien Lesen tritt dieser Fehler unter normalen Umständen nicht auf.
04 (nicht verwendet)
05 Ein neuer Extent konnte nicht eröffnet werden, weil das Inhaltsverzeichnis der Diskette voll ist. Trotzdem können aber noch Einträge in die Datei vorgenommen werden - allerdings mit anderen Recordnummern. Schon vorhandene Extents können ja noch unbelegte Blöcke enthalten.
06 Die Diskette ist voll, so daß der neue Record nicht mehr Platz findet. Hier gilt dasselbe wie für den Fehler 5: Schon vorhandene Blöcke können durchaus noch Platz für einzelne Records bieten - allerdings nur mit anderen Recordnummern.

Aufruf: c:= 34
de:= Adresse eines erweiterten Dateibeschreibers

Rückgabe: a:= 0, wenn das Schreiben erfolgreich war,
3,5 oder 6, wenn ein Fehler auftrat.

Beispiel:

```
lxi  h,10000        ;Nummer des zu schreibenden Records
shld efcbr0         ;Bytes R0,R1 des erweiterten FCB
lxi  d,efcb         ;Anfangsadresse FCB
mvi  c,22h
call 0005h
ana  a
jnz  error          ;genauere Untersuchung möglich
```

---> In den Record Nummer 10000 der durch den Dateibeschreiber angegebenen Datei wird der Inhalt des Datenpuffers übertragen.

Wahlfreien Record schreiben mit Blocklöschung

Funktion: *Übertragen des Datenpuffers in einen wahlfreien Record einer Datei, wobei ein neuer Block gelöscht wird*

Diese Funktion entspricht fast der einfachen Funktion zum wahlfreien Schreiben. Wird ein einzelner Record in einen neu angelegten Block geschrieben, gehören die anderen Records des Blockes auch zur Datei. Unschön dabei ist, daß diese Records zufällige Daten enthalten und nicht von echten Records unterschieden werden können. Die Funktion 'Schreiben mit Blocklöschung' löscht beim Schreiben in einen neuen Block diese Pseudo-Records mit Nullen. So können sie leichter von echten Records unterschieden werden. Enthielt der Block schon vor dem Schreibbefehl andere Records, werden diese natürlich nicht gelöscht.

Nach der Operation steht auch hier im Register A eine Null, wenn das Schreiben erfolgreich war. Bei einem Fehler kann das Register A die folgenden Werte enthalten:

01 (nicht verwendet)
02 (nicht verwendet)
03 Ein Extent kann nicht geschlossen werden. Wie beim wahlfreien Lesen tritt dieser Fehler unter normalen Umständen nicht auf.
04 (nicht verwendet)
05 Ein neuer Extent konnte nicht eröffnet werden, weil das Inhaltsverzeichnis der Diskette voll ist. Trotzdem können aber noch Einträge in die Datei vorgenommen werden - allerdings mit anderen Recordnummern. Schon vorhandene Extents können ja noch unbelegte Blöcke enthalten.
06 Die Diskette ist voll, so daß der neue Record nicht mehr Platz findet. Hier gilt dasselbe wie für den Fehler 5: Schon vorhandene Blöcke können durchaus noch Platz für einzelne Records bieten - allerdings nur mit anderen Recordnummern.

Aufruf: c:= 40
de:= Adresse eines erweiterten Dateibeschreibers

Rückgabe: a:= 0, wenn das Schreiben erfolgreich war,
3,5 oder 6, wenn ein Fehler auftrat.

Beispiel:

```
lxi  h,10000      ;Nummer des zu schreibenden Records
shld efcbr0       ;Bytes R0,R1 des erweiterten FCB
lxi  d,efcb       ;Anfangsadresse FCB
mvi  c,28h
call 0005h
```

```
ana  a
jnz  error          ;genauere Untersuchung möglich
```

---> Der Inhalt des Datenpuffers wird in den Record Nr. 10000 der durch den Dateibeschreiber gegebenen Datei übertragen. Wird dabei ein neuer Block erzeugt, werden die nicht belegten Records dieses Blockes mit Null-Bytes gefüllt.

Wahlfreien Record setzen

Funktion: *Berechnet anhand der Dateibeschreiber-Information die aktuelle Recordnummer.*

Der Übergang vom wahlfreiem Lesen/Schreiben zum sequentiellen Zugriff ist sehr einfach: Bei jedem wahlfreien Zugriff wird der Dateibeschreiber automatisch für den sequentiellen Zugriff vorbereitet. Umgekehrt ist die Sache jedoch nicht so einfach: Aus der Information im Dateibeschreiber muß die Nummer für die Bytes R0 und R1 speziell berechnet werden. Genau diese Aufgabe übernimmt die Funktion 'wahlfreien Record setzen'.

Berechnet wird die Nummer des Records, der bei einem sequentiellen Zugriff als nächstes gelesen/beschrieben würde. Diese Nummer wird gleich in die Bytes R0/R1/R2 des Dateibeschreibers übertragen. Der Wert des Bytes R2 ist jedoch immer Null.

In die Berechnung gehen nur die Inhalte der Bytes EX, S2 und CR des Dateibeschreibers ein. Die Funktion liefert deshalb immer ein Ergebnis - auch bei einem nicht vollständig initialisierten Dateibeschreiber. Unsinnige Bytes EX und S2 liefern aber natürlich auch ein unsinniges Ergebnis. Die Funktion führt keinen Diskettenzugriff aus.

Aufruf: c:= 36
de:= Anfangsadresse eines erweiterten FCB

Rückgabe: nichts

Beispiel:

```
lxi  d,efcb
mvi  c,24h
call bdos
```

---> Die Bytes R0 und R1 des Dateibeschreibers enthalten jetzt die Nummer des Records, in den bei einem sequentiellem Zugriff ohne weitere Vorkehrungen als nächstes geschrieben würde.

6.5.8 Beispiel zum Umgang mit relativen Dateien

Erklärungen zu RAND2.ASM

Der Großteil des Programms RAND2.ASM ist identisch mit dem Programm RAND1.ASM. Lediglich die Routinen, die einen Record der Datei direkt ansteuern, wurden durch neue ersetzt. Beim Experimentieren mit dem Programm RAND2.ASM können Sie also genauso vorgehen, wie beim ersten Wahlfreier-Zugriff-Programm. Es ist ebenfalls nicht nur auf den Schneider-Computern sondern auf jedem beliebigen CP/M 2.2-Rechner lauffähig.

Mit den Befehlen zur sequentiellen Dateiverwaltung ist die Routine zum Aufsuchen und Schreiben/Lesen etwa 100 Zeilen lang. Mit den neuen Befehlen zum wahlfreien Zugriff schrumpft dieser Aufwand zu Null zusammen: Alle Verwaltungsaufgaben wie Öffnen und Schließen von Extents werden automatisch erledigt. Lediglich zum Aufruf und für eine einzige Fehlermeldung sind noch etwa 10 Zeilen nötig.

Dabei ist die Dateibearbeitung sogar noch komfortabler: Vor dem beschreiben zum Beispiel des Records Nummer 2000 mußten bisher alle Extents im Bereich von 0 bis 1999 von Hand mühsam erstellt werden - und das obwohl diese Extents keine sinnvollen Daten beinhalten. Dieser Aufwand entfällt jetzt ersatzlos. Die überflüssigen Extents werden unter der neuen Verwaltung gar nicht erstellt, so daß Sie zusätzlichen Platz im Inhaltsverzeichnis gewinnen. Auch auf der Diskette selbst bleibt mehr Platz: Zum erstellen eines Extents mußte nach der alten Methode mindestens ein Block Daten in den vorhergehenden Extent geschrieben werden.

Jetzt können Sie auch 65535 echte Records eingeben: Die Records mit den Nummern 4096, 8192, 16384 ... überlagern nicht mehr den Record mit der Nummer 0.

Bei weiteren Experimenten könnten Sie folgende Änderungen einfügen:

- Ersetzen Sie den Befehl zum einfachen Schreiben eines Records durch den Befehl zum Schreiben mit Blocklöschung. Das gelingt durch Auswechseln der BDOS-Funktionsnummer in einer einzigen Zeile am Ende des Programms. Beim Testen sollten Sie dann besonders auf die Pseudo-Records achten, die bei jedem Anlegen eines neuen Blocks entstehen: Das Ergebnis bei der Ausgabe sieht jetzt wesentlich besser aus. Dafür wird das Programm aber etwas langsamer.
- Beim Schreiben/Lesen wird nur unterschieden, ob die Operation erfolgreich war oder nicht und eine gemeinsame Fehlermeldung ausgegeben. Wenn Sie weitere Informationen gewinnen wollen, können Sie aber eine genauere Aufschlüsselung der 6 Fehlermeldungen vornehmen.

Das Listing zu RAND2.ASM

```
                ;Die Datei 'TEST.TXT' mit Befehlen zum relativen Lesen
                ;  wahlfrei beschreiben und lesen
                ;
                ;****** Konstanten: Nummern der CP/M-Funktionen
0005 =          bdos    equ     0005h
0001 =          rdcha   equ     01h             ;Zeichen von der Tastatur lesen
0002 =          wrcha   equ     02h             ;Zeichen an Bildschirm ausgeben
0009 =          type    equ     09h             ;Zeichenkette auf Bildschirm
000A =          rdbuf   equ     0Ah             ;Zeichenkette von der Tastatur
000F =          open    equ     0Fh             ;Datei öffnen
0010 =          close   equ     10h             ;Datei schließen
0011 =          search  equ     11h             ;Directoryeintrag suchen
0014 =          read    equ     14h             ;sequentielles Lesen
0015 =          write   equ     15h             ;sequentielles Schreiben
0016 =          make    equ     16h             ;Datei erzeugen
001A =          setdma  equ     1Ah             ;Pufferspeicherlage wählen
0021 =          randrd  equ     21h             ;wahlfreies Lesen
0022 =          randwr  equ     22h             ;wahlfreies Schreiben
0100            org     0100h
                ;
                ;****** Initialisierung: Lokalen Stapel anlegen
0100 210000             lxi     h,0
0103 39                 dad     sp              ;hl enthält alten Stapel
0104 22DC03             shld    oldstk          ;Zwischenspeichern
0107 31DC03             lxi     sp,oldstk       ;neuer Stapel
                ;
                ;****** Datei 'TEST.TXT' suchen
010A 111C03             lxi     d,dma           ;Pufferspeicher festlegen
010D 0E1A               mvi     c,setdma
010F CD0500             call    bdos
0112 11F602             lxi     d,efcb          ;erweiterter Dateibeschreiber
0115 0E11               mvi     c,search        ;ersten Eintrag suchen
0117 CD0500             call    bdos
011A 3C                 inr     a               ;Vergleich mit 'FFh'
011B C22D01             jnz     found
                ;
                ;****** Datei 'TEST.TXT' nicht gefunden
011E 11F602             lxi     d,efcb
0121 0E16               mvi     c,make          ;Datei neu erzeugen
0123 CD0500             call    bdos
0126 3C                 inr     a               ;Vergleich mit 'FFh'
0127 CA7501             jz      full            ;Directory voll
012A C33901             jmp     menue
                ;
                ;****** Datei 'TEST.TXT' schon vorhanden
012D 11F602     found:  lxi     d,efcb
0130 0E0F               mvi     c,open          ;Datei eröffnen
0132 CD0500             call    bdos
0135 3C                 inr     a               ;Vergleich mit 'FFh'
0136 CA6F01             jz      nodat           ;(dürfte nicht vorkommen)
                ;
                ;****** Datei mit neuen Daten füllen bzw. fortsetzen
0139 118201     menue:  lxi     d,help          ;Menü ausgeben
013C 0E09               mvi     c,type
013E CD0500             call    bdos
0141 0E01               mvi     c,rdcha         ;Zeichen von Tastatur abwarten
```

```
0143 CD0500             call    bdos
0146 F5                 push    psw             ;Zeichen speichern
0147 117904             lxi     d,crlf          ;neue Zeile
014A 0E09               mvi     c,type
014C CD0500             call    bdos
014F F1                 pop     psw             ;Zeichen zurück
0150 D631               sui     '1'             ;Code für Record lesen
0152 CADE03             jz      lesen
0155 3D                 dcr     a               ;'2' ist Code für Record schrei
0156 CA1504             jz      schrib
0159 3D                 dcr     a               ;'3' ist Code für Programmende
015A C23901             jnz     menue           ;falsche Eingabe -> wiederholen
                ;
                ;****** Datei schließen
015D 11F602             lxi     d,efcb
0160 0E10               mvi     c,close         ;Datei schließen
0162 CD0500             call    bdos
0165 3C                 inr     a               ;Test auf 'FFh'
0166 CA6F01             jz      nodat           ;dürfte nicht passieren
                ;
                ;****** Korrekte Programmbeendigung
0169 111A02             lxi     d,mess1         ;Meldung: Programmende
016C C37801             jmp     mess
                ;
                ;****** Fehler 1: Datei nicht gefunden
016F 115302     nodat:  lxi     d,mess2         ;Adresse 1. Botschaft
0172 C37801             jmp     mess
                ;
                ;****** Fehler 2: Diskette oder Directory voll
0175 11C702     full:   lxi     d,mess3         ;Adresse 2. Botschaft
0178 0E09       mess:   mvi     c,type          ;Zeichenkette ausgeben
017A CD0500             call    bdos
                ;
                ;****** Programm beenden
017D 2ADC03             lhld    oldstk          ;alten Stackpointer zurück
0180 F9                 sphl                    ;restaurieren
0181 C9                 ret                     ;Programmende ohne Warmstart
                ;
                ;****** Texte in der Hauptprogrammschleife
0182 0D0A0D0A46help:    db      13,10,13,10,'Folgende Aktionen stehen zur '
01A3 4175737761         db      'Auswahl: ',13,10
01AE 09313A2052         db      9,'1: Record aus der Datei lesen',13,10
01CE 09323A2052         db      9,'2: Record in die Datei schreiben',13,10
01F1 09333A2050         db      9,'3: Programmende',13,10
0203 2020202042         db      '    Bitte wählen Sie: $'
021A 416C6C6573mess1:   db      'Alles in Ordnung: Datei ''TEST.TXT'' korrekt '
0245 6765736368         db      'geschlossen',13,10,'$'
0253 2A2A2A2A2Amess2:   db      '***** Fehler: Datei nicht gefunden',13,10
0277 4163687475         db      'Achtung: dürfte eigentlich nicht auftreten, '
02A3 7761687273         db      'wahrscheinlich ein Programmfehler',13,10,'$'
02C7 2A2A2A2A2Amess3:   db      '***** Fehler: Diskette oder Verzeichnis voll'
02F3 0D0A24             db      13,10,'$'
                ;
                ;****** Im Programm verwendeter Dateibeschreiber
02F6 00         efcb:   db      0               ;Schreiben auf Bezugslaufwerk
02F7 5445535420         db      'TEST    TXT'   ;11 Zeichen Dateiname
0302 00         efcbex: db      0               ;Extent-Nr, niederwertig
0303 00                 db      0
```

```
0304 00         efcbs2: db      0               ;Extent-Nr, höherwertig
0305                    ds      17              ;beliebiger Inhalt
0316 00         efcbcr: db      0               ;1. benötigter Record im Extent
0317 000000     rand:   db      0,0,0           ;erweiterter FCB: R0, R1, R2
                ;
                ;****** Vom Programm benutzte Zwischenspeicher
031A 80         puffer: db      128             ;maximale Eingabelänge
031B                    ds      1               ;Tatsächlich benötigte Länge
031C            dma:    ds      128             ;Raum für Eingaben
039C                    ds      64              ;Platz für Maschinenstapel
03DC            oldstk: ds      2               ;Speicher: alter Stapelzeiger
                ;****** Wahlfreien Record von der Diskette lesen
                ;(Record lesen und Meldungen ausgeben)
03DE CD4804     lesen:  call    inpnr           ;Zu lesende Recordnummer eingeben
03E1 CD0B05             call    diskin          ;gewählten Record lesen
03E4 DA3901             jc      menue           ;Ende bei Fehler
03E7 11BF04             lxi     d,ergmes        ;Meldung ausgeben
03EA 0E09               mvi     c,type
03EC CD0500             call    bdos
                ;(Gelesenen Record ausgeben)
03EF 211C03             lxi     h,dma           ;Hier steht das Gelesene
03F2 0680               mvi     b,80h           ;128 Zeichen
03F4 7E         nxtcha: mov     a,m             ;Zeichen aus Puffer
03F5 23                 inx     h
03F6 E67F               ani     7Fh             ;Bit 7 ausblenden
03F8 FE7F               cpi     7Fh             ;Steuerzeichen?
03FA CA0204             jz      steuer
03FD FE20               cpi     20h             ;Steuerzeichen?
03FF D20404             jnc     keinst
0402 3E2E       steuer: mvi     a,2Eh           ;Durch Punkt ersetzen
0404 5F         keinst: mov     e,a
0405 C5                 push    b
0406 E5                 push    h
0407 0E02               mvi     c,wrcha
0409 CD0500             call    bdos            ;anzeigen
040C E1                 pop     h
040D C1                 pop     b
040E 05                 dcr     b
040F C2F403             jnz     nxtcha
0412 C33901             jmp     menue
                ;
                ;****** Wahlfreien Record auf Diskette schreiben
04   CD4804     schrib: call    inpnr           ;Zu beschreibende Nummer eingeben
0418  5                 push    h               ;Recordnummer speichern
0419 11[illegible]04    lxi     d,getmes
041C 0E09               mvi     c,type
041E CD0500             call    bdos
0421 111A03             lxi     d,puffer
0424 0E0A               mvi     c,rdbuf         ;Text von Tastatur einlesen
0426 CD0500             call    bdos            ;Steht jetzt im DMA-Bereich
0429 3A1B03             lda     puffer+1        ;Länge der Eingabe
042C 5F                 mov     e,a
042D 1600               mvi     d,0
042F 211C03             lxi     h,dma           ;Anfang Eingabepuffer
0432 19                 dad     d               ;Anfang des nicht benötigen
0433 3E80               mvi     a,80h           ;Pufferlänge
0435 93                 sub     e               ;a:= noch freier Bereich
0436 CA4104             jz      clend           ;nichts mehr frei
```

```
0439 47                 mov     b,a             ;Schleifenzahl
043A AF                 xra     a               ;a:= Null
043B 77         clear:  mov     m,a             ;nicht benötigten Bereich
043C 23                 inx     h               ;löschen
043D 05                 dcr     b
043E C23B04             jnz     clear
0441 E1         clend:  pop     h               ;Recordnummer restaurieren
0442 CD0605             call    diskot          ;Record auf Diskette schreiben
0445 C33901             jmp     menue
                ;
                ;****** Dezimale Zahl von der Tastatur einlesen
0448 117C04     inpnr:  lxi     d,gibnum        ;Meldung ausgeben
044B 0E09               mvi     c,type
044D CD0500             call    bdos
0450 210000             lxi     h,0             ;Startwert
0453 E5         ziff:   push    h
0454 0E01               mvi     c,rdcha         ;Ziffer einlesen
0456 CD0500             call    bdos
0459 E1                 pop     h
045A D630               sui     '0'             ;Konvertierung
045C FE0A               cpi     0Ah
045E D26E04             jnc     endzah          ;keine Ziffer -> Ende
0461 29                 dad     h               ;bisherige Zahl mit 10
0462 54                 mov     d,h             ;multiplizieren
0463 5D                 mov     e,l             ;de -> Zahl * 2
0464 29                 dad     h
0465 29                 dad     h               ;hl -> Zahl * 8
0466 19                 dad     d               ;hl -> Zahl * 10
0467 5F                 mov     e,a
0468 1600               mvi     d,0             ;de -> letzte dezimale
046A 19                 dad     d               ;zum bisherigen addieren
046B C35304             jmp     ziff            ;nächste Ziffer
046E E5         endzah: push    h
046F 117904             lxi     d,crlf          ;neue Zeile
0472 0E09               mvi     c,type
0474 CD0500             call    bdos
0477 E1                 pop     h
0478 C9                 ret
                ;
                ;****** Meldungen bei der Ein- und Ausgabe
0479 0D0A24     crlf:   db      13,10,'$'
047C 0D0A426974gibnum:  db      13,10,'Bitte dezimale Nummer des zu '
049B 626568616E         db      'behandelnden Datensatzes eingeben: $'
04BF 47656C6573ergmes:  db      'Gelesener Datensatz:',13,10,'$'
04D6 4269747465getmes:  db      'Bitte geben Sie den zu schreibenden Text ein:'
0503 0D0A24             db      13,10,'$'
                ;
                ;****** Durch Register hl adressierten Record auf Diskette
0506 0E22       diskot: mvi     c,randwr
0508 C30D05             jmp     diskrw
                ;
                ;****** Durch Register hl adressierten Record lesen
050B 0E21       diskin: mvi     c,randrd
050D 221703     diskrw: shld    rand            ;Recordnummer speichern
0510 11F602             lxi     d,efcb          ;  im efcb
0513 CD0500             call    bdos            ;wahlfrei lesen/schreiben
0516 A7                 ana     a               ;muß Null sein
0517 C8                 rz                      ;alles in Ordnung
```

```
                ;
                ;****** Fehler beim Schreiben/Lesen aufgetreten
0518 112205             lxi     d,errmsg        ;Fehlermeldung
051B 0E09               mvi     c,type
051D CD0500             call    bdos
0520 37                 stc                     ;Flag: Fehler
0521 C9                 ret                     ;Zurück ins Hauptprogramm
                ;
                ;****** Fehlermeldungen
0522 2A2A2A2A2Aerrmsg:  db      '***** Fehler beim Schreiben/Lesen '
0544 4175666765         db      'aufgetreten !',13,10,'$'
                ;
0554                    end
```

6.6 Fehlermeldungen des BDOS

Nicht immer kann das CP/M-Betriebssystem BDOS-Befehle so ausführen, wie Sie das wünschen. Die bisher besprochenen Fehler sind allerdings weniger ernster Natur, so daß sie nachträglich leicht korrigiert werden könnten. Die BDOS-Funktionen beenden in diesen Fällen ihre Arbeit ganz normal und übergeben dem aufrufenden Programm nur eine Mitteilung über das Mißlingen der Operation.

Andere Fehler sind jedoch so schlimm, daß Sie von einem Anwenderprogramm nicht mehr selbständig behoben werden können. In diesen Fällen wird eine Fehlermeldung auf dem Bildschirm im Klartext ausgegeben. Nachdem Sie diese Meldung durch einen Tastendruck quittiert haben, wird das Anwenderprogramm gewaltsam abgebrochen und die Systemmeldung 'A>' oder 'B>' erscheint. Die während der Programmausführung eingegebenen und noch nicht auf Diskette abgespeicherten Daten sind dann in der Regel verloren. Nur mit Tricks wie Abspeichern der gesamten TPA und anschließendes untersuchen des so abgespeicherten mit DDT können Sie Ihre Daten manchmal noch retten. Das gelingt aber nur, wenn Ihre Daten in einigermaßen geordneter Form im Speicher vorliegen.

Gute Anwenderprogramme sollten vor solchen Fehlermeldungen möglichst abgesichert sein. Das gelingt nur, wenn Sie vor jeder kritischen Operation eine Abfrage durchführen, ob die geforderten Bedingungen auch erfüllt sind. Glücklicherweise gibt es aber unter CP/M 2.2 nur 4 dieser fatalen Fehler.

Auf den Schneider-Computern erscheinen vor diesen Standard-Meldungen noch andere Fehlermeldungen mit dem Zusatz 'Retry, Cancel or Ignore' oder etwas entsprechendes. Für diese zusätzlichen Meldungen ist jedoch nicht der BDOS-Teil des Betriebssystems, sondern der hardwareabhängige BIOS-Teil verantwortlich. Wenn Ihnen diese nicht gefallen, können Sie diese auch unterdrücken. Näheres dazu steht im Handbuch Ihrer Floppydisk.

Versuch, eine schreibgeschützte Diskette zu beschreiben

Meldung: *Bdos Err On <Laufwerk>: R/O*

Signalisiert wird mit dieser Meldung nicht der durch Überkleben/Öffnen der Schreibschutzkerbe erzeugte hardwaremäßige Schreibschutz einer Diskette, sondern der softwaremäßige Schreibschutz: Nach einem verbotenen

Diskettenwechsel wird als Schutzmaßnahme automatisch die Schreibfunktion abgeschaltet. Dasselbe können Sie auch absichtlich mit der BDOS-Funktion 'Laufwerke schreibschützen' erreichen.

Um diese Fehlermeldung zu vermeiden, sollten Sie vor dem Eröffnen einer zu beschreibenden Datei mit der speziellen BDOS-Funktion testen, ob das Laufwerk schreibgeschützt ist und im Falle des Falles die Bearbeitung der Datei unterlassen. Während der Bearbeitung einer Datei ist das Abfragen des Schreibschutzstatus vor jedem Schreibbefehl nicht mehr sinnvoll, denn wenn die Diskette mit noch geöffneten Dateien gewechselt wurde, sind Ihre Daten sowieso verloren.

Durch das Drücken einer beliebigen Taste nach dieser Fehlermeldung wird das Anwenderprogramm abgebrochen und ein Warmstart ausgeführt.

Eine Datei ist schreibgeschützt

Meldung: *Bdos Err On <Laufwerk>: File R/O*

Diese Fehlermeldung tritt auf, wenn sie versucht haben, eine schreibgeschützte Datei auf einer ansonsten beschreibbaren Diskette zu ändern.

Um diese Fehlermeldung zu vermeiden, sollten Sie nach dem Eröffnen einer Datei das Byte T1 im Dateibeschreiber untersuchen. Das ist das erste Byte des Dateityps im Dateinamen. Eine schreibgeschützte Datei wird durch ein gesetztes Bit 7 signalisiert. Ist dieses Bit tatsächlich gesetzt, sollten Sie entweder die weitere Bearbeitung der Datei unterlassen oder (eventuell nach einer Warnung auf dem Bildschirm) mit der BDOS-Funktion 'Datei-Attribute setzen' den Schreibschutz entfernen.

Erscheint die Fehlermeldung erst einmal auf dem Bildschirm, wird durch jede beliebige Eingabe das Programm unterbrochen.

Ansprechen eines nicht vorhandenen Laufwerkes

Meldung: *Bdos Err On <Laufwerk>: Select*

Diese Meldung erscheint, Daten von einem nicht vorhandenen Diskettenlaufwerk gelesen werden sollten. Der beste Schutz vor dieser Meldung ist, vor jedem Eröffnen einer Datei das erste Byte im Dateibeschreiber daraufhin zu testen, ob damit wirklich ein existentes Laufwerk angesprochen wurde. Entsprechendes gilt natürlich auch für alle anderen Diskettenoperationen. Eine Überprüfung während der Bearbeitung einer Datei ist nicht sinnvoll, denn eine unvorhergesehene Veränderung eines Dateibeschreibers bedeutet ja, daß zusätzlich andere Dinge schief gelaufen sind.

Sobald die Meldung auf dem Bildschirm erschienen ist, läßt sich die Laufwerkswahl nicht mehr rückgängig machen: Jeder Tastendruck führt zu einem Warmstart. Wenn die Fehlermeldung beim Versuch auftrat, nur auf eine einzelne Datei auf einem nicht existenten Laufwerk zuzugreifen, ist danach wieder alles in Ordnung. Besonders heimtückisch ist aber, wenn die Fehlermeldung beim Wählen eines nicht existierenden Laufwerkes als Bezugslaufwerk auftritt: Nach dem Wamstart wird automatisch versucht, das Bezugslaufwerk zu aktivieren, wodurch natürlich sofort wieder eine Select-Fehlermeldung auftritt und so weiter...

Eine derart verfahrene Situation können Sie nur durch Aus- und Wiedereinschalten des Rechners beseitigen. Dieser Fehler ist praktisch die einzige Methode, den Computer unter CP/M 2.2 zum Absturz zu bringen.

Schreib-/Lesefehler auf einem existierenden Laufwerk

Meldung: *Bdos Err On <Laufwerk>: Bad Sector*

Diese Meldung tritt auf, wenn mit der Hardware eines eigentlich vorhandenen Laufwerks irgendetwas nicht in Ordnung ist. Meistens wird die Fehlermeldung beim Versuch, auf eine hardwaremäßig schreibgeschützte Diskette zu schreiben, auftreten oder wenn Sie vergessen haben, überhaupt eine Diskette einzulegen. Seltener ist ein wirklicher Lesefehler durch eine falsch formatierte oder beschädigte Diskette.

Ein programmtechnischer Schutz vor dieser Fehlermeldung ist nicht möglich. Höchstens könnte man vor dem Diskettenzugriff eine Meldung wie 'Bitte vergewissern Sie sich, daß sich eine Diskette im Laufwerk befindet' auf den Bildschirm ausgeben. Eine absolut sichere Methode ist das aber nicht.

Durch die Eingabe von ^C nach dem Erscheinen der Fehlermeldung wird das Anwenderprogramm wie bei den anderen BDOS-Fehlermeldungen abgebrochen und ein Warmstart ausgeführt.

Gelegentlich tritt ein 'echter' Lesefehler auf: Irgendein Bit im gelesenen Sektor hat sich während der Lagerzeit der Diskette geändert. In einem Ascii-Text ist das nicht besonders schlimm: Das falsche Bit äußert sich nur als Tippfehler im Text, den man verbessern kann. In CP/M 2.2 ist es deshalb möglich, den Lesefehler zu ignorieren: Drücken Sie nicht ^C, sondern ENTER, wird die Programmbearbeitung so fortgesetzt, als wäre kein Lesefehler aufgetreten.

Das können Sie ausnützen, wenn ein Programm in einem Laufwerk mit einer nicht eingelegten Diskette eine Datei eröffnen soll: Das Programm

möchte das gesamte Inhaltsverzeichnis der Diskette nach der Datei durchsuchen. Nach jedem Record erscheint dann die Meldung 'Bad Sektor'. Dadurch sollten Sie sich nicht verwirren lassen: Drücken Sie immer wieder die ENTER-Taste. Nachdem alle 16 Records des Inhaltsverzeichnis durchsucht wurden, beendet das BDOS den Diskettenzugriff von selbst und das Anwenderprogramm bringt lediglich die Meldung 'Datei nicht gefunden'. In der Regel ist dann noch eine Korrektur möglich, ohne daß Sie Ihre Daten in der TPA des Rechners verlieren.

6.7 Tips zum Umgang mit dem BDOS

Zum Entwickeln von CP/M-Programmen gibt es viele 'ungeschriebene Gesetze': Offiziell werden von der CP/M-Norm bestimmte Vorgehensweisen nicht verlangt, trotzdem haben sich diese aber fest eingebürgert und werden von fast jedem Anwendungsprogramm verwendet. Wenn Sie die wichtigsten dieser Methoden beherrschen, können Sie sich viel Arbeit ersparen.

Übergabe von Parametern an CP/M-Programme

Den schon mit dem CP/M-Betriebssystem mitgelieferten COM-Programmen können Sie Parameter übergeben, indem Sie hinter dem Programmnamen noch weitere Worte eintippen. Bei Ihren selbstprogrammierten COM-Programmen ist das ebenfalls möglich.

Der Standard-Datenpuffer

Der Console-Command-Processor wandelt die Zeichen, die hinter dem Programmnamen eingegeben wurden zunächst in Großbuchstaben um. Diese werden dann ohne weitere Veränderung im Speicherbereich ab der Adresse 0081h abgelegt. Danach folgt ein Byte mit dem Wert Null. Der Wert der zwischen dem Ende der Eingabe und der Adresse 0100h liegenden Bytes ist undefiniert. In die Adresse 0080h wird die Länge der Eingabezeichenkette geschrieben. Nach dem Aufruf eines COM-Programmes steht die Eingabezeile immer noch in diesem Speicherbereich. Das COM-Programm kann also bedenkenlos darauf zugreifen.

Wenn Sie also etwa 'test param1 param2' eingeben, wird das Programm TEST.COM gestartet. Es findet dann im Speicher folgende Werte:

```
Inhalt
14  ' ' 'P' 'A' 'R' 'A' 'M' '1' ' ' 'P' 'A' 'R' 'A' 'M' '2' 0
80h 81h 82h 83h 84h 85h 86h 87h 88h 89h 8Ah 8Bh 8Ch 8Eh 8Fh 90h
Adresse
```

Die Zahl 14 gibt die Anzahl der ab der Adresse 81h stehenden Zeichen an - wortwörtlich die Eingabe. Die danach folgenden Bytes haben undefinierte Werte. Beachten Sie, daß der Speicherbereich ab der Adresse 80h gleichzeitig der Standard-Datenpuffer für die Disketten-Ein-/Ausgabe ist. Wenn Sie die Eingabe noch verwenden wollen, muß vor dem ersten Diskettenzugriff also entweder die Eingabe in einen sicheren Bereich kopiert oder der Datenpuffer in einen anderen Bereich gelegt werden. Da die letzten 2 Bytes

des Datenpuffers zur internen Verarbeitung der Zeile benötigt werden darf die Eingabe die Länge von 125 Zeichen nicht überschreiten.

Der Standard-Dateibeschreiber

Der Speicherbereich von der Adresse 005Ch bis zur Adresse 007F ist für einen beliebigen Dateibeschreiber reserviert. Das korrekte Initialisieren eines Dateibeschreibers ist jedoch gar nicht so einfach. Besondere Schwierigkeiten bereitet die automatische Umsetzung eines Klarschrift-Dateinamens inklusive Laufwerksangabe in die interne Darstellung.

Der Console-Command-Prozessor betrachtet deshalb die ersten beiden hinter dem Programmnamen stehenden Worte als Dateinamen und nimmt Ihnen diese Arbeit gleich ab. Das Ergebnis dieser Umsetzung wird dabei zusätzlich zur Klarschrift-Ablage der Eingabe im Datenpuffer berechnet.

Das erste Wort hinter dem Programmnamen wird so umgesetzt, daß ohne weitere Vorbereitungen der BDOS-Dateiöffnungsbefehl mit dem Speicherbereich ab der Adresse 005Ch als Dateibeschreiber durchgeführt werden kann. Nach dem Start eines COM-Programmes enthalten die Adressen 005Ch also den Laufwerkscode und die Adressen 005Dh bis 0067h den Dateinamen in Großschrift. Dabei liegen Name und Typ an den richtigen Stellen und nicht benötigte Stellen werden mit Leerzeichen gefüllt. Die Bytes 0068h bis 006Bh und die Adresse 007Ch enthalten Null-Bytes.

Die Bytes 006Ch bis 007Bh sind für die Blockbelegungstabelle der Datei reserviert und werden erst nach der Eröffnung der Datei benötigt. Unmittelbar nach dem Programmstart enthalten diese 16 Bytes deshalb die codierte Form des zweiten Wortes der Eingabe. Der Aufbau dieses Bereiches entspricht ganz genau dem des Bereiches 005Ch bis 006Bh. Wenn Sie einen zweiten Dateinamen benötigen, sollten Sie vor der Eröffnung der ersten Datei diese Bytes in einen sicheren Bereich kopieren. Sinnvollerweise kopiert man diesen gleich an die endgültige Position eines 2. Dateibeschreibers. Die zweite Datei kann dann ebenfalls ohne weitere Vorbereitungen eröffnet werden. Für einen Rename-Aufruf ist der Standard-FCB schon ohne die Kopieroperation vorbereitet.

In die beiden Dateinamen gehen nur die Zeichen bis zum ersten verbotenen Zeichen ein. Ist schon im ersten Namen ein verbotenes Zeichen enthalten, besteht der zweite Name nur aus Leerzeichen. Da auch mehrdeutige Dateinamen erlaubt sind, sind Fragezeichen nicht verboten. Sterne ('*') werden in eine Folge von Fragezeichen umgesetzt. Besteht ein Name oder/und der Namensteil nach dem Punkt aus mehr als den erlaubten Zeichen, so werden nur die ersten 8 beziehungsweise 3 beachtet. Befinden sich vor oder zwischen den Dateinamen mehrere Leerzeichen, so werden diese übersprungen.

Mit der Eingabe 'test b:0123456789.abcde probe;.dat xyz' erzielen Sie in den ersten 32 Bytes des Standard-Dateibeschreibers die folgenden Werte:

```
Inhalt
01h '0' '1' '2' '3' '4' '5' '6' '7' 'A' 'B' 'C' 00h 00h 00h 00h
5Ch 5Dh 5Eh 5Fh 60h 61h 62h 63h 64h 65h 66h 67h 68h 69h 6Ah 6Bh
Adresse

Inhalt
00h 'P' 'R' 'O' 'B' 'B' 'E' ' ' ' ' ' ' ' ' ' ' ' ' ' ' 00h 00h 00h 00h
6Ch 6Dh 6Eh 6Fh 70h 71h 72h 73h 74h 75h 76h 77h 78h 79h 7Ah 7Bh
Adresse
```

Die genauen Zusammenhänge können Sie mit Hilfe des Programms in Bild 6-5 erforschen. Dieses Programm macht nichts anderes, als den Inhalt des Standard-Dateibeschreibers und des Datenpuffers auf den Bildschirm auszugeben. Wegen der geringen Länge lohnt es sich kaum, dafür extra den Assembler zu verwenden. Am besten geben Sie es mit Hilfe des DDT.COM ein und speichern es dann mit SAVE 1 PARAM.COM ab.

```
0100  LXI  H,005C      0112  JNC  0117
0103  MVI  B,A4        0115  MVI  A,2E
0105  MOV  A,M         0117  MOV  E,A
0106  INX  H           0118  MVI  C,06
0107  PUSH B           011A  CALL 0005
0108  PUSH H           011D  POP  H
0109  ANI  7F          011E  POP  B
010B  CPI  7F          011F  DCR  B
010D  JZ   0115        0120  JNZ  0105
0110  CPI  20          0123  RET
```

Bild 6-5: *Übergabeparameter auf Bildschirm ausgeben*

Test auf gültige Dateinamen

Beim Erzeugen einer neuen Datei ist es sehr unangenehm, wenn dazu ungültige Dateinamen verwendet werden. Sie sollten also im Normalfall unbedingt eine Prüfung des Namens vornehmen.

Wenn der Dateiname vom CCP erstellt wurde, haben Sie es am einfachsten: Das einzige ungültige Zeichen, das enthalten sein kann, ist das Fragezeichen. Mit einer Schleife sollten Sie den Bereich von 005Dh bis 0067h daraufhin überprüfen. Sie sollten auch sicherstellen, daß überhaupt ein Dateiname eingegeben wurde. Dazu genügt es, wenn das Byte an der Adresse 005Dh oder das an der Adresse 0065h einen Wert ungleich 32 hat.

Wenn der Dateiname nicht vom CCP codiert wurde, sondern später im Programm durch eine Abfrage eingegeben wurde, ist die Sache schon schwieriger. Zunächst sollten Sie sicherstellen, daß keinerlei verbotene Zeichen im Namen enthalten sind. Leerzeichen dürfen zwar enthalten sein, aber nicht

irgendwo in der Mitte des Namens, sondern nur am Ende. Den Punkt zwischen Namen und Typ müssen Sie durch eine gewisse Anzahl von Leerzeichen ersetzen. Gegebenenfalls müssen auch überflüssige Zeichen im Namen entfernt werden, um die Grenze zwischen Namen und Typ an die richtige Stelle zu bringen. Wichtig ist auch, daß die Trennung zwischen einer eventuellen Laufwerksangabe und dem eigentlichen Namen richtig erfolgt.

Diese Prüfungen müssen Sie aber wirklich nur beim Neuerzeugen einer Datei durchführen. Da ein ungültiger Dateiname nicht auf der Diskette enthalten ist, ergibt sich beim bloßen Eröffnen schon aus diesem Grund eine Fehlermeldung - und das ist ja gerade der Sinn der Abfrage.

Anlegen eines lokalen Stapels

Nach dem Aufruf eines COM-Programmes befindet sich der Maschinenstapel irgendwo im CCP-Bereich des Computers. Dort ist jedoch nur Platz für 7 Einträge vorhanden. Für komplexe Anwendungsprogramme ist das in der Regel nicht ausreichend. Als erstes sollten Sie in Ihren Programmen also den Stapelzeiger auf einen genügend großen freien Bereich in der TPA des Computers richten.

Das BDOS legt bei jedem Einsprung ebenfalls einen lokalen Stapelzeiger an, der erst unmittelbar vor dem Rücksprung ins Anwenderprogramm auf den alten Wert zurückgesetzt wird. Ein BDOS-Aufruf belegt im Stapel also wirklich nur den Platz für den Befehl 'call 0005' und Sie brauchen sich nicht darum zu kümmern, wieviel Platz Sie im Stapel für die einzelnen BDOS-Funktionen reservieren müssen. Aus diesem Grund dürfen Sie, auch wenn sich der Stapel noch im CCP-Bereich befindet, eine BDOS-Funktion ausführen. Zu empfehlen ist diese Methode aber nicht.

Im allgemeinen wird ein Programm durch einen Warmstart beendet, damit das Betriebssystem vollständig restauriert wird. Wenn Sie sicher sind, daß Ihr Programm weder den CCP noch irgendein anderes Teil des Systems zerstört hat, gibt es eine weitere Möglichkeit:

Der sofort nach dem Aufruf aktive Maschinenstapel bietet eigentlich Raum für 8 Einträge. Der erste Eintrag ist jedoch von der Rücksprungadresse in den CCP belegt. Sie können dann Ihr Anwenderprogramm wie ein gewöhnliches Unterprogramm durch einen RET-Befehl beenden. Danach erscheint wie gewöhnlich die Bereitschaftsmeldung 'A>' oder 'B>' des Betriebssystems - aber ohne den lästigen Warmstart.

Diese Methode können Sie sogar anwenden, wenn Sie in Ihrem Programm einen lokalen Stapel verwenden: Vor dem Rücksprung müssen Sie eben den alten Stapel restaurieren. Das gelingt mit einer Befehlsfogle wie in Bild 6-6:

```
lxi  h,0000     ;hl löschen
dad  sp         ;Stapel zum Register hl addieren
shld oldstck    ;hl enthält jetzt den alten Stapelzeiger.
                ;Dieser wird an der Adresse 'oldstck' zwischengespeichert.
lxi  sp,newstck;neuen Stapelzeiger setzen
     .
     .    (hier steht ein Anwenderprogramm)
     .
lhld oldstck    ;alten Stapelzeiger ins hl-Register
sphl            ;hl in das Stackpointer-Register übertragen
ret             ;dieser Rücksprung beendet das Programm
```

Bild 6-6: *Start und Ende eines CP/M-Programms*

Der Console-Command-Processor liegt ungeschützt in der TPA und kann von CP/M-Programmen ohne weiteres überschrieben werden. Sie sollten sich also genau überlegen, ob Sie das Einsparen des Warmstarts riskieren können.

Unerwünschten Programmabbruch vermeiden

Nicht immer können BDOS-Fehlermeldungen, die zum Programmabbruch führen, vermieden werden. Durch einen Trick können Sie ein unerwünschtes Programmende aber vermeiden:

Zum Ausführen des Warmstarts springt das BDOS nach der Meldungsausgabe an die Adresse 0000h. Dort befindet sich gewöhnlich ein Sprungbefehl zur eigentlichen Warmstartroutine. Durch eine Änderung diese Sprungbefehls können Sie die Warmstartroutine durch eine eigene ersetzen. Das könnte zum Beispiel eine Fortsetzung des abgebrochenen Programms sein.

Bei einem eigenen Warmstart-Ersatz sollten Sie folgendes beachten:

- Nach dem Sprung zur Adresse 0 zeigt der Stackpointer auf einen unbestimmten Wert. Vor dem Beginn der eigentlichen Fehlerbehandlungsroutine sollte dieser unbedingt wieder auf einen vernünftigen Wert gesetzt werden.
- Die Warmstart-Fehlerroutine wird nur nach einem Diskettenfehler angesprungen. Vor der Fortsetzung des Anwenderprogramms sollte zumindest ein Disk-Reset durchgeführt werden, damit der Fehler nicht wiederholt wird.

Das Anwendungsprogramm echt beenden können Sie nach dieser Methode nur noch, wenn Sie den Sprungbefehl an der Adresse 0 restaurieren oder einen Warmstart mit Hilfe der BDOS-Funktion Nr. 0 auslösen.

Speicherplatzabfrage

An der Adresse 0005h steht ein Sprung zur ersten Adresse des BDOS. Diese Zieladresse ist gleichzeitig das allererste vom CP/M-Betriebssystem benötigte Byte. Die Speicherzellen 0006h/0007h enthalten die reine Adresse des Sprungzieles und können deshalb als Zeiger auf das Ende der TPA betrachtet werden.

Die Abfrage des Zeigers kann von speicherintensiven CP/M-Programmen verwendet werden, die mit wachsendem Speicherplatz immer besser arbeiten. Das ist zum Beispiel bei Kopierprogrammen der Fall.

Diesen Zeiger können Sie absichtlich verändern: Wenn Sie die beiden Bytes an den Adressen 0006h/0007h auf einen niedrigeren Wert setzen, glauben Anwenderprogramme, daß das BDOS an dieser Adresse beginnt und lassen den darüberliegenden Speicherbereich unverändert. Auf diese Weise kann Speicherplatz für besondere Anwendungen reserviert werden. Die BDOS-Routinen müssen dann aber wirklich an der neuen Adresse beginnen. Am einfachsten erreichen Sie das durch einen Sprungbefehl an der neuen Obergrenze, der zur alten Anfangsadresse weist. So wird gewissermaßen nur eine Umleitung gelegt.

An dieser Stelle können aber auch die Aufrufparameter untersucht werden und nur bei bestimmten BDOS-Funktionen das normale BDOS angesprungen werden. Andere Funktionen können durch neue ersetzt werden. Nach dieser Methode arbeiten zum Beispiel DDT.COM und XSUB.COM. Beim Aufruf kopiert sich das XSUB-Programm selbst ans Ende der TPA und verändert den TPA-Zeiger, so daß es von einem Anwenderprogramm nicht mehr überschrieben werden kann. Dort macht es nichts anderes, als die BDOS-Funktion mit der Nummer 10 'Zeichenkette von der Tastatur einlesen' durch eine Funktion zu ersetzen, die die Zeichenkette vom Diskettenlaufwerk einliest.

Dieser Schutz funktioniert ohne weitere Maßnahmen nur bis zum nächsten Warmstart. Dann werden alle Zeiger restauriert. Soll eine Routine in einem geschützten Speicherbereich länger aktiv bleiben, muß es also auch die CP/M-Warmstartfunktion ersetzen. Dabei muß sowohl der Einsprung über die BDOS-Funktion mit der Nummer 0 als auch der Einsprung über den Vektor an der Speicherstelle 0 berücksichtigt werden. Ab diesem Zeitpunkt wird allerdings das Betriebssystem nicht mehr restauriert.

Eine Routine in einem geschützten Speicherbereich, die über den nächsten Warmstart hinaus aktiv ist, muß also auch den Console-Command-Processor enthalten. Es können so nicht mehrere derartige Routinen gleichzeitig aktiv sein - denn nur eine kann den CCP ja wirklich enthalten.

Minimierung der Diskettenzugriffe

Durch eine geschickte Reihenfolge beim Lesen und Schreiben mehrerer Records können Sie in der Regel einige Diskettenzugriffe einsparen. Das führt zu einer beachtlichen Beschleunigung von Programmen. Wenn Sie etwa 4 aufeinanderfolgende Records von einer Datei in die andere kopieren wollen, könnten Sie so vorgehen:

Sie lesen einen Record aus der einen Datei und schreiben ihn sofort in die andere Datei bevor Sie einen weiteren Record lesen. Wenn Ihre Disketten in Sektoren zu 512 Bytes organisiert sind, sind die folgenden Schreib-/Lesevorgänge notwendig: Zunächst wird ein Sektor von der Diskette gelesen und 128 Bytes daraus in den Datenpuffer kopiert. Danach wird der Zielsektor angesteuert. Da auf diesen nur in 512-Byte-Sektoren geschrieben werden kann, die anderen 384 Bytes aber nicht verlorengehen dürfen, wird zunächst der Zielsektor in den Sektorpuffer gelesen. Jetzt werden 128 Bytes des Zielsektors durch den Inhalt des Datenpuffers ersetzt. Als nächstes soll wieder der Quellsektor gelesen werden. Damit die neuen Daten nicht verlorengehen, muß der Zielsektor sofort auf die Diskette zurückgeschrieben werden. Jetzt wird wieder die Quellspur angesteuert und ein neuer Record übertragen. Zum Kopieren von 4 Records sind nach dieser Methode also 12 Diskettenzugriffe und 8 Spurwechsel von der Quell- auf die Zielspur nötig.

Eine zweite Möglichkeit ist, jeweils die 4 Records eines Sektors unmittelbar hintereinander zu lesen und danach erst gebündelt in den Zielsektor zu schreiben. Bei der Anforderung des ersten Records wird der Quellsektor in den Sektorpuffer geladen und der gesuchte Record daraus gelesen. Die anderen 3 Records befinden sich danach schon im Sektorpuffer, so daß für die weiteren Leseoperationen kein Diskettenzugriff mehr notwendig ist. Beim Schreiben der Records in den Zielsektor ist es ähnlich: Das zurückschreiben der Daten erfolgt erst, wenn ein neuer Sektor angewählt wird und die Daten im Puffer verlorengehen würden. So werden alle 4 Records zuerst in den Puffer geschrieben, ohne daß ein Diskettenzugriff erfolgt. Erst bei der Anwahl eines neuen Sektors erfolgt dieser. Zum Kopieren der 4 Records sind nach dieser Methode nur 2 Spurwechsel und 2 Diskettenzugriffe nötig. Sie erhalten also gegenüber der ersten Methode eine Beschleunigung um den Faktor 4.

Die 2 Spurwechsel könnten Sie auch noch sparen, wenn Sie jeweils eine ganze Anzahl eventuell benachbarter Spuren der einen Datei lesen, bevor Sie einen anderen Diskettenbereich zum Ablegen in der anderen Datei ansteuern. So wäre noch einmal eine Beschleunigung möglich. Als Faustregel können Sie sich merken: Je mehr Pufferspeicher Sie verwenden, desto schneller ist das Kopierprogramm.

Wenn Sie mit langen Dateien zum wahlfreien Zugriff arbeiten, werden Sie häufig kurz hintereinander Records aus verschiedenen Extents benötigen. Die gewöhnlichen Schreib-/Lesefunktionen steuern in diesem Fall bei fast jedem Record das Inhaltsverzeichnis der Datei an, so daß derselbe Effekt auftritt, wie im ersten (negativen) Beispiel.

Ein Ausweg ist möglich, indem Sie ein und dieselbe Datei mit verschiedenen Dateibeschreibern gleichzeitig eröffnet halten. Jedem Extent der Datei wird ein eigener Dateibeschreiber zugeordnet. Vor dem eigentlichen Schreib-/Lesevorgang sollten Sie untersuchen, in welchem Extent der gesuchte Record liegt. Durch die FCB-Adresse sollte der Schreib-/Lesefunktion dann der Dateibeschreiber übergeben werden, der schon auf den gesuchten Extent zeigt. In diesem Beschreiber befinden sich schon die zum Zugriff nötigen Daten, so daß ein spezielles Lesen des Inhaltsverzeichnisses unterbleiben kann. Die angeforderten Records können direkt angesteuert werden.

7 Die Zeropage

Die Nullseite des Speichers (Bereich 0000h bis 00FFh) beinhaltet verschiedene Informationen über den Zustand des CP/M-Betriebssystems, die von Anwenderprogrammen verwendet werden können. Der genaue Aufbau dieses Bereiches ist in der folgenden Tabelle beschrieben:

Adresse	Inhalt/Bedeutung
0000h-0002h	Sprungbefehl zur Warmstartroutine des Betriebssystems.
0003h	Beinhaltet das Input/Output-Byte, das die Zuordnung der virtuellen Ausgabekanäle zu den realen Geräten regelt. Das ist genau das I/O-Byte, das auch mit den BDOS-Funktionen 7 und 8 verändert werden kann.
0004h	Dieses Byte enthält in den Bits 0 bis 3 die Nummer des Bezugslaufwerkes und in den Bits 4 bis 7 die aktuelle Usernummer. Nach jedem Warmstart wird die Voreinstellung anhand dieses Wertes vorgenommen. Die BDOS-Funktionen zum Einstellen der Benutzernummer und des Bezugslaufwerkes verändern dieses Byte jedoch nicht. Wenn also eine in einem Anwenderprogramm getroffene Neueinstellung auch nach einem Warmstart erhalten bleiben soll, muß dieses Byte 'von Hand' geändert werden.
0005h-0007h	Über die Adresse 0005h wird das BDOS angesprungen. Diese 3 Bytes enthalten deshalb einen Sprungbefehl zur eigentlichen BDOS-Routine. Da diese Adresse die allererste vom gesamten CP/M-System belegte Speicherstelle ist, stellen die Bytes 0006h/0007h gleichzeitig einen Zeiger auf die Grenze der TPA dar.
0008h-002Fh	In Standard-CP/M-Systemen wird dieser Speicherplatz nicht genutzt. In den CPC-Computern sind in diesen Bytes jedoch die Routinen zum Bankswitching enthalten. In Anwenderprogrammen sollten diese Speicherstellen deshalb nicht verwendet werden.
0030h-0037h	Das ist der Einsprungbereich für den RST 6-Befehl. In einem Standard-CP/M-System ist dieser Bereich nicht genutzt, aber bereits für eine spätere Verwendung re-

serviert. In den CPC-Computern übernimmt dieser Bereich jedoch die Funktionen des RST7-Bereiches, da in diesen der RST7 für andere Zwecke benötigt wird.

0038h-003Ah In einem Standard-CP/M-System wird dieser RST 7-Bereich nur vom DDT-Programm benötigt: Die Adresse 0038h wird bei einem Breakpoint in einem zu testenden Programm angesprungen. In den CPC-Computern ist dieser Bereich jedoch vom Einsprung in die Interrupt-Routine belegt. Das DDT-Programm wurde deshalb von Schneider extra so angepaßt, daß die Breakpoint-Bearbeitung über die Adresse 0030h stattfindet.

003Bh-003Fh Dieser Bereich wird normalerweise weder von einem Standard-CP/M-System noch von einem CPC-Computer verwendet, ist aber für eine spätere Verwendung reserviert. In einem CPC-Computer wäre es aber möglich, die Behandlung eines externen Interrupt-Signals über diese Adressen abzuwickeln.

0040h-004Fh Anwenderprogrammen ist die Verwendung dieser 16 Bytes verboten. Das dem Computer angepaßte BIOS darf jedoch nötigenfalls diesem Bereich benutzen. Denkbar wäre zum Beispiel eine weitere Bankswitching-Logik.

0050h-005Bh Weder unter CP/M 2.2 noch in den CPC-Computern werden diese Bytes verwendet. Sie sind jedoch für eine eventuelle Verwendung durch spätere Betriebssystemversionen reserviert.

005Ch-007Ch Das ist der Standard-Dateibeschreiber, der beim Start eines Programms vom CCP initialisiert und dem Anwenderprogramm übergeben wird.

007Dh-007Fh Raum für den erweiterten Dateibeschreiber: Werden mit dem Standard-Dateibeschreiber wahlfreie Dateizugriffe durchgeführt, werden diese Bytes zur Ablage der Recordnummer benötigt.

0080h-00FFh Das ist der Standard-Datenpuffer, in dem ohne weitere Vorkehrungen die Ergebnisse aus Diskettenzugriffen abgelegt werden. Nach dem Programmstart beinhaltet dieser Bereich zusätzlich den Text der Eingabezeile, mit der das Programm gestartet wurde.

Hinweise zu den RST-Befehlen

Beim Selbsterstellen von CP/M-Programmen werden Sie wahrscheinlich nie die RST-Befehle mit den Nummern 1 bis 5 verwenden. Trotzdem sind diese aber nicht umsonst vorhanden:

Alle Routinen, die etwas mit der normalen Ein-/Ausgabe in den CPC-Computern zu tun haben, verwenden diese RST-Befehle, ohne daß Sie es bemerken. Davon betroffen sind Bildschirm-, Druckerausgaben und Tastatureingaben. Sogar die 300 mal in der Sekunde auftretende Interrupt-Routine benötigt diese Restart-Befehle. Diese Restarts sollten Sie also niemals verändern, wenn Sie einen Absturz des Computers vermeiden wollen.

Unter CP/M 2.2 steht nur der RST6-Befehl zu Ihrer freien Verfügung: Die Bytes in den Adressen 0030h bis 0037h dürfen Sie tatsächlich beliebig verändern. In diesem Fall haben Sie aber keine Garantie, daß das Programm auch auf anderen CP/M-Versionen lauffähig ist. Als normaler Homecomputer-Anwender dürfte Ihnen das aber gleichgültig sein.

Schlimmer ist aber, daß Sie in diesem Fall das Programm nicht mit dem DDT.COM im Single-Step-Betrieb testen können. Sie können dann das neuentwickelte Programm nur noch starten und zusehen, ob's läuft. Sie sollten es sich genau überlegen, ob die Vorteile eines selbstdefinierten Restart-Befehls in Ihrem Fall diese Einschränkung aufwiegen.

Die BDOS- und Warmstarteinsprünge

Durch das Verändern des Sprungbefehles an der Speicherstelle 0000h, so daß er auf ein eigenes Programm zeigt, können Sie die CP/M-Warmstartroutine durch eine eigene ersetzen. Dadurch ist es zum Beispiel möglich, einen Programmabbruch durch einen BDOS-Error zu verhindern. In diesem Fall kann ein Anwenderprogramm nur noch durch die BDOS-Funktion mit der Nummer 0 abgebrochen werden, da alle anderen Warmstartauslöser den Sprung zur Adresse 0 beinhalten.

In Ihrer neuen Warmstartroutine sollten Sie äußerste Vorsicht walten lassen: In der Regel zeigt der Stackpointer nach dem Ansprung der Routine auf einen unsinnigen Wert. Vor dem Weiterarbeiten müssen Sie ihn also unbedingt restaurieren. Außerdem sollten Sie in der Routine das Diskettensystem zurücksetzen.

Nicht ratsam ist es auch, in der Warmstartroutine noch geöffnete Dateien zu schließen oder irgendwelche Daten auf die Diskette zu schreiben. Der Warmstart zeigt ja gerade, daß damit etwas nicht in Ordnung war. Ver-

suchen Sie es trotzdem, können wichtige Daten auf Ihrer Diskette zerstört werden.

Durch Herabsetzen des Wertes in den Speicherzellen 0006h/0007h können sie die Obergrenze der TPA verschieben und so Speicherplatz vor dem Überschreiben durch andere Programme schützen. Weil diese Bytes gleichzeitig die Zieladresse eines Sprungbefehles darstellen, wird aber als Nebeneffekt bei jedem Aufruf einer BDOS-Routine an den Anfang dieses Bereiches gesprungen. Dort muß also ein weiterer Sprungbefehl zur alten BDOS-Routine stehen. Selbstverständlich können aber an dieser Stelle die Übergabeparameter des BDOS-Aufrufs untersucht werden und so bestimmte BDOS-Funktionen durch eigene ersetzt werden.

Bei jedem Warmstart wird der Wert an der Adresse 0006h aber wieder rückgängig gemacht: Die Warmstartroutine wurde ja gerade geschaffen, um auch eventuell fehlerhafte Änderungen des Betriebssystems zu korrigieren. Wenn der geschützte Speicherbereich längere Zeit erhalten bleiben soll, muß auch die Warmstartroutine verändert werden. In diesem Fall wird das Betriebssystem aber nicht mehr restauriert, so daß dann in der eingehängten Routine auch der vollständige CCP enthalten sein muß.

Das I/O-Byte und das Bezugslaufwerk

Diese Bytes der Zeropage werden in der Regel durch bestimmte BDOS-Funktionen verändert: Damit ist garantiert, daß die Änderungen zu allen CP/M-Versionen kompatibel sind. Zumindest unter CP/M können Sie diese Bytes aber auch direkt verändern:

Der Wert im Input/Output-Byte ist genau derselbe, den Sie auch der entsprechenden BDOS-Routine übergeben müssen oder erhalten. Die Befehlsfolge

```
mvi  a,81h!mvi  e,08h!call 0005h
```

ist also vollständig kompatibel zur Befehlsfolge

```
mvi a,81h!sta 0003h
```

Die Änderung des Bezugslaufwerkes und der Usernummer über die Speicherstelle 0004h ist nicht ganz kompatibel zur Änderung über die BDOS-Funktionen: Nach einem Warmstart wird die Änderung, die Sie mit Hilfe der BDOS-Funktionen durchgeführt haben, rückgängig gemacht. Nach dem Programmende befinden Sie sich in diesem Fall in derselben Usernummer und dasselbe Bezugslaufwerk ist aktiviert, wie vor dem Programmstart.

Wenn Sie die Änderung direkt in die Speicherstelle 0004h eintragen, bleibt diese auch nach dem Warmstart erhalten. So können Sie also in einem Anwenderprogramm das Bezugslaufwerk dauerhaft umstellen. Das birgt aber eine zusätzliche Gefahr: Wenn Sie versehentlich ein nicht existentes Laufwerk selektiert haben, erscheint zunächst die Fehlermeldung 'select'. Im einen Fall wird danach das alte Bezugslaufwerk restauriert und alles ist in Ordnung. Im anderen Fall bleibt das falsche Laufwerk jedoch weiter adressiert, so daß endlos 'Select'-Fehlermeldungen erscheinen.

8 Das Herz des Systems: BIOS

8.1 Allgemeines zum BIOS

Die Norm des BIOS

Im BIOS Ihres Computer, dem 'Basic Input/Output System', sind alle hardwareabhängigen Teile des Betriebssystems zusammengefaßt. Trotzdem gibt es aber noch einen genormten Aufruf dieser Teile: Die hardwareunabhängigen Teile des Betriebsystems können auf jedes beliebige der Norm entsprechende BIOS zugreifen, ohne sich darum kümmern zu müssen, wie diese hardwareabhängigen Teile im Detail ausgeführt werden. Die Entwickler des CP/M-Betriebssystems brachten das Kunststück fertig, das Funkionieren des Computers mit nur 17 hardwareabhängigen Routinen zu sichern. Dafür hat aber das BIOS keine Ahnung von Dateien, sondern kann nur Dinge wie auf einen mit Spur und Sektor angegebenen Abschnitt der Diskette schreiben (egal ob dieser in einer Datei liegt oder nicht) oder ein einzelnes Zeichen auf dem Bildschirm ausgeben. Das hat aber den Vorteil, daß ein für seinen Computer passendes BIOS vom Hersteller ganz einfach entworfen werden kann.

Wenn Sie nur Standard-CP/M-Programme entwickeln möchten, dann brauchen Sie über den Aufbau des BIOS nicht Bescheid zu wissen. alle bisher erhaltenen Informationen genügen Ihnen dann. Dieses Kapitel ist also für Sie nur wichtig, wenn Sie gerne mit Ihrem Computer 'basteln' und es mit der Kompatibilität Ihrer Programme und Disketten zu denen anderer Computerbesitzer nicht so ernst nehmen.

Wenn Sie lieber beim Standard bleiben wollen, dann hat dieses Kapitel im wesentlichen rein informellen Charakter. Nur einige wenige Tricks können Sie dann praktisch anwenden.

Die Aufteilung des BIOS

Unmittelbar hinter dem Ende des BDOS, also an der Adresse FBASE + 0DFAh beginnt eine Tabelle, die 17 Sprungbefehle beinhaltet. Diese 17 Sprungbefehle führen zu den verschiedenen BIOS-Unterprogrammen. Das BDOS kennt natürlich seine Endadresse und kann deshalb die Sprungtabelle ohne Probleme finden und zu einem bestimmten Unterprogramm springen. Die absolute Lage des BDOS mitsamt dem BIOS ist jedoch nicht festgelegt: CP/M kann an jeden Speicher ab einer Mindestgröße von 20 KByte opti-

mal angepaßt werden. Bild 8-1 zeigt den Aufbau dieser Sprungtabelle im Detail. Die Abkürzung BIBASE steht in dieser Tabelle für die Anfangsadresse der Sprungtabelle beziehungsweise für das erste Byte nach dem Ende des BDOS-Teiles. Da das BDOS und andere Teile des BIOS unbedingt im RAM liegen müssen, befindet sich diese Sprungtabelle in der Regel ebenfalls in einem RAM.

```
BIBASE+00h:    jmp  BOOT      ;vollständiger Reset
BIBASE+03H:    jmp  WBOOT     ;Warmstart
BIBASE+06h:    jmp  CONST     ;Test, ob Taste gedrückt ist
BIBASE+09h:    jmp  CONIN     ;Zeichen von Tastatur holen
BIBASE+0Ch:    jmp  CONOUT    ;Zeichen an Konsole ausgeben
BIBASE+0Fh:    jmp  LIST      ;Zeichen an Drucker ausgeben
BIBASE+12h:    jmp  PUNCH     ;Zeichen an Puncher ausgeben
BIBASE+15h:    jmp  READER    ;Zeichen vom Reader einlesen
BIBASE+18h:    jmp  HOME      ;Laufwerk auf Spur 0 stellen
BIBASE+1Bh:    jmp  SELDSK    ;Laufwerk auswählen
BIBASE+1Eh:    jmp  SETTRK    ;Spur der Diskette auswählen
BIBASE+21h:    jmp  SETSEC    ;Sektornummer wählen
BIBASE+24h:    jmp  SETDMA    ;Datenpufferadresse wählen
BIBASE+27h:    jmp  READ      ;gewählten Record lesen
BIBASE+2Ah:    jmp  WRITE     ;gewählten Record schreiben
BIBASE+2Dh:    jmp  LISTST    ;Druckerstatus feststellen
BIBASE+30h:    jmp  SECTRAN   ;Sektornummer umrechnen
```

Bild 8-1: *Die BIOS-Sprungtabelle*

Der zweite große Abschnitt des BIOS ist ein RAM-Bereich, in dem verschiedene Daten zwischengespeichert werden können. Es gibt dabei sowohl Speicherstellen, die vom BIOS selbst benötigt werden, als auch Speicherstellen, die für eine Verwendung durch das BDOS reserviert sind. Üblicherweise hat dieser Speicher eine Größe von etwa 1 KByte. Dieser Umfang ist aber keineswegs genormt, sondern kann je nach Größe der Computeranlage variieren.

Für das BIOS sind folgende Speicherbereiche reserviert:

- Für jedes existierende Laufwerk je eine 16 Bytes lange Haupttabelle, aus der das BDOS die Lage weiterer Tabellen über die Daten des Laufwerkes ersehen kann: der Disk-Parameter-Header.
- Für jedes existierende Laufwerk je ein Zwischenspeicher, in dem die Prüfsumme der Inhaltsverzeichniseinträge abgelegt werden kann. Die Größe dieses Bereiches ist von den Laufwerksdaten abhängig.
- Für jedes existierende Laufwerk je ein Zwischenspeicher, in dem die Blockbelegungstabelle der eingelegten Diskette gespeichert werden kann. Die Mindestgröße dieses Bereichs ist von den Laufwerksdaten abhängig.
- Für jeden angeschlossenen Laufwerkstyp je eine 15 Bytes lange Tabelle, aus der alle zur Verwaltung nötigen Informationen über die spe-

ziellen Daten dieses Laufwerkstyps gelesen werden können: der Disk-Parameter-Block.

- Nur für bestimmte Diskettenformate eine Tabelle, mit Hilfe der die virtuellen Sektornummern der Diskette in die realen Nummern übersetzt werden können.
- Ein 128 Bytes langer Speicherbereich, um Informationen aus dem Inhaltsverzeichnis zwischenzuspeichern.

Wenn die Sektoren der Disketten länger als 128 Bytes sind, muß im BIOS-RAM mindestens ein Zwischenspeicher vorhanden sein, in dem ein kompletter Sektor Platz findet. Dieser Zwischenspeicher wird aber nur vom BIOS benötigt.

Weiterhin können wie in jedem Programm noch kleinere Bereiche reserviert sein, in denen das BIOS Zwischenergebnisse aus Berechnungen abspeichern kann oder ähnliches.

Alle Zwischenspeicher können an beliebigen Stellen im BIOS-Bereich des Speichers stehen. Das gilt auch für die Tabellen und Zwischenspeicher, die nur vom BDOS benötigt werden.

Der dritte Bereich des BIOS sind die eigentlichen Routinen. Diese können ebenfalls an beliebigen freien Speicherstellen stehen. Im Gegensatz zu den anderen Betriebssystemteilen können diese Routinen bei entsprechendem Aufbau auch in einem ROM stehen. Bei den CPC-Computern ist das auch tatsächlich der Fall: Von den gesamten ROMs in den CPC-Computern werden ca. 8 KByte unter CP/M verwendet. Denkbar wäre sogar eine CP/M-Version, in der die BIOS-Routinen und alle nicht vom BDOS benötigten Speicher (zum Beispiel der Bildschirm) in einer anderen Speicherbank liegen, so daß für TPA und BDOS volle 64 KByte Speicher zur Verfügung stehen. Das ist erlaubt, solange die Speicherverwaltung so organisiert ist, daß die zusätzlichen Speicherbänke für das BDOS und die Anwenderprogramme 'unsichtbar' bleiben.

Nur in den CPC-Computern liegt ab der Adresse B900h eine weitere Sprungtabelle mit etwa 350 Einträgen. Diese ist analog zur BIOS-Sprungtabelle aufgebaut und ist sozusagen die BIOS-Schnittstelle zum Schneider-internen Betriebssystem. Unter CP/M liegt aber der größte Teil dieser Sprungtabelle brach, denn über sie werden zum Beispiel die Graphik- und Soundroutinen der CPC-Computer angesprochen. Wenn Sie die genauen Bedeutungen dieser Einsprünge kennen, dürfen Sie sie ohne weiteres auch in CP/M-Programmen verwenden. Sie sollten sich aber bewußt sein, daß diese Programme dann aber nicht mehr auf jedem beliebigen CP/M-Computer laufen werden, sondern nur auf einem Schneider-Computer.

Allgemein übliche Prinzipien in einem BIOS

Obwohl vom Standard nicht verlangt, haben sich doch einige Verfahren zum Bearbeiten bestimmter Vorgänge durchgesetzt, die immer wieder gesehen werden. Dazu tragen sicher die Vorschläge zum Aufbau eines BIOS bei, die von Digital Research zur CP/M-Dokumentation geliefert werden. Einzelne BIOS-Versionen können aber in Details durchaus davon abweichen.

Die Ansteuerung der Ein-/Ausgabegeräte

Das BDOS stellt an der Adresse 0003h des Arbeitsspeichers das Input/Output-Byte zur Verfügung, das Informationen darüber enthält, welchen realen Geräten die logischen Ausgabekanäle zugeordnet werden. Komplexere BIOS-Versionen werten dieses Byte vor jeder Zeichenübertragung aus und verteilen die Ein-/Ausgaben auf verschiedene Geräte. Das ist aber durchaus nicht unbedingt nötig: Das BDOS übt keinerlei Kontrolle aus, ob die Gerätevorschläge auch eingehalten werden. Es gibt durchaus CP/M-Computer bei denen unabhängig vom Wert des I/O-Bytes immer dieselben Geräte angesprochen werden.

```
Konsolenkanal:
  TTY: = Serielle Ein-/Ausgabe 1 (SIO 1)
  CRT: = Eingabe: Tastatur, Ausgabe: Bildschirm
  BAT: = Eingabe: Reader-Kanal, Ausgabe: Drucker
  UC1: = Serielle Ein-/Ausgabe 2 (SIO 2)

Lochstreifenleserkanal:
  TTY: = Serielle Eingabe 1 (SIO 1)
  PTR: = nicht implementiert
  UR1: = Serielle Eingabe 2 (SIO 2)
  UR2: = Tastatur

Lochstreifenstanzerkanel:
  TTY: = Serielle Ausgabe 1 (SIO 1)
  PTP: = nicht implementiert
  UP1: = Serielle Ausgabe 2 (SIO 2)
  UP2: = Bildschirm

Druckerkanal:
  TTY: = Serielle Ausgabe 1 (SIO 1)
  CRT: = Bildschirm
  LPT: = Drucker
  UL1: = Serielle Ausgabe 2 (SIO 2)
```

Bild 8-2: *Ein-/Ausgabegeräte der Schneider-Computer*

Zumindest das BIOS, das zur Schneider 3"-Floppydisk geliefert wird, wertet diese Informationen aus, mit der Gerätezuordnung wurden aber eigene Wege gegangen. Das ist hier auch sinnvoll, denn welcher Homecomputerbesitzer hat schon einen Fernschreiber und möchte den anstelle des Monitors

anschließen. Bild 8-2 zeigt die realen Geräte, die unter CP/M angesprochen werden können. Die Abkürzungen SIO1 und SIO2 bedeuten dabei 'Serielle Ein-/Ausgabe 1' und 'Serielle Ein-/Ausgabe 2'. Offenbar wurden hier schon Vorbereitungen für ein später erhältliches Zusatzgerät zu den Schneider-Computern getroffen.

Durch raffinierte Zuweisungen der Ausgabegeräte können Sie so hübsche Effekte erzielen: Zum Protokollieren eines Programmablaufs drücken Sie normalerweise die ^P-Taste. Dieses Verfahren hat jedoch einen Haken: Nach jedem Warmstart wird der Drucker automatisch abgeschaltet. Das Ablaufen von Submit-Programmen läßt sich so zum Beispiel nicht protokollieren. Wenn Sie jetzt aber dem Reader-Kanal das Gerät UR2 (Tastatur) zuweisen und der Konsole das Gerät BAT (Batch-Modus), laufen danach alle Eingaben dauerhaft über den Reader-Kanal, was aber weiterhin die Tastatur bleibt. Alle Ausgaben laufen dauerhaft über den Drucker. Leider können Sie dann aber auf dem Bildschirm nicht mehr verfolgen, was Sie eintippen.

Die Gerätezuweisungen des BIOS der Vortex-Floppydisk sind ähnlich gestaltet. Dabei gibt es nur 2 Ausnahmen: Das Gerät BAT: im Konsolenkanal und das Gerät CRT: im Druckerkanal sind nicht implementiert. Den eben beschriebenen Trick können Sie also dort nicht ausführen.

Für das Einbinden nicht implementierter Geräte ins BIOS gibt es ebenfalls einige übliche Vorgehensweisen:

- Der Aufruf eines nicht existenten Ausgabegerätes sollte korrekt so abgeschlossen werden, daß das aufrufende Programm glaubt, daß das Zeichen korrekt abgesetzt wurde, die Routine aber trotzdem wirkungslos blieb. Am einfachsten gelingt das mit einem RET-Befehl.
- Der Aufruf eines nicht existenten Eingaberätes sollte das Signal 'End of File' liefern, also das Zeichen ^Z oder 1Ah. Am einfachsten gelingt das mit der Befehlsfolge 'mvi a,1Ah!ret'.

Regeln für die Disketten-Ein- und Ausgabe

Genau wie dem BDOS können auch dem BIOS Daten für die Disketten nur in 128-Byte-Portionen übergeben werden. Hardwaremäßig dürfen die Disketten aber in größere Sektoren aufgeteilt sein. Das BIOS muß dann automatisch die Aufgabe übernehmen, die Records so zusammenzustellen und aufzuteilen, daß das BDOS davon nichts bemerkt. Diese Vorgänge nennt man das 'Sektor Blocking und Deblocking'. Immerhin stellt das BDOS aber bei jedem Schreibbefehl Informationen zur Verfügung, mit denen das BIOS das Blocking optimieren kann:

- Im Normalfall liest das BIOS erst einen ganzen Sektor von der Diskette in den Puffer, ersetzt 128 Bytes durch die zu schreibenden Daten und speichert den Sektor wieder zurück. Ist dieser Record jedoch der erste in einem neu anzulegenden Block, stehen im gelesenen Sektor nur zufällige Werte. Diesen Sachverhalt teilt das BDOS dem BIOS im Falle des Falles mit. Das BIOS verzichtet dann auf das unsinnige vorbereitende Lesen.
- Damit für mehrere zu schreibende Sektoren nur ein Schreibzugriff notwendig ist, werden die Daten eines Records zunächst nur im Sektorpuffer zwischengespeichert. Erst wenn der Sektorpuffer für Daten eines anderen Sektors benötigt wird, erfolgt das Zurückschreiben. Diese Methode birgt jedoch eine große Gefahr in sich: Nach dem Schließen einer Datei könnte die Diskette gewechselt oder ein Warmstart ausgeführt werden. Der Sektorpuffer enthält dann aber noch nicht zurückgeschriebene Daten des Inhaltsverzeichnisses. Beim nächsten Diskettenzugriff würde der Sektorpuffer auf die falsche Diskette abgespeichert. Die zuletzt bearbeitete Datei der alten Diskette wäre dann nicht geschlossen und das Inhaltsverzeichnis der neuen enthielte unsinnige Daten. Das darf natürlich nicht sein. Das BDOS teilt dem BIOS deshalb mit, ob mit dem Schreiben ein Zugriff auf das Inhaltsverzeichnis erfolgt. In diesem Fall schreibt das BIOS den Sektor sofort auf die Diskette zurück, auch wenn noch kein neuer Sektor adressiert wurde. Dadurch stört ein Diskettenwechsel nicht mehr.
- Ist der der zu schreibende Record weder für das Inhaltsverzeichnis noch für einen neu anzulegenden Block bestimmt, erfolgt das Blocking/Deblocking ganz normal.

Sind Ihre Disketten wirklich in physischen 128-Byte-Sektoren organisiert, sind diese Informationen zum Blocking/Deblocking natürlich überflüssig. Das BIOS beachtet diese in diesem Fall einfach nicht.

Gelegentlich ruft das BDOS die BIOS-Routinen zur Disketten-/Spurwahl auf, obwohl die gewählte Spur dann gar nicht benötigt wird. Üblicherweise speichert ein BIOS bei einem Aufruf dieser Funktionen die gewählten Nummern nur in irgendeiner Speicherstelle ab. Erst unmittelbar vor dem endgültigen Lese-/Schreibzugriff wird die endgültige Spur und Laufwerkswahl ausgeführt. So werden unnötige Positionierungen des Schreib-/Lesekopfes und unnötiges Sektorpuffer-Zurückschreiben vermieden.

Gelegentlich treten bei jedem Diskettenlaufwerk Lesefehler auf. Die meisten werden jedoch nur von zufälligen, nicht wiederholbare Störungen bei der Datenübertragung verursacht. Beim nächsten Versuch können die Daten daher schon wieder richtig gelesen werden. In so einem Fall wäre es sehr ungünstig, wenn sofort eine 'Bad Sector'-Fehlermeldung auftreten würde.

Üblicherweise unternimmt ein BIOS bei einem nicht erfolgreichen Zugriff deshalb selbständig bis zu 10 weitere Leseversuche. Nach dem 5. Vesuch wird zusätzlich der Schreib-/Lesekopf neu einjustiert. Erst wenn der gewünschte Sektor dann immer noch nicht zu lesen war, wird eine Fehlermeldung an das BDOS übergeben. Durch diese Methode ist ein weitgehend störungsfreier Betrieb des CP/M-Systems gewährleistet.

Das in den Schneider-Computern vorhandene BIOS hält sich auch an diese Richtlinien, so daß Sie bei Ihrer Programmierarbeit in keinem Punkt Einschränkungen hinnehmen müssen.

BIOS-Versionen anderer Computerhersteller können zusätzlich noch 'Sektor- und Spur-Übersetzungen' beinhalten: Die logischen Sektor- und Spurnummern, die das BDOS verwendet, werden vor der Weitergabe an die Laufwerke durch andere, den physischen Daten der Disketten entsprechende Werte ersetzt. Üblich ist bei doppelseitigen Laufwerken bei manchen Herstellern, den Spuren auf der Oberseite die Nummern 0 bis 39 (79) und den Spuren auf der Unterseite die Nummern 40 (80) bis 79 (159) zu geben. Andere wieder geben den Spuren auf der Oberseite gerade Nummern und den Spuren auf der Unterseite ungerade Nummern. Dadurch können jeweils 2 Spuren mit benachbarten Nummern erreicht werden, ohne den Schreib-/Lesekopf zu bewegen. Häufig ist auch die Variante, einer Spur auf der Oberseite und der entsprechenden auf der Unterseite eine gemeinsame Nummer zu geben. Ober- und Unterseite wird dann mit Hilfe von zusätzlichen (logischen!) Sektornummern unterschieden. Bei den Sektoren gibt es natürlich auch wieder verschiedene Numerierungsmöglichkeiten.

Aufruf von BIOS-Routinen aus Anwenderprogrammen

Auch aus Anwenderprogrammen können Sie direkt BIOS-Routinen aufrufen. Programme, die BIOS-Aufrufe beinhalten, sind aber nur unter Ihrer CP/M-Version lauffähig. Sie können solche Programme zum Beispiel unter CP/M 3.1 nicht mehr weiterverwenden.

Besonders vorsichtig sollten Sie mit den BIOS-Funktionen zum Schreiben/Lesen von Daten umgehen. In der Regel kann das geordnete Bearbeiten von Dateien und der direkte Zugriff auf bestimmte Spuren und Sektoren nicht gemischt werden: Vor dem Weiterarbeiten mit BDOS-Funktionen muß das Laufwerk zurückgesetzt und der Datenpuffer neu festgelegt werden. Diese Einschränkung bedingt gleichzeitig, daß auch keine Dateien während dieser Diskettenbearbeitung geöffnet sein dürfen. Wenn Sie also mehrere Records von einer bestimmten Diskettenstelle lesen und in eine Datei übertragen wollen, müssen Sie zunächst alle zu lesenden Daten im RAM zwischenspeichern, bevor Sie mit dem Schreiben beginnen.

BIOS-Aufruf in Assembler

Der Sprungbefehl an der Adresse 0 zeigt direkt auf den Warmstart-Einsprung WBOOT der BIOS-Sprungtabelle. Entsprechend zur Notation im Bild 8-1 enthalten die Adressen 0001h/0002h also den Wert BIBASE+03h. In einem Anwenderprogramm können Sie daraus leicht die Anfangsadresse der BIOS-Sprungtabelle und die Adressen der einzelnen Routinen berechnen. In Assembler könnte der Aufruf der Routine zur Bildschirm-Zeichenausgabe so aussehen:

```
mvi  c,'X'      ;auszugebendes Zeichen
lxi  d,0Ch-3    ;0Ch ist Funktionsnummer
lhld 0001h      ;Sprungtabellenanfang+3
dad  d          ;hl enthält Adresse der Routine
pchl            ;Sprung zur Adresse (hl)
```

Die Zahl, mit der das DE-Registerpaar geladen wird, ist der um 3 verminderte Wert des im Bild 8-1 angegebenen Versatzes hinter BIBASE.

Bestehende Programme, die BIOS-Aufrufe enthalten, verwenden in der Regel auch die Speicherzellen 0001h und 0002h. Wenn Sie vor dem TPA-Ende einen dauerhaft geschützten Speicherbereich erzeugen und zu diesem Zweck die Warmstartadresse verändern, suchen die Programme die BIOS-Tabelle an einer falschen Stelle. Vor dieser Änderung sollten Sie also die gesamte BIOS-Tabelle in einen neuen Bereich kopieren. Besonders gefährlich ist in diesem Zusammenhang das Programm XSUB.COM: Es verändert zwar den Warmstarteinsprung an der Adresse 0000h, kopiert aber nicht die Sprungtabelle. Beim Versuch, mit aktiviertem XSUB BIOS-Rotinen zu verwenden, wird mit Sicherheit der Rechner abstürzen.

BIOS-Aufruf in Turbo-Pascal

Wesentlich einfacher ist der Aufruf der BIOS-Routinen in Turbo-Pascal: Hier stehen die Funktionen *BIOS*, *BIOSHL* und die Prozedur *BIOS* zur Verfügung. Analog zu den BDOS-Funktionen/Prozeduren wird auch den BIOS-Operationen als erster Parameter die Funktionsnummer und als zweiter der eigentliche Parameter, den die Funktion verarbeitet, übergeben. BDOS-Operationen, die ein Ergebnis liefern werden mit den Funktionen *BIOS* und *BIOSHL* aufgerufen. Die Funktion *BIOS* liefert den Wert, der nach der Verarbeitung im Register A und *BIOSHL* den Wert, der im Register HL steht. Bei Operationen, die kein Ergebnis liefern, ergeben sich zufällige Werte.

In Turbo-Pascal sieht der Aufruf zur Bildschirmausgabe des Zeichens 'X' so aus:

```
bios(4,ord('X'))
```

Beachten Sie, daß die Funktionsnummer (hier 4) genau ein Drittel des im Bild 8-1 angegebenen Versatzes beträgt. Die Funktion *ord('X')* berechnet den Ascii-Wert des Zeichens 'X'.

Hier gilt für die Verwendung von BIOS-Routinen dasselbe, wie bei der Assemblerprogrammierung: BIOS-Funktionen zum Diskettenzugriff und normale Dateioperationen dürfen nicht gleichzeitig verwendet werden.

8.2 Die Standard-BIOS-Funktionen

8.2.1 BIOS-Routinen zur Systeminitialisierung

Der Kaltstart (BOOT)

Funktion: *Computer vollständig zurücksetzen.*

Sofern diese Routine überhaupt funktioniert, wird der Computer bei deren Anwendung so zurückgesetzt, als ob er aus- und wiedereingeschaltet worden wäre.

Nach Abschluß der Initialisierung wird das BDOS und der CCP in das RAM geholt. Danach wird durch einen Sprung zur Adresse FBASE-0806h der CCP gestartet. Dem CCP muß im C-Register das Standard-Bezugslaufwerk und die Standard-Usernummer übergeben werden. In der Regel also der Wert 0 für Laufwerk A und Usernummer 0. Für Spezialzwecke kann das aber auch anders sein.

Adresse: BIBASE+00h (Funktion Nr. 0)

Aufruf: nichts

Rückgabe: nichts

Beispiel:

```
lxi  d,00h-3
lhld 0001h
dad  d
pchl
```

---> Der Rechner wurde vollständig zurückgesetzt

Der Warmstart (WBOOT)

Funktion: *Beenden eines Anwenderprogramms*

Durch den Ansprung dieser Routine wird ein Anwenderprogramm abgebrochen und der CP/M-Prompt 'A>' oder 'B>' erscheint. In der Regel wird diese Routine nur indirekt über den Sprungbefehl an der Adresse 0 oder über die BDOS-Funktion Nummer 0 angesprungen.

Die Warmstartroutine führt folgendes durch:

1. CCP und BDOS von den Systemspuren des Laufwerkes A laden und so restaurieren.
2. Die Sprungbefehle an den Adressen 0000h und 0005h restaurieren.
3. Falls in der Speicherstelle 0004h ein nicht existentes Laufwerk als Bezugslaufwerk gewählt wurde, sollte diese auf einen gültigen Wert gesetzt werden. Andernfalls muß die Speicherstelle 4 unverändert bleiben.
4. Durch einen Sprung zur Adresse FBASE-0806h oder wahlweise FBASE-0803h ruft die Warmstartroutine danach CCP auf. Dem CCP muß dabei im Register C der Wert der Speicherstelle 0004h (Usernummer+Bezuglaufwerk) übergeben werden.

Adresse: BIBASE+03h (Funktion Nr. 1)

Aufruf: nichts

Rückgabe: nichts

Beispiel:

```
jmp  0000h
```

---> Das Anwenderprogramm wird abgebrochen.

8.2.2 BIOS-Routinen zur Ein- und Ausgabe

Der Konsolenstatus (CONST)

Funktion: *Feststellen, ob am Konsolenkanal ein Zeichen anliegt.*

Diese Routine stellt anhand des Bytes an der Adresse 0003h fest, welches reale Ausgabegerät dem Konsolenkanal zugeordnet ist und teilt dem aufrufenden Programm mit, ob auf diesem Gerät ein Zeichen zum Lesen bereit liegt.

Wenn kein Zeichen anliegt, kann ein Anwenderprogramm darauf eingehen und auf den fruchtlosen Leseversuch verzichten. Stattdessen kann etwas anderes erledigt werden.

Adresse: BIBASE+06h (Funktion Nr. 2)

Aufruf: nichts

Rückgabe: a:= 255, wenn ein Zeichen bereit liegt,
0, wenn kein Zeichen bereit ist.

Beispiel:

```
      lxi  d,0006h-3
wait:push d
      call bios
      pop  d
      ana  a
      jz   wait
      jmp  ok
bios:lhld 0001h
      dad  d
      pchl
```

---> Dieses Programm wartet solange, bis eine Taste gedrückt wird.

Die Konsoleneingabe (CONIN)

Funktion: *Zeichen vom Konsolenkanal einlesen.*

Anhand des Wertes in der Speicherstelle 0003h wird festgestellt, von welchem Eingabegerät ein Zeichen gelesen werden soll. In der Regel wird das jedoch die Tastatur sein.

Die Routine wartet solange, bis ein Zeichen zur Verfügung steht und kehrt erst dann zum aufrufenden Programm zurück. Stand bei der letzten Konsolenstatus-Abfrage ein Zeichen zur Verfügung kehrt die Routine sofort zurück und liefert dieses Zeichen, auch wenn die Taste inzwischen wieder losgelassen wurde.

Wenn Sie also während des Wartens auf einen Tastendruck die Kontrolle über den Computer behalten möchten um jederzeit abbrechen zu können, sollten Sie diesen Wartevorgang mit mit Hilfe der Konsolenstatus-Routine erledigen.

Adresse: BIBASE+09h (Funktion Nr. 3)

Aufruf: nichts

Rückgabe: a:= Code eines Zeichens

Beispiel:

```
      lxi  d,0009h-3
      call bios
      jmp  ok
bios:lhld 0001h
      dad  d
      pchl
```

---> Im Register A steht der Code der gedrückten Taste.

Die Konsolenausgabe (CONOUT)

Funktion: *Ausgabe eines Zeichens auf den Konsolenkanal.*

An das durch die Speicherstelle 0003h bestimmte Ausgabegerät wird ein Zeichen übergeben und in das aufrufende Programm zurückgekehrt. Wenn das Ausgabegerät noch nicht bereit ist, ein Zeichen zu übernehmen, versucht die Routine die Übertragung solange, bis sie klappt.

Die Routine darf auch eine einfache Verarbeitung der auszugebenden Zeichen vornehmen. Möglich wäre zum Beispiel eine Umkodierung von in Anwenderprogrammen verwendeten Steuerzeichen in dem System verständliche Codes. Manche Terminaltypen benötigen nach jedem gesendeten Zeichen eine kurze Pause: Alle während dieser Pause gesendeten Zeichen gehen verloren. Das BIOS darf deshalb ohne weiteres diese Zeit durch Warten überbrücken. In den CPC-Computern wurde aber von dieser Möglichkeit kein Gebrauch gemacht.

Adresse: BIBASE+0Ch (Funktion Nr. 4)

Aufruf: c:= Code des auszugebenden Zeichens

Rückgabe: nichts

Beispiel:

```
mvi  c,'Y'
lxi  d,000Ch-3
lhld 0001
dad  d
pchl
```

---> Das Zeichen 'Y' wird an den Konsolenkanal ausgegeben.

Die Druckerausgabe (LIST)

Funktion: *Ausgeben eines Zeichens auf den Druckerkanal.*

Das der Routine übergebene Zeichen wird auf das durch das Byte an der Adresse 0003h bestimmte Ausgabegerät ausgegeben. Die Routine wartet in jedem Fall solange, bis das Zeichen wirklich an das Gerät ausgegeben wurde. Sollte das Gerät also versehentlich nicht eingeschaltet sein, erhält das aufrufende Programm nie mehr die Kontrolle über den Computer zurück.

Adresse: BIBASE+0Fh (Funktion Nr. 5)

Aufruf: c:= Code des auszugebenden Zeichens

Rückgabe: nichts

Beispiel:

```
mvi  c,'A'
lxi  d,000Fh-3
lhld 0001
dad  d
pchl
```

---> Das Zeichen 'A' wird an den Drucker ausgegeben.

Der Druckerstatus (LISTST)

Funktion: *Feststellen, ob der Druckerkanal bereit ist, ein Zeichen zu übernehmen.*

Mit Hilfe dieser Routine kann schon vor dem Druckversuch festgestellt werden, ob der Druckerkanal bereit ist, ein Zeichen zu übernehmen oder nicht. Das 'Aufhängen' des Computers durch einen fruchtlosen Druckversuch kann so vermieden werden. Das ist zum Beispiel für Spooler wichtig, die während der Computer nichts besseres zu tun hat, als auf einen Tastendruck durch den Anwender zu warten, ein paar Zeichen einer Datei auf den Drucker ausgeben. Damit ist gleichzeitiges Drucken und Arbeiten mit dem Computer möglich.

Wenn diese Funktion auf einem Computer nicht implementiert werden kann, muß sie immer den Wert 00h zurückgeben. In den CPC-Computern ist die Funktion aber vorhanden.

Adresse: BIBASE+2Dh (Funktion Nr. 15)

Aufruf: nichts

Rückgabe: c:= 255, wenn der Drucker bereit ist, ein Zeichen zu übernehmen,
0, wenn der Drucker kein Zeichen übernehmen kann oder die Statusfunktion nicht implementiert ist.

Beispiel:

```
     lxi  d,002Dh-3
wait:push d
     call bios
     pop  d
     ana  a
     jz   wait
     jmp  ok
bios:lhld 0001h
     dad  d
     pchl
```

---> Dieses Programm wartet solange, bis das Gerät im Druckerkanal ein Zeichen übernehmen kann und springt dann zur Stelle 'ok'.

Der Lochstreifenstanzer (PUNCH)

Funktion: *Ausgabe eines Zeichens an den Lochstreifenstanzerkanal.*

Anhand des I/O-Bytes stellt diese Routine fest, welches reale Gerät als Lochstreifenstanzer verwendet werden soll und gibt ein Zeichen darauf aus.

Die Routine wartet analog zur Druckerausgaberoutine so lange, bis das Puncher-Gerät bereit ist, ein Zeichen zu übernehmen.

Adresse: BIBASE+12h (Funktion Nr. 6)

Aufruf: c:= Code des auszugebenden Zeichens

Rückgabe: nichts

Beispiel:

```
mvi  c,'A'
lxi  d,0012h-3
lhld 0001h
dad  d
pchl
```

---> Das Zeichen 'A' wird auf das Lochstreifenstanzergerät ausgegeben.

Der Lochstreifenleser (READ)

Funktion: *Einlesen eines Zeichens vom Lochstreifenleser.*

Die Routine gibt das Zeichen, das im durch das I/O-Byte bestimmten Reader-Gerät anliegt an das Hauptprogramm zurück. Wenn kein Zeichen zur Verfügung steht, kehrt die Routine nicht zum aufrufenden Programm zurück, sondern wartet auf ein Zeichen.

Wenn keine weiteren Zeichen mehr zu erwarten sind, muß die Routine den Code 1Ah beziehungsweise das Zeichen ^Z als End-Of-File-Marke zurückgeben. Ist in der Computeranlage kein Readergerät implementiert, sollte immer das Zeichen ^Z zurückgegeben werden.

Adresse: BIBASE+15h (Funktion Nr. 7)

Aufruf: nichts

Rückgabe: a:= Code des gelesenen Zeichens beziehungsweise End-Of-File-Marke

Beispiel:

```
lxi  d,0015h-3
lhld 0001h
dad  d
pchl
```

---> Register A enthält jetzt den Code des auf dem Readerkanal anliegenden Zeichens

8.2.3 BIOS-Routinen zur Laufwerkssteuerung

Schreib-/Lesekopf justieren (HOME)

Funktion: *Justieren des Schreib-Lesekopfes auf die Spur Null*

Bei vielmaliger Kopfbewegung des Laufwerkes können sich mechanische Einstellungenauigkeiten so summieren, daß der Schreib-/Lesekopf über eine falsche Spur positioniert wird. Bei den meisten Laufwerken gibt es deshalb die Möglichkeit, den Schreib-/Lesekopf mit Hilfe einer Lichtschranke oder eines mechanischen Anschlages unabhängig von der aktuellen Position genau auf die Spur Null zu positionieren. Die BIOS-Funktion führt diese Justierung für das aktuelle Laufwerk aus.

In der Praxis wird jedoch beim Aufruf der Funktion nur ein entsprechendes Flag in einer bestimmten Speicherstelle gesetzt. Erst unmittelbar vor dem nächsten echten Schreiben oder Lesen wird die Justierung tatsächlich durchgeführt. So werden mehrmalige Justierungen ohne zwischenzeitliches Lesen/Schreiben vermieden.

Adresse: BIBASE+18h (Funktion Nr. 8)

Aufruf: nichts

Rückgabe: nichts

Beispiel:

```
lxi  d,0018h-3
lhld 0001h
dad  d
pchl
```

---> Beim nächsten Schreib-/Lesebefehl wird das Laufwerk neu justiert.

Aktuelles Laufwerk wählen (SELDSK)

Funktion: *Mit dieser Funktion kann gewählt werden, auf welches Laufwerk die folgenden Schreib-/Lesezugriffe erfolgen.*

Das BDOS (oder ein anderes aufrufendes Programm) übergibt dieser Routine die Nummer des zu wählenden Laufwerkes, wobei das Laufwerk A die Nummer 0, und das Laufwerk P die Nummer 15 hat. Weiterhin übergibt das BDOS einen Code, der angibt ob dieses Laufwerk seit dem letzten Warmstart das erste Mal oder schon zum wiederholten Mal angesprochen wird. Das BIOS zu den 3"-Floppydisks der CPC-Computer wertet diese Information aus: Bei der ersten Verwendung eines Laufwerks wird das Format der Diskette bestimmt. So können auch Disketten im Daten- und im IBM-Format verwendet werden. Bei weiteren Zugriffen ohne zwischenzeitlichen Warmstart kann das Diskettenformat aber nicht mehr verändert werden. Außer in ganz bestimmten Fällen hat diese Formaterkennung aber nur für das Laufwerk B einen praktischen Nutzen: Bei einem Warmstart muß sich im Laufwerk A eine Systemdiskette befinden. Sofort nach dem Warmstart wird dann - ohne daß Sie es verhindern können - eine Formaterkennung durchgeführt.

Als Ergebnis erhält das aufrufende Programm die Adresse des Disk-Parameter-Headers des gewählten Laufwerks. So kann das BDOS die verschiedensten Diskettenformate richtig verwalten. Wenn das gewählte Laufwerk nicht existiert, ergibt sich statt dieser Adresse der Wert Null, so daß das BDOS die Fehlermeldung 'Select' ausgeben kann. Genau wie beim Justieren des Schreib-/Lesekopfes braucht auch hier das echte Aktiveren des Laufwerkes erst erfolgen, wenn wirklich eine Datenübertragung stattfindet.

Adresse: BIBASE+1Bh (Funktion Nr. 9)

Aufruf: c:= Nummer des zu wählenden Laufwerkes,
e:= Bit 0 = 0, wenn eine Disketten-Formaterkennung durchgeführt werden soll,
Bit 0 = 1, wenn das schon beim letzten Zugriff verwendete Format weiterverwendet werden soll.

Rückgabe: hl:= Adresse des zum gewählten Laufwerk gehörenden Disk-Parameter-Headers (DPH), wenn das Laufwerk existiert,
0, wenn das gewählte Laufwerk nicht existiert.

Beispiel:

```
lxi  d,001Bh-3
lhld 0001h
dad  d
mvi  c,00h          ;Laufwerk A
```

```
mvi  e,00h          ;Formaterkennung
pchl
```

---> HL enthält die Adresse des DPH für das Laufwerk A und das Laufwerk ist initialisiert. Dabei wurde auch das Format der Diskette im Laufwerk A bestimmt.

Spur wählen (SETTRK)

Funktion: *Wählen der Diskettenspur, auf die zugegriffen werden soll.*

Alle folgenden Diskettenzugriffe erfolgen auf die hier gewählte Spur des aktuellen Diskettenlaufwerks. Wie üblich wird die tatsächliche Kopfpositionierung aber bis zum tatsächlichen Diskettenzugriff zurückgestellt.

Die erste Spur der Diskette hat die Nummer 0. Zwischen Ober- und Unterseite der Diskette muß der Programmierer nicht unterscheiden: Das BIOS nimmt diese Unterscheidung anhand der Spurnummer selbst vor.

Bei der Programmierung müssen Sie selbst darauf achten, daß nur eine wirklich existierende Spur angesteuert wird: Die Routine liefert bei zu großen Nummern keine Fehlermeldungen, so daß beim Zugriff der Schreib-/Lesekopf bei der letzten bespielten Spur nicht stoppt.

Adresse: BIBASE+1Eh (Funktion Nr. 10)

Aufruf: bc:= zu wählende Spurnummer

Rückgabe: nichts

Beispiel:

```
lxi  b,0010h
lxi  d,001Eh-3
lhld 0001
dad  d
pchl
```

---> Der nächste Schreib-/Lesezugriff erfolgt auf die Spur 16 der aktuellen Diskette

Sektor wählen (SETSEC)

Funktion: *Wahl des Sektors, in den als nächstes geschrieben wird.*

Trotz des Namens 'Sektor' wird mit dieser Routine die Nummer eines 128 Bytes langen Aufzeichnungsabschnitts festgelegt, gleichgültig wie groß die physischen Sektoren sind.

Wenn die Diskette in 128 Bytes langen physischen Sektoren organisiert ist, muß dieser Routine die physische und nicht die logische Sektornummer übergeben werden: Aufeinanderfolgende Records einer Datei können durchaus nicht aufeinanderfolgende Sektornummern haben. Vor dem Zugriff müssen diese Nummern also noch umgerechnet werden. Dafür gibt es aber eine besondere BIOS-Funktion.

Die dieser Routine zu übergebenden Sektornummern liegen zwischen 1 und der maximalen Sektornummer. Die SETSEC-Funktion übernimmt aber keine Kontrolle, ob die übergebene Nummer auch korrekt ist. Bei unsinnigen Nummern ergeben sich also später Lesefehler.

Adresse: BIBASE+21h (Funktion Nr. 11)

Aufruf: bc:= physische Sektornummer

Rückgabe: nichts

Beispiel:

```
lxi  b,000Ah        ;physische Sektornummer
lxi  d,0021h-3
lhld 0001h
dad  d
pchl
```

---> Der nächste Diskettenzugriff erfolgt auf den Sektor Nummer 10 der schon früher gewählten Spur des aktuellen Diskettenlaufwerkes.

Sektornummer übersetzen (SECTRAN)

Funktion: Übersetzung der logischen in die physische Sektornummer.

Der BIOS-Routine zur Sektorwahl müssen physische Sektornummern übergeben werden. Aufeinanderfolgende Records einer Aufzeichnung haben aber aufeinanderfolgende logische Sektornummern, die von den physischen Sektornummern abweichen können. Dadurch erhoffen sich die Hersteller einen schnelleren Diskettenzugriff. Die SECTRAN-Routine übernimmt die Umrechnung der logischen in die physischen Nummern, weshalb diese Routine unmittelbar vor jeder SETSEC-Routine aufgerufen werden sollte.

Außer der logischen Sektornummer wird dieser Routine auch noch die Anfangsadresse einer Umrechnungstabelle übergeben. Die Adresse dieser Tabelle kann das BDOS den ersten beiden Bytes des Disk-Parameter-Headers des aktuellen Laufwerks entnehmen.

Wenn die Diskette wie bei den CPC-Computern in Sektoren mit mehr als 128 Bytes organisiert ist, wird keine Sektorübersetzung vorgenommen. Dafür enthält das BIOS einen Blocking/Deblocking-Algorithmus. In diesem Fall enthalten die ersten Bytes des DPH den Wert Null und die SECTRAN-Routine gibt die Sektornummer unverändert zurück.

Adresse: BIBASE+30h (Funktion Nr. 16)

Aufruf: bc:= logische Sektornummer
de:= Adresse der Übersetzungstabelle aus dem DPH

Rückgabe: hl:= physische Sektornummer

Beispiel:

```
lxi  b,0010h        ;logische Sektornummer
lxi  d,table        ;Adresse der Übersetzungstabelle
call SECTRAN
mov  b,h
mov  c,l            ;physische Sektornummer berechnet
call SETSEC         ;Sektor wählen
```

---> Beim nächsten Diskettenzugriff wird der Sektor mit der Nummer 10 bearbeitet.

8.2.4 BIOS-Routinen zum Schreiben/Lesen von Daten

Pufferadresse festlegen (SETDMA)

Funktion: *Adresse des 128-Byte-Datenpuffers festlegen, aus dem Daten auf die Diskette übertragen werden oder in dem Daten von der Diskette abgespeichert werden.*

Die BIOS-Funktion SETDMA entspricht genau der gleichnamigen BDOS-Funktion. Auch hier ist der Bereich von 0080h bis 00FFh der Standard-Datenpuffer, der solange die SETDMA-Funktion noch nicht aufgerufen wurde verwendet wird.

Adresse: BIBASE+24h (Funktion Nr. 12)

Aufruf: bc:= Anfangsadresse des Datenpuffers

Rückgabe: nichts

Beispiel:

```
lxi  b,1000h
lxi  d,0024h-3
lhld 0001
dad  d
pchl
```

---> Alle Daten von den Disketten werden ab jetzt im Speicherbereich von 1000h bis 107Fh abgelegt.

Logischen Sektor von der Diskette lesen (READ)

Funktion: *Liest 128 Bytes Daten von der Diskette.*

Übertragen wird der mit Hilfe der Funktionen SELDSK, SETSEC und SETTRK gewählte Sektor auf einem der angeschlossenen Laufwerke in den Datenpuffer. Die gewählten Nummern werden durch das Lesen nicht verändert, so daß wiederholtes Lesen ohne sonstige Vorkehrungen immer auf denselben Sektor wirkt.

Der aufrufenden Routine wird mitgeteilt, ob das Lesen erfolgreich war oder nicht. Der einzige Fehler, der in dieser Ebene auftreten kann, ist ein echter Lesefehler, zum Beispiel durch eine fehlende Diskette oder der Adressierung eines nicht existenten Sektors.

Wenn sich der gesuchte Record schon in einem BIOS-internen Puffer befindet, braucht kein echter Diskettenzugriff ausgeführt werden. Bei Disketten mit mindestens 256 Bytes langen physischen Sektoren ist das immer dann der Fall, wenn aufeinanderfolgende 128 Byte-Records gelesen werden.

Adresse: BIBASE+27h (Funktion Nr. 13)

Aufruf: nichts

Rückgabe: a:= 0, wenn kein Fehler auftrat,
1, wenn das Lesen nicht erfolgreich war.

Beispiel:

```
lxi  d,0027h-3
lhld 0001h
dad  d
pchl
```

---> Im Datenpuffer befindet sich der schon früher gewählte 128-Byte-Ausschnitt der aktuellen Diskette.

Logischen Sektor auf die Diskette schreiben

Funktion: *Schreibt 128 Bytes Daten auf die Diskette.*

Diese Funktion kopiert den Datenpuffer auf die gewählte Position des aktuelle Diskettenlaufwerkes. Diese Position wird dabei nicht verändert, so

daß aufeinanderfolgende Schreibbefehle ohne zusätzliche Maßnahmen immer auf denselben Diskettensektor wirken.

War das Schreiben nicht erfolgreich, wird eine Fehlermeldung zurückgegeben. Das kann wie beim Lesen nur bei einem physischen Diskettenfehler auftreten.

In der Regel braucht der logische Sektor nicht sofort auf die Diskette zurückgeschrieben zu werden: Ein echter Diskettenzugriff erfolgt erst, wenn der BIOS-interne Datenpuffer für andere Zwecke benötigt wird. In bestimmten Fällen kann jedoch davon abgewichen werden. Die aufrufende Routine übergibt dazu der BIOS-Funktion einen zusätzlichen Wert, um das zu steuern:

0 Wenn das Blocking/Deblocking ganz normal durchgeführt werden soll.
1 Wenn der logische Sektor sofort auf die Diskette zurückgespeichert werden soll. Das BDOS wählt diese Aufzeichnungsvariante, wenn ins Inhaltsverzeichnis der Diskette geschrieben wird und ein nachfolgender Diskettenwechsel erlaubt sein soll. Sie können das aber auch in anderen Fällen anfordern.
2 Wenn das Lesen des physikalischen Sektors vor dem eigentlichen Schreiben entfallen kann. Das BDOS übergibt diesen Wert, wenn ein bisher nicht belegter Sektor erstmals beschrieben wird, so daß der BIOS-interne Sektorpuffer nicht mit sinnvollen Werten initialisiert zu werden braucht.

Adresse: BIBASE+2Ah (Funktion Nr. 14)

Aufruf: c:= 0, zum Schreiben mit normalem Blocking,
1, zum Schreiben mit sofortigem Zurückspeichern,
2, zum Schreiben ohne vorbereitendes Lesen.

Rückgabe: a:=0, wenn das Schreiben erfolgreich war,
1, wenn ein Fehler auftrat.

Beispiel:

```
        mvi  c,01h
        lxi  d,002Ah-3
        call bios
        ana  a
        jnz  error
        jmp  ok
bios:lhld 0001h
        dad  d
        pchl
```

---> Der Datenpuffer wurde in den schon früher gewählten Diskettensektor übertragen und sofort zurückgespeichert.

8.3 Die zusätzlichen BIOS-Routinen der CPC-Computer

Die bisher besprochenen BIOS-Routinen stehen in jedem beliebigen CP/M 2.2-Computer in genau der gleichen Weise zur Verfügung. Wenn Sie sich also auf die Verwendung dieser Routinen beschränken, sind Ihre Programme immer noch auf mehreren Computertypen lauffähig - wenn auch nur unter der Betriebssystem-Version CP/M 2.2 und nicht etwa unter CP/M 3.1.

In den Schneider-Computern gibt es aber noch einige zusätzliche BIOS-Routinen, die von Anwenderprogrammen unter CP/M verwendet werden dürfen. Mit deren Hilfe können Sie einige tiefer in die Hardware eingreifende Probleme bequem lösen - zum Beispiel eine Diskettenspur in einem speziellen Sonderformat zu formatieren. Solche Programme sind dann natürlich nur auf einem Schneider-Computer lauffähig, aber immerhin noch unabhängig von der gegenwärtigen Version. Also sowohl auf einem CPC 464 als auch auf einem CPC 664 oder CPC 6128.

In den CPC-Computern werden diese Routinen immer dann verwendet, wenn die Teile BIOS und BDOS des Betriebssystems nicht unbedingt zur Verfügung stehen. Also etwa während der Kaltstartlader auf der Systemdiskette aktiv ist oder ein Warmstart ausgeführt wird. Auch die Standard-BIOS-Routinen stützen sich intern auf diese erweiterten Routinen.

8.3.1 Die zusätzlichen Diskettenbefehle

Die zusätzlichen Diskettenbefehle können über eine zusätzliche Sprungtabelle erreicht werden, die vollständig analog zur standardmäßigen BIOS-Sprungtabelle aufgebaut ist. Diese Sprungtabelle beginnt immer an der Adresse BE80h und reicht bis zur Adresse BEA6h, hat also 13 Einträge.

Jeder Eintrag zeigt auf eine bestimmte ROM-Routine, die auch von den gewöhnlichen BIOS-Routinen verwendet wird. Eine gewöhnlich BIOS-Funktion springt aber direkt in die benötigte Routine, ohne die Umleitung über den Vektor im RAM zu nehmen. Die Sprungtabelle könnte also beim BIOS-internen Gebrauch genausogut fehlen. Im Gegensatz zu den patchbaren 'normalen' Firmware-Einsprüngen in den CPC-Computern nützt hier eine Änderung nichts: Die eingebauten Diskettenroutinen können auf diese Weise also nicht durch eigene ersetzt werden.

Den Aufbau der Sprungtabelle sehen Sie im Bild 8-3.

Adresse	Bedeutung der Routine
BE80	BIOS-Meldungen einschalten/sperren
BE83	Zeitkonstanten für die Laufwerke
BE86	Diskettenformat festlegen
BE89	512 Byte-Sektor sofort lesen
BE8C	512 Byte-Sektor sofort schreiben
BE8F	Spur formatieren
BE92	Schreib-/Lesekopf sofort positionieren
BE95	Laufwerkstatus ermitteln
BE98	Wiederholungen bei Schreib-/Lesefehler
BE9B	Externer BIOS-Zugang
BE9E	Schneider: Fast/Slow-Modus wählen
	Vortex: Laufwerksmotor ausschalten
BEA1	Schneider: SIO initalisieren
	Vortex: nicht benützt
BEA4	Text für automatischen Befehl initialisieren

Bild 8-3: *Zusätzliche BIOS-Funktionen in den CPC-Computern*

Die Sprungtabelle wird sowohl vom originalen Schneider-Floppydiskcontroller als auch vom Controller der Vortex-Floppydisk zur Verfügung gestellt. In der Regel sind auch die dazugehörigen Routinen vollständig kompatibel, so daß Programme, die die zusätzlichen BIOS-Routinen verwenden, sowohl mit einer Vortex-Floppydisk als auch mit einer Schneider-Floppydisk zusammenarbeiten.

Sie können sich darauf verlassen, daß ohne besonderen Hinweis die folgenden Erklärungen sowohl für die Vortex- als auch für die Schneider-Floppydisk gelten. Immer wenn es Unterschiede gibt, werden beide Varianten berücksichtigt.

BIOS-Meldungen einschalten/sperren

Funktion: *Wählen, ob Diskettenfehler auf dem Bildschirm detailliert beschrieben werden.*

Normalerweise erscheint bei Diskettenfehlern die genaue Fehlerursache im Klartext auf dem Bildschirm, zusammen mit der Frage 'Retry,Ignore or Cancel ?' oder einer anderen, sinngemäß gleichen Meldung. Erst wenn die Operation abgebrochen wird, erscheint

```
Bdos Err On <Laufwerk>: Bad Sector
```

oder

```
Bdos Err On <Laufwerk>: Select
```

Die zusätzlichen Meldungen werden vom hardwareabhängigen Teil des BIOS gesteuert und lassen sich für besondere Anwendungen abschalten: Der Benutzer bemerkt dann in Zukunft nur noch die Standard-BDOS-Fehler.

Adresse: BE80h

Aufruf: a:= 0, wenn die BIOS-Meldungen angezeigt werden,
255, wenn die BIOS-Meldungen unterdrückt werden sollen.

Rückgabe: nichts

Beispiel:

```
mvi  a,FFh
call BE80h
```

---> Ab jetzt erscheinen nur noch die Standard-BDOS-Fehlermeldungen.

Zeitkonstanten für die Laufwerke

Funktion: *BIOS an verschiedene Laufwerkstypen anpassen*

Mit dieser Funktion können sie dem BIOS alle wichtigen Zeitkonstanten ihres Laufwerkstyps mitteilen. Dadurch ist es zum Beispiel möglich, ein anderes Laufwerke als das Original-Schneider-Laufwerk zu verwenden und trotzdem den bisherigen Diskcontroller zu behalten.

Beim Aufruf müssen Sie der Routine die Anfangsadresse einer 9 Bytes langen Tabelle übergeben. Diese Tabelle muß die neuen Zeitkonstanten in der Form nach Bild 8-4 enthalten:

Die Werte in den Bytes 4, 7 und 8 der Tabelle greifen massiv in die Datenaufzeichnungsprozedur ein. Sie sollten diese 3 Werte deshalb bei einer eigenen Anwendung der Zeitkonstanten-Routine unverändert übernehmen, um eine sichere Datenaufzeichnung zu gewährleisten.

Das Byte 6 der Tabelle gibt die Zeit an, die der Schreib-/Lesekopf des Laufwerks braucht, um von einer Spur zur nächsten zu wechseln. Nach einer Reihe von Spurwechseln muß vor dem Datenzugriff noch gewartet werden, bis der Schreib-/Lesekopf vollständig zur Ruhe kommt. Die Dauer der Sicherheitszeitspanne gibt das Byte 5 der Tabelle an. Diese beiden Bytes müssen den Leistungsdaten der verwendeten Floppydisks angepaßt werden.

Byte	Größe	Default Wert	Bedeutung
0/1	16 Bit	0032h	Hochlaufzeit des Diskettenmotors (Einheit: 1/50 Sekunde)
2/3	16 Bit		Nachlaufzeit des Diskettenmotors (Einheit: 1/50 Sekunde)
		00FAh	Schneider: 5 Sekunden
		0096h	Vortex: 3 Sekunden
4	8 Bit	AFh	Wartezeit nach dem Sektorschreiben (Einheit: 1/100000 Sekunde)
5	8 Bit	1Eh	Grundwartezeit bei Spurwechsel (Einheit: 1/1000 Sekunde)
6	8 Bit		Zusatzwartezeit pro zu wechselnder Spur (Einheit: 1/1000 Sekunde) intern wird immer auf gerade Werte gerundet
		0Ch	Schneider: Bits 0-7
		C4h	Vortex: Bits 0-3 für eingebautes Laufwerk Bits 4-7 für externes Zusatzdrive
7	8 Bit	01h	Verzögerung bis zum Abheben des Kopfes nach letztem Zugriff (Einheit: 32/1000 Sekunde)
8	8 Bit	03h	Bits 1-7: Verzögerung nach dem Aufsetzen des Kopfes bis zum ersten Zugriff (Einheit: 1/250 Sekunde) Bit 0 muß immer 1 sein, sonst Systemabsturz

Bild 8-4: *Zeitkonstanten für die Diskettenlaufwerke*

Die Nachlaufzeit des Diskettenmotors nach dem letzten Zugriff können Sie frei nach Ihrem Geschmack wählen. In der Regel sollten Sie eine kürzere Zeit wählen, wenn Sie nur ein Diskettenlaufwerk besitzen: Dann können Sie die Disketten schnell wechseln. Wenn Sie zwei Diskettenlaufwerke besitzen (und Sie nur selten wechseln müssen) sollten Sie eine längere Zeit wählen, damit beim nächsten Zugriff die Diskette möglichst noch in Bewegung ist. In diesem Fall wird die Hochlaufzeit des Diskettenmotors eingespart und der Zugriff erfolgt schneller.

Adresse: BE83h

Aufruf: hl:= Anfangsadresse der 9 Bytes langen Zeittabelle

Rückgabe: nichts

Beispiel:

```
        lxi  h,tab
        call BE83h
        jmp  ok
tab:    dw   0032h,00FAh              ;Standardtabelle
        db   AFh,1Eh,0Ch,01h,03h
```

---> Die Zeitkonstanten wurden auf die Werte wie sofort nach dem Einschalten gestellt.

Diskettenformat festlegen

Funktion: *Vorbereiten des Betriebssystems für ein bestimmtes Standard-Diskettenformat.*

Die CPC-Computer kennen mehrere Standard-Diskettenformate. Mit Hilfe dieser Routine kann sehr leicht eines ausgewählt und die Parametertabellen im BIOS mit den passenden Werten belegt werden: Bis zur nächsten Änderung werden dann alle Disketten in diesem Format verwaltet.

Leider ist diese Routine beim Controller der 3"-Floppydisk nicht kompatibel zur entsprechenden Routine des Vortex-Controllers, obwohl eine gewisse Ähnlichkeit vorhanden ist. Im folgenden werden diese beiden Geräte deshalb getrennt behandelt.

Adresse: BE86h

Die Routine im Schneider-Floppydiskcontroller

Hier können nur die 3 Schneider-Formate eingestellt werden. Zur Wahl eines bestimmten Formates können jeweils meherere verschiedene Werte an die Routine übergeben werden.

Aufruf: a:= 40h-5Fh: Vorbereitungen für CP/M-Format
C0h-FFh: Vorbereitungen für Daten-Format
00h-3Fh, 80h-BFh: Vorbereitungen für IBM-Format
e:= Nummer des Laufwerkes, dessen Format geändert wird

Rückgabe: nichts

Die Routine im Vortex-Floppydiskcontroller

Bei der Vortex-Floppydisk hat die Routine an der Adresse BE86h einige weitere Bedeutungen.

1. Möglichkeit:

Bei jedem Aufruf der Möglichkeit 1 wird die Bildschirmausgabe umgeschaltet: Wenn vorher die beschleunigte Ausgabe (FAST-Modus) aktiv war, wird die normale eingeschaltet und umgekehrt. Diese Möglichkeit gibt es aber nur unter VDOS 2.0.

Aufruf: a:= FEh

Rückgabe: nichts

2. Möglichkeit:

Beim Aufruf der 2. Möglichkeit wird die Adresse des Flagbytes des gewählten Laufwerks festgestellt. Dieses Flagbyte kann anschließend entweder nur gelesen oder auch verändert werden. Die einzelnen Bits des Flagbytes haben die in Bild 8-5 beschriebenen Bedeutungen.

Bit	Bedeutung/Wert
0	0 bei einem angeschlossenen Einzelkopflaufwerk 1 wenn ein Doppelkopflaufwerk angeschlossen ist
4	0 beim Lesen/Schreiben eines einzelnen Sektors 1 beim fortlaufenden Lesen/Schreiben von Sektoren
5	0 wenn unter dieser Nummer das eingebaute Laufwerk angesprochen wird. 1 wenn das eingebaute Laufwerk mit dieser Nummer durch ein externes Laufwerk (in der Regel eine 3"-Floppy) ersetzt wird.
6	0 wenn vor dem nächsten Datenzugriff der Schreib-/Lesekopf auf die Spur 0 justiert werden soll 1 wenn der Schreiblesekopf nicht justiert wird

Bild 8-5: *Laufwerks-Flagbyte im Vortex-Floppydiskcontoller*

Bit	Bedeutung/Wert
1	0 wenn der Kaltstartlader Meldungen über die getroffenen Systemvoreinstellungen ausgeben soll 1 wenn diese Meldungen unterdrückt werden
2	(nur unter VDOS 1.0) 0 wenn die BIOS-Fehlermeldungen sichtbar sind 1 wenn die BIOS-Fehler nicht angezeigt werden
3	0 wenn der Diskettenmotor ausgeschaltet ist 1 wenn der Diskettenmotor eingeschaltet ist

Bild 8-6: *Laufwerksunabhängige Bits des Flagbytes*

Im Flagbyte des Laufwerks A sind 3 weitere Bits belegt. Diese Bits beziehen sich aber nicht unbedingt nur auf das Laufwerk A. Bild 8-6 zeigt deren Bedeutungen.

Die Adresse des Flagbytes eines Laufwerks können Sie so erhalten:

Aufruf: a:= 255, als Code für die Adreßberechnung
e:= 0 bzw. 1 für Flagbyte des Laufwerks A oder B

Rückgabe: hl:= Adresse des Flagbytes zum gewählten Laufwerk

3. Möglichkeit:

Mit der letzten Variante können Sie ein eingebautes Laufwerk durch ein externes ersetzen beziehungsweise das externe wieder abschalten.

In der Regel wird das externe Laufwerk ein Schneider 3"-Laufwerk sein. Deshalb werden beim Aufruf dieser Variante die Diskettenparameter gleich für 3"-Disketten vorbereitet. Standardmäßig können nur Disketten im CP/M-Format gelesen werden, aber durch einen manuellen Eingriff in die Parametertabellen können Sie das ändern.

In der Regel ist es unter CP/M nicht sinnvoll, das Laufwerk A durch ein externes zu ersetzen: Da der Warmstart das BDOS immer vom Laufwerk A holt, müssen Sie dieses erst einmal auf eine 3"-Diskette kopieren. Theoretisch möglich wäre dieser Austausch aber schon.

Aufruf: a:= 0,wenn die beiden eingebauten Laufwerke eingeschaltet sind,
1, wenn das eingebaute Laufwerk A durch ein externes ersetzt ist,
2, wenn das Laufwerk B durch ein externes ersetzt ist.

Rückgabe: nichts

Anwendungsbeispiel:
Dieses Beispiel gilt nur für ein Schneider-Laufwerk.

```
mvi  a,C0h          ;Code für Diskette im Datenformat
mvi  e,01h          ;Code für Laufwerk B
call BE86h
```

---> Ins Laufwerk B können ab jetzt auch unter CP/M Disketten im Datenformat eingelegt werden.

512-Byte-Sektor sofort lesen

Funktion: *Beliebigen physischen 512-Byte-Sektor von einer Diskette in den Speicher kopieren*

Diese Routine lädt einen beliebigen Sektor sofort von der Diskette in den Speicher, wobei die Blocking/Deblocking-Routinen des gewöhnlichen BIOS umgangen werden: Die Daten werden ohne Pufferung direkt in den Zielspeicherbereich geschrieben. Insbesondere bleiben dabei der BIOS-interne Sektorpuffer, die Marken für den aktuellen Record/Spur und die DMA-Adresse im Standard-BIOS unverändert. Nur mit Hilfe dieser Routine können Sie also direkte Diskettenzugriffe und geordnete Dateizugriffe bedenkenlos mischen.

Wegen der nicht notwendigen Kopieroperationen der Daten von einem Speicherbereich in einen anderen erfolgen die Diskettenzugriffe mit dieser Routine wesentlich schneller als mit den normalen BIOS/BDOS-Routinen. Dafür können Sie Programme, die diese Routinen verwenden, nicht mit jedem beliebigen Diskcontroller verwenden, den es für CPC-Computer gibt. Als Sektornummern müssen die echten Nummern angegeben werden, die auch auf der Diskette abgespeichert sind. Bei einer 3"-CP/M-Diskette also Nummern zwischen 41h und 49h.

Mit dieser Routine können Sie auch Sektoren lesen, die mehr oder weniger als 512 Bytes besitzen - Sie müssen nur dafür sorgen, daß der freie Speicherplatz in dem der Sektor abgelegt wird genügend Raum bietet.

Tritt ein Lesefehler auf, unternimmt die Routine selbständig eine gewisse Zahl weiterer Leseversuche. Erst wenn Sie dann immer noch nicht erfolgreich ist, wird eine Fehlermeldung ausgegeben.

Adresse: BE89h

Aufruf: e:= Laufwerksnummer (0=A, 1=B)
d:= gewünschte Spur
c:= gewünschte Sektornummer
hl:= Speicheradresse, ab der die Daten abgelegt werden

Rückgabe: cy:= Carry-Flag gesetzt, wenn Lesen erfolgreich war,
Carry-Flag zurückgesetzt, wenn Fehler auftrat.

Beispiel:

```
lxi  d,0000h   ;Laufwerk/Spur
lxi  c,41h     ;Sektornummer
lxi  h,0100h   ;Pufferadresse
call BE89h
jnc  error
```

---> Im Speicher ab der Adresse 0100h befindet sich der erste Sektor der Spur Nr. 0 einer 3"-CP/M-Systemdiskette im Laufwerk A.

512-Byte-Sektor sofort schreiben

Funktion: *Schreiben eines beliebigen physischen Sektors auf eine Diskette in einem der Laufwerke.*

Diese Routine entspricht genau der Routine zum Sektor lesen. Auch hier erfolgt die Übertragung der Daten sofort und ohne Zwischenspeicherung in einem zusätzlichen Sektorpuffer. Der Routine müssen genau dieselben Parameter wie beim Lesen eines Sektors übergeben werden.

Wenn das Schreiben beim ersten Versuch nicht erfolgreich ist, unternimmt die Routine selbständig weitere Schreibversuche.

Adresse: BE8Ch

Aufruf: e:= Laufwerksnummer (0=A, 1=B)
d:= gewünschte Spur
c:= gewünschte Sektornummer
hl:= Anfang des Speicherbereichs, aus dem Daten genommen werden

Rückgabe: cy:= Carry-Flag gesetzt, wenn Schreiben erfolgreich, Carry-Flag zurückgesetzt, wenn Fehler auftrat.

Beispiel:

```
lxi  d,0000h   ;Laufwerk/Spur
lxi  c,41h     ;physische Sektornummer
lxi  h,0100h   ;Pufferadresse
call BE8Ch
jnc  error
```

---> Der Speicherbereich von 0100h bis 02FFh wird in den ersten Sektor der Spur Nr. 0 einer 3"-CP/M-Systemdiskette im Laufwerk A kopiert.

Spur formatieren

Funktion: *Eine einzelne Spur einer Diskette neu formatieren.*

Das Format der Spur können Sie frei wählen: Sowohl die Anzahl und Länge der Sektoren, als auch die Numerierung können durch Parameter vorgewählt werden. So können Sie zum Beispiel eine bestimmte Spur umformatieren und eine (für den Normalanwender) kopiergeschützte Diskette

erzeugen. In der Regel werden Sie aber gleich alle Spuren einer Diskette zusammen formatieren und so den Extra-Aufruf des Programms FORMAT.COM ersetzen. Das könnte zum Beispiel in einem Anwenderprogramm von Nutzen sein.

Der Routine müssen Sie zunächst die Nummer des Laufwerks in dem sich die zu formatierende Diskette befindet und die Nummer der zu formatierenden Spur mitteilen.

Die speziellen Informationen über die einzelnen Sektoren müssen in einer Tabelle zusammengefaßt sein. Für jeden Sektor sind in der Tabelle 4 Bytes reserviert, bei 9 Sektoren hat sie also eine Länge von 36 Bytes. Die 4 Bytes des Sektors haben die Bedeutungen nach Bild 8-7.

```
Byte 0:   Spur, auf der sich der Sektor befindet
Byte 1:   Kennzeichen ob Vorder- oder Rückseite der Diskette
          (bei Einzelkopflaufwerken immer 0)
Byte 2:   Sektornummer (kann beliebig gewählt werden)
Byte 3:   Sektorgröße (0 für  128 Bytes
                       1 für  256 Bytes
                       2 für  512 Bytes
                       3 für 1024 Bytes usw.)
```

Bild 8-7: *Sektor-Informationen beim Formatieren*

Die Sektor-Informationen werden beim Formatieren auf der Diskette zusammen mit dem jeweiligen Sektor abgespeichert. Die Spurnummer im Byte 0 sollte deshalb der Nummer entsprechen, die auch beim Aufruf der Routine vorgegeben wurde. Bei einem späteren Leseversuch gibt es sonst Differenzen zwischen der gesuchten und der gefundenen Spurnummer. Die Spur mit der falschen Nummer ist dann nicht mehr zu lesen. Dasselbe gilt für die Kopfnummer im Byte 1 und die Sektorlänge im Byte 3. Die Sektornummer dürfen Sie jedoch beliebig wählen. Oft wird diese Möglichkeit verwendet, um verschiedene Diskettenformate einfach zu unterscheiden. Das ist auch auf den 3"-Disketten der Fall: Das CP/M-Format hat Nummern zwischen 41h und 49h und das Datenformat zwischen C1h und C9h.

Die Sektoren auf der Diskette erhalten der Reihe nach die Nummern, die in der Tabelle angegeben sind. Sind die Nummern in der Tabelle ungeordnet, befinden sich auch die Sektoren in genau derselben Weise 'ungeordnet' auf der Diskette. In der Regel ist das aber erwünscht: Befindet sich zwischen zwei logisch aufeinanderfolgenden Sektoren ein weiterer Sektor, hat der Computer Zeit den ersten Sektor zu verarbeiten, während der unwichtige unter dem Kopf vorbeistreicht. Wird dann der 2. Sektor benötigt, steht dieser fast sofort zur Verfügung und es braucht nicht eine volle Diskettenumdrehung gewartet zu werden.

Als Anzahl der Sektoren pro Spur und Länge der Sektoren nimmt die Formatierroutine automatisch die im Disk-Parameter-Block des Laufwerkes enthaltenen Werte. In der Regel werden also 9 Sektoren zu je 512 Bytes formatiert. Für Spezialzwecke können Sie das aber ändern. Sie müssen selbständig darauf achten, daß die Tabelle die richtige Länge besitzt und sinnvolle Werte enthält. Bei einem Fehler wird ohne Warnung Unsinn auf die Diskette geschrieben.

Bei einem Schreibfehler unternimmt die Routine automatisch eine gewisse Zahl von weiteren Versuchen, die Spur zu formatieren.

Adresse: BE8Fh

Aufruf: e:= Laufwerksnummer (0=A, 1=B)
d:= Spurnummer
hl:= Anfangsadresse der Format-Tabelle

Rückgabe: cy:= Carry-Flag gesetzt, wenn Formatieren erfolgreich, Carry-Flag zurückgesetzt, wenn Fehler auftrat.

Beispiel:

```
        lxi  d,0100h        ;Laufwerk A, Spur 1
        lxi  h,form         ;Format-Tabelle
        call BE8Fh
        jnc  error          ;Fehler beim formatieren
        jmp  ok
form:db  01h,0,41h,2        ;Spur 1, Seite, Sektor 41h, 512 Bytes
     db  01h,0,46h,2
     db  01h,0,42h,2
     db  01h,0,47h,2
     db  01h,0,43h,2
     db  01h,0,48h,2
     db  01h,0,44h,2
     db  01h,0,49h,2
     db  01h,0,45h,2
```

---> Die Spur 0 der Diskette im Laufwerk A wurde genau im CP/M-Format einer 3"-Diskette formatiert.

Schreib-/Lesekopf sofort positionieren

Funktion: *Eine frei wählbare Spur des Laufwerks wird sofort angesteuert, ohne daß wirklich ein Zugriff erfolgen muß.*

Bei der Steuerung durch das Standard-BIOS wird der Schreib-/Lesekopf erst über die gewählte Spur bewegt, wenn wirklich ein Schreib-/Lesezugriff erfolgt. Mit dieser Routine können Sie ihn jedoch sofort auf eine frei wählbare Spur setzen.

Nach dem Aufsuchen der Spur werden die beim Formatieren abgelegten Sektor-Marken untersucht. Entsprechen die darin gespeicherten Werte nicht der gesuchten Spurnummer, werden weitere Positionierungsversuche vorgenommen. Wird die Spur nach einer gewissen Zeit nicht gefunden, wird eine Fehlermeldung zurückgegeben.

Adresse: BE92h

Aufruf: e:= Laufwerksnummer (0=A, 1=B)
d:= Nummer der anzusteuernden Spur

Rückgabe: cy:= Carry-Flag gesetzt bei erfolgreicher Suche,
Carry-Flag zurückgesetzt, wenn Fehler auftrat.

Beispiel:

```
lxi  d,0200h        ;Laufwerk A, Spur 2
call BE92h
jnc  error
```

---> Der Schreib-/Lesekopf des Laufwerks A steht jetzt über der Spur 2 der Diskette.

Laufwerkstatus ermitteln

Funktion: *Bestimmte systeminterne Meldungen über Laufwerksoperationen werden durch diese Routine an das Anwenderprogramm weitergegeben.*

Das Ergebnis dieser Funktion ist eine 8-Bit-Zahl, in deren Bits mehrere Informationen über den Zustand des Laufwerkes verschlüsselt sind. Die meisten dieser Informationen sind aber nur von Bedeutung, wenn Sie sehr hardwarenah programmieren.

Wenn bei einer Operation ein Fehler auftrat, ist mindestens eines der Bits 3-7 des Ergebnisbytes gesetzt. Die Bits 0-2 enthalten Informationen über das angesprochene Laufwerk und können beliebige Werte annehmen. So ist ein einfacher Test möglich, ob ein Laufwerk verfügbar ist: Nur wenn die Bits 3-7 den Wert 0 haben, liegt eine Diskette im angesprochenen Laufwerk. Sie können also mit diesem Trick lästige BIOS- und BDOS-'Bad Sector'-Meldungen vermeiden. Das ist auch die einzige sinnvolle Anwendung dieser Routine in einem Anwenderprogramm.

Adresse: BE95h

Aufruf: a:= Laufwerksnummer (0=A, 1=B)

Rückgabe: a:= Bits 3-7 haben den Wert 0, wenn das Laufwerk verfügbar ist.
cy:= Carry-Flag gesetzt wenn Status korrekt geholt,
Carry-Flag zurückgesetzt, wenn Fehler auftrat.

Beispiel:

```
mvi  a,01h          ;Laufwerk B
call BE95h
jnc  error          ;Routine nicht korrekt ausgeführt
ani  F8h            ;unrelevante Bits ausblenden
jnz  nodrive        ;keine Diskette
```

---> Im Laufwerk B befindet sich mit Sicherheit eine Diskette.

Wiederholungen bei Schreib-/Lesefehler

Funktion: *Festlegen, wie oft eine Diskettenoperation bei einem Fehler automatisch wiederholt wird.*

Die Faustregel, einen mißglückten Schreib-/Lesevorgang 10mal wiederholen zu lassen, braucht nicht unbedingt eingehalten zu werden: Sowohl höhere als auch niedrigere Werte sind erlaubt. Mit Hilfe dieser Routine können Werte zwischen 1 und 255 eingestellt werden. In der Regel ist aber der Standardwert 10 optimal.

Adresse: BE98h

Aufruf: a:= Wiederholungswert

Rückgabe: nichts

Beispiel:

```
mvi  a,1        ;nur eine Wiederholung
call BE98h
```

---> Ein mißglückter Diskettenzugriff durch eine zufällige elektrische Störung wird nach diesem Aufruf nicht mehr automatisch wiederholt. Sie werden also danach gelegentlich 'Bad Sektor'-Meldungen unbekannter Ursache erhalten.

Externer BIOS-Zugang

Funktion: *Aufruf einer beliebigen CPC-Firmwareroutine aus dem CP/M*

Im Normalfall müssen Sie bei der Verwendung eines Unterprogramms das die ROMs des CPC-Computers benötigt, streng darauf achten, daß sich der

Stapel im 'zentralen RAM' befindet. Nur so vermeiden Sie es, daß sich die Routine selbst den 'Boden unter den Füßen' wegzieht. Ein ähnliches Problem haben Sie, wenn sie etwa aus dem Speicherbereich von 0 bis 3FFFh eine Routine im parallel liegenden ROM ohne Umweg über die Firmware-Sprungtabelle aufrufen wollen.

Dauerhafte ROM-Umschaltungen oder das Wählen eines anderen Bildschirmmodes sind auf diese Weise überhaupt nicht möglich: Bei jedem Interrupt wird die alte Konfiguration wiederhergestellt, so daß es zu undefinierbaren Resultaten kommen kann.

In solchen Fällen können Sie die Routine 'Externer BIOS-Zugang' verwenden, die alle Schwierigkeiten selbständig beseitigt: Dieser übergeben Sie die Adresse des aufzurufenden Programms. Die BIOS-Eingangsroutine legt dann für die Dauer der Programmausführung selbständig einen lokalen Stapel an. Wenn in dem ausgeführten Programm eine ROM- oder Bildschirmmodeumschaltung ausgeführt wurde, bleibt sie auch danach erhalten. Beliebige CPC-Firmware-Aufrufe sind so problemlos möglich.

In der Praxis ist die BIOS-Zugangs-Routine ähnlich handzuhaben wie die RST-Befehle im normalen CPC-Betriebssystem: Sie schreiben in die CPU-Register wie gewöhnlich die Übergabeparameter und rufen dann für jede beliebige Firmwareroutine das Unterprogramm an der Adresse BE9Bh auf. Die eigentliche Adresse der Firmwareroutine steht in den 2 Bytes, die dem 'call BE9Bh' unmittelbar folgen. Durch die besondere Behandlungsweise des Stapels werden diese beiden Bytes bei der Rückkehr aus dem Unterprogramm automatisch übersprungen.

Adresse: BE9Bh

Aufruf & Rückgabe: nichts

Beispiel:

```
lxi  d,0000h        ;x-Koordinate: 0 (Startpunkt)
lxi  h,0000h        ;y-Koordinate: 0
call BE9Bh          ;externer BIOS-Zugang
dw   BBC0h          ;Adresse von 'GRA MOVE ABSOLUTE'
lxi  d,0279h        ;x-Koordinate: 639 (Endpunkt)
lxi  h,018Fh        ;y-Koordinate: 399
call BE9Bh          ;externer BIOS-Zugang
dw   BBF6h          ;Adresse von 'GRA LINE ABSOLUTE'
```

---> Unter Verwendung der normalen CPC-Firmwareeinsprünge wird unter CP/M eine diagonale Linie über den Bildschirm gezogen. Die Lage des Stackpointers und der Bildschirmmode sind dabei völlig unwichtig.

Fast/Slow-Modus wählen

Funktion: *Restauration des BC'-Registers vor jedem CPC-Firmware-einsprung ein-/ausschalten*

Im Basic-Betriebssystem der CPC-Computer darf der alternative Registersatz (AF', BC', DE', HL') nicht verwendet werden: Diese Register enthalten wichtige Informationen zum Bankswitching und werden von der Interruptroutine ständig benötigt. Im CP/M-Betriebssystem aber werden alle BIOS-internen Firmware-Aufrufe über die Routine zum 'externen BIOS-Zugang' geleitet. Das gilt auch für den CP/M-Interrupt.

Von der BIOS-Zugangs-Routine gibt es im ROM 2 Versionen: Die eine verlangt dieselbe Einschränkung wie unter BASIC: das BC'-Register darf nicht angetastet werden. Im Gegensatz dazu restauriert die andere das BC'-Register vor jedem Firmwareaufruf. Zusätzlich werden alle 8 'gestrichenen'-Register und das IY-Register auf den Stapel gerettet. Diese werden nach dem Firmware-Aufruf restauriert, so daß jede beliebige Firmwareroutine nur noch die Standard-Register dauerhaft verändern kann. Auf diese Weise dürfen alle CP/M-Programme jedes einzelne CPU-Register uneingeschränkt verwenden. Diese Eigenschaft wird von vielen komplexeren Programmen benötigt. Turbo-Pascal ist ein Beispiel dafür.

Dieser Komfort hat jedoch den Nachteil, daß die Firmwareroutinen wesentlich langsamer werden. Besonders bei kurzen Routinen, bei denen der Zeitbedarf zum Registerumschalten in der Größenordnung der eigentlichen Ausführungszeit liegt, wirkt sich das im Verhältnis sehr stark aus. Im Controller der 3"-Floppydisk wurde deshalb dem Anwender die Möglichkeit gegeben sich zu entscheiden, ob er lieber schnelle Firmwareroutinen oder einen uneingeschränkten Registersatz verwendet. Standardmäßig ist der SLOW-Modus eingeschaltet. Mit Hilfe von SETUP.COM oder der speziellen Umschaltroutine an der Adresse BE9Eh kann diese Einstellung geändert werden.

Leider bietet der Controller der Vortex-Floppydisk diese Umschaltmöglichkeit nicht: Der Computer befindet sich hier immer im SLOW-Modus. In den dadurch frei gewordenen Eintrag in der erweiterten BIOS-Sprungtabelle wurde deshalb eine zusätzliche Routine eingesetzt: Ein Sprung zur Adresse BE9Eh führt beim Vortex-System zum sofortigen Abschalten der Diskettenmotoren, was beim Schneider-Laufwerk unmöglich ist. Das ist etwa dann nützlich, wenn eine Diskette sofort nach dem Zugriff gewechselt werden soll und nicht erst nach einer 5 Sekunden langen Wartezeit.

Adresse: BE9Eh

Aufruf Schneider:
a:= 0 zum Wählen des SLOW-Modus,
255 zum Wählen des FAST-Modus.
Aufruf Vortex:
nichts (zum Abschalten des Diskettenmotors)

Rückgabe: nichts

Beispiel:

```
mvi  a,FFh
call BE9Eh
```

---> Bei einer Schneider-Diskettenstation laufen jetzt unter anderem die Bildschirmausgaben schneller ab, dafür darf aber der alternative Registersatz nicht mehr verwendet werden.

SIO initialisieren

Funktion: *Initialisieren des seriellen Ein-/Ausgabeinterfaces*

Sowohl der Schneider- als auch der Vortex-Floppydiskcontroller sind für eine zusätzliche serielle Schnittstelle vorbereitet. Ein Aufruf dieser Routine initialisiert die Schnittstelle mit bestimmten Standardwerten.

Bei der Vortex-Floppydisk gibt es diese Routine nicht. Der entsprechende Platz in der Sprungtablle ist aber reserviert, so daß sie beim Erwerb einer seriellen Schnittstelle noch eingebaut werden kann.

Adresse: BEA1h

Aufruf: nichts

Rückgabe: nichts

Beispiel:

```
call BEA1h     (nur für Schneider-Controller)
```

---> Die serielle Ein-/Ausgabeschnittstelle wurde initialisiert.

Autostart-Kommando initialisieren

Funktion: *Eine Gruppe von Zeichen wird so im Speicher abgelegt, daß das BIOS im folgenden statt Eingaben von der Tastatur diese Zeichen zurückgibt.*

Diese Routine wird auch im Kaltstartlader gleich nach dem Einschalten des Rechners aufgerufen und verursacht den automatischen Start eines CP/M-

Programms. Sie können die Autostart-Routine aber auch noch später verwenden. Mit deren Hilfe können Sie etwa auf einfache Weise CP/M-Programme sich gegenseitig aufrufen lassen. Dieser Trick ist aber natürlich nur auf einem CPC-Computer zulässig. In Standard-CP/M-Programmen verwendet man zu diesem Zweck programmgesteuert erstellte $$$.SUB-Dateien.

Im Gegensatz zu SUBMIT-/XSUB-Programmen können mit dieser Methode auch nicht gepufferte Eingaben automatisch erledigt werden, also nicht nur mit der BDOS-Funktion Nr. 9, sondern auch mit den BDOS-Funktionen Nr. 1 und 6 angeforderte Eingaben.

Eine Tastatureingabe hat vor dem Autostart-String Vorrang. Danach wird im Normalfall das Einlesen des Autostart-Strings fortgesetzt. Tastatureingaben können so mitten im String stehen, so daß in der Eingabezeile ein unsinniger Programmname steht. Deshalb kann der Autostart so eingestellt werden, daß die Zeichenkette bei der ersten Tastatureingabe gelöscht wird.

Der Routine zum Initialisieren des Autostarts wird mitgeteilt, ob die Zeichenkette bei einer Eingabe zu löschen ist oder nicht. Zusätzlich muß ihr die Anfangsadresse einer Tabelle übergeben werden. Das erste Byte der Tabelle enthält die Länge des Autostart-Strings und ab dem zweiten Byte stehen die einzelnen Zeichen des Strings.

Adresse: BEA4h

Aufruf: a:= 0, wenn der String bei einer Tasteneingabe gelöscht werden soll,
ungleich 0, wenn der String nicht gelöscht werden soll.
hl:= Anfangsadresse der Zeichenketten-Tabelle

Rückgabe: nichts

Beispiel:

```
        mvi  a,FFh          ;Unterbrechnung nicht möglich
        lxi  h,tabl
        call BEA4h          ;Autostart-String initialisieren
        jmp  0000h          ;Warmstart
tabl:db  4,'DIR',13         ;Länge, Text 'DIR'+ Wagenrücklauf
```

---> Nach dem Programmende durch den Warmstart wird automatisch das Disketten-Inhaltsverzeichnis ausgegeben.

8.3.2 Die serielle Ein-/Ausgabeschnittstelle

Sowohl der Schneider- als auch der Vortex-Floppydiskcontroller unterstützen ein Zusatzgerät zur seriellen Ein-/Ausgabe, sofern es vorhanden ist. Dazu gibt es im RAM eine weitere Sprungtabelle mit 8 Einträgen. Diese 'Kommunikationsschnittstelle' liegt im Bereich von BEA7h bis BEBEh.

Bei der Anforderung einer Datenübertragung über einen der Ein-/Ausgabekanäle mit den Standard-BIOS-Routinen wird zunächst anhand des Input/Output-Bytes untersucht, welches reale Gerät angesprochen werden soll. Wird schließlich eine Bildschirm-/Tastatur- oder Drucker-Ein-/Ausgabe benötigt, übergibt die BIOS-Routine die Bearbeitung an die entsprechenden CPC-Firmware-Routine. Anders ist es jedoch bei den Aufrufen, die schließlich bei einer Ein-/Ausgabe über die seriellen Schnittstellen enden: In so einem Fall wird nicht irgendeine undefinierbare Routine im BIOS-ROM aufgerufen, sondern einer der RAM-Vektoren im Bereich von BEA7h bis BEBEh. Damit sollte ursprünglich das BIOS für eine eventuell geänderte Hardware vorbereitet werden.

Diese RAM-Vektoren werden ohne Ausnahme von jeder einzelnen BIOS-gesteuerten seriellen Ein-/Ausgabe aufgerufen. Durch eine Veränderung eines oder mehrere RAM-Vektoren können Sie eine 'Umleitung' auf eine eigene Ein-/Ausgaberoutine einbauen. Diese ist dann völlig standardgemäß ins CP/M-Betriebssystem eingebunden. Dem aufrufenden Programm ist es völlig gleichgültig, ob diese eigene Routine ein serielles Ein-/Ausgabegerät oder irgendein anderes Gerät anspricht. Sie können diese Aufrufe also ohne weiteres durch zusätzliche Bildschirmausgaben oder Kassetten Ein-/Ausgaben ersetzen.

Beachten Sie, daß die schon besprochene Sprungtabelle im Bereich von BE80h bis BEA6h sozusagen 'umgekehrt' wie die Sprungtabelle im Bereich von BEA7h bis BEBEh aufgebaut ist: Die erste Tabelle wird von Anwenderprogrammen im RAM aufgerufen und leitet die Bearbeitung an das BIOS im ROM weiter. Eine Änderung der Sprungziele ist hier nicht sinnvoll. Die zweite dagegen wird vom BIOS aufgerufen und leitet die Bearbeitung an eine Routine im RAM oder in einem anderen ROM weiter. Hier ist es gleichgültig, ob diese externe Routine von Ihnen selbst oder von der Firma Schneider entworfen wurde.

Die Funktionen der Kommunikationsschnittstelle

Vom BIOS unterstützt werden zwei serielle Ein-/Ausgabegeräte: SIO 1 und SIO 2. Zur Steuerung einer jeden sind 4 Routinen nötig, deren Bedeutungen unten erklärt sind. Die Kommunikationsschnittstelle hat also insgesamt 8 Einträge.

Eingabestatus der SIO

Erhält vom BIOS:
nichts
Liefert ans BIOS:
a:= 0, wenn kein Zeichen bereit liegt,
255, wenn ein Zeichen zur Eingabe bereit liegt.

Zeichen von SIO einlesen

Erhält vom BIOS:
nichts
Liefert ans BIOS:
a:= Code des gelesenen Zeichens

Ausgabestatus der SIO

Erhält vom BIOS:
nichts
Liefert ans BIOS:
a:= 0, wenn das serielle Ausgabegerät beschäftigt ist,
255, wenn das seriell Ausgabegerät bereit ist, ein Zeichen zu übernehmen.

Zeichen an SIO ausgeben

Erhält vom BIOS:
c:= Code des auszugebenden Zeichens
Liefert ans BIOS:
nichts

Die Lage der SIO-Einsprünge

Im Allgemeinen sind die erweiterten BIOS-Einsprünge der Vortex-Floppydisk vollständig kompatibel zu denen der Schneider-Floppydisk. Unverständlicherweise wurde aber hier eine Ausnahme gemacht: In der Sprungtabelle von BEA7h bis BEBEh sind zwar bei beiden Systemen dieselben Einsprünge vorhanden, aber jeweils in einer anderen Reihenfolge. Obwohl die eigentlichen Datenübertragungsroutinen bei beiden Diskcontrollern gleich aufgebaut sein können, muß die Einbindung der Routinen ins Betriebssystem auf verschiedene Weise erfolgen. Die genaue Lage der Einsprünge zeigt Bild 8-8.

Die Sprungvektoren im Schneider-BIOS zeigen schon auf vorhandene serielle Ein-/Ausgaberoutinen im ROM. Natürlich können diese trotzdem ge-

ändert werden. Das ist aber nur nötig, wenn wirklich eine anderer Hardware als die vorgesehene angeschlossen wird. Im Vortex-BIOS zeigen die Vektoren auf 'leere' Routinen, die ohne Schaden anzurichten die Kontrolle an das aufrufende Programm zurückgeben. Beim Zeicheneinlesen wird zum Beispiel immer der Code 1Ah, also die Endemarkierung einer Übertragung, zurückgegeben. Wenn ein zusätzliches Gerät angeschlossen wird, müssen diese Einsprünge also in jedem Fall geändert werden. Immerhin gibt es aber keine hardwaremäßigen Inkompatibilitäten.

Adresse	Bedeutung im Schneider-BIOS	Bedeutung im Vortex-BIOS
BEA7h	Eingabestatus SIO 1	Eingabestatus SIO 1
BEAAh	Zeichen holen SIO 1	Eingabestatus SIO 2
BEADh	Ausgabestatus SIO 1	Ausgabestatus SIO 1
BEB0h	Zeichen senden SIO 1	Ausgabestatus SIO 2
BEB3h	Eingabestatus SIO 2	Zeichen holen SIO 1
BEB6h	Zeichen holen SIO 2	Zeichen holen SIO 2
BEB9h	Ausgabestatus SIO 2	Zeichen senden SIO 1
BEBCh	Zeichen senden SIO 2	Zeichen senden SIO 2

Bild 8-8: *Die Kommunikationsschnittstelle bei Verwendung einer Vortex- und einer Schneider-Floppydisk*

8.4 Tips zum Umgang mit dem BIOS

Der Stapel während BIOS-Aufrufen

Die Standard-BIOS-Routinen legen sich selbständig einen lokalen Stapel an. Erst beim Rücksprung wird wieder der ursprüngliche Stapel eingeschaltet. Aus der Sicht eines Anwenderprogrammen belegt ein BIOS-Aufruf also nur die 2 Bytes für den unmittelbaren Aufruf. Beim Reservieren von Speicherbereich für den Anwenderprogrammstapel brauchen Sie sich also nur um den Bereich kümmern, der von Ihren eigenen Unterroutinen benötigt wird.

Anders ist es jedoch bei den zusätzlichen BIOS-Routinen, die es nur in den CPC-Computern gibt: Bei einem Sprung in den Bereich von BE80h bis BEA6h wird kein lokaler Stapel angelegt. Hier müssen Sie selbst dafür sorgen, daß der Stapel in Ihrem Anwenderprogramm eine genügend große 'Sicherheitsreserve' bereit hält. Ein Aufruf einer dieser Routinen über den 'externen BIOS-Eingang' um automatisch einen lokalen Stapel zu erhalten ist nicht sinnvoll: Der externe BIOS-Eingang ist ja selbst eine der zusätzlichen Routinen. Natürlich dürfen Sie aber in eigener Regie im TPA-Bereich des Speichers soviele lokale Stapel anlegen wie Sie wollen.

Die Kommunikationsschnittstelle im Bereich von BEA7h bis BEBEh wird nur von den Routinen im Standard-BIOS aufgerufen. Deshalb steht der seriellen Ein-/Ausgaberoutine schon von vornherein ein genügend großer lokaler Stapel zur Verfügung. Beim Erstellen einer eigenen Ein-/Ausgaberoutine brauchen Sie sich also um die Lage des Stackpointers nicht zu kümmern.

Unter CP/M darf eine Routine einer tieferen Schicht des Betriebssystems niemals eine Routine einer höheren Schicht aufrufen: Eine selbstentworfene Standard-BIOS-Routine darf also keine BDOS-Routinen aufrufen. Die Routinen der Kommunikationsschnittstelle sind den Standard-BIOS-Routinen untergeordnet. Wenn also eine neue serielle Ein-/Ausgaberoutine jetzt ein Zeichen auf den Bildschirm ausgeben soll, dürfen Sie dazu nicht die Standard-BIOS-Zeichenausgabe oder gar die entsprechende BDOS-Funktion aufrufen: Erlaubt ist an dieser Stelle nur noch ein Aufruf der auch im Basic-System vorhandenen Firmware-Sprungtabelle des CPC-ROMs.

Niemals dürfen Sie auf die Idee verfallen, etwa einen Drucker-Spooler nach folgendem Prinzip aufzubauen: Bei jedem Aufruf der BIOS-Konsoleneingaberoutine, bei der noch kein Zeichen zur Verfügung steht hat das BIOS nichts besseres zu tun, als auf einen Tastendruck zu warten. Diese

Zeit könnte dazu genutzt werden, 'zwischendurch' ein paar Zeichen auf den Drucker auszugeben. Keinesfalls darf die BIOS-Routine dazu aber das BDOS aufrufen und die auszugebenden Zeichen mit Hilfe der normalen Dateioperationen aus einer Datei nachladen. Das würde zum vollständigen Systemabsturz führen.

Mischen von BIOS und BDOS-Routinen

Wenn Sie BIOS- und BDOS-Aufrufe gemischt verwenden wollen, kann das zu unerwünschten Resultaten führen. Harmlos sind diese Resultate nur bei den Bildschirmausgaben: Im schlimmsten Fall kann die Ausgabe etwas in Unordnung geraten. Dann müssen Sie eben mit einer geringfügig geänderten Ausgaberoutine einen neuen Versuch starten.

Gefährlich wird es jedoch, wenn Sie Diskettenbefehle des BIOS und des BDOS abwechselnd verwenden: Etwa ein Spurwechsel durch die Anwendung der entsprechenden BIOS-Routine wird vom BDOS nicht registriert. Beim nächsten geordneten BDOS-Dateizugriff glaubt das BDOS, daß sich der Schreib-/Lesekopf noch über der alten Spur befindet und führt keine Neueinstellung durch. Als Folge werden Ihre Daten in einen falschen Bereich der Diskette geschrieben. In der Regel führt das zur Zerstörung einer wichtigen Datei.

Bei der Verwendung von BIOS-Diskettenbefehlen ist die erneute Verwendung von BDOS-Dateizugriffen nur nach einem Disk-Reset erlaubt. Ein Disk-Reset ist aber wiederum nur erlaubt, wenn alle Dateien ordnungsgemäß geschlossen sind. In solchen Fällen sollten Sie bei der Programmierung also besondere Vorsicht walten lassen und das Programm zuerst auf einer leeren Diskette testen.

Ein verbotener 'BIOS-Einsprung'

Im Normalfall (bei nicht verschobenem CP/M) befindet sich die Standard-BIOS-Tabelle im Bereich von AD00h bis AD32h. Bei Verwendung des Schneider-Diskcontrollers befindet sich an der Adresse AD33h ein weiterer Sprungbefehl. Es hat den Anschein, als ob es sich dabei um einem weiteren, bisher noch nicht erwähnten BIOS-Einsprung handeln würde. Das ist jedoch nicht der Fall!

Bei der Standard-CP/M-Installation befindet er sich nur 'zufällig' direkt hinter der BIOS-Sprungtabelle. Bei einer Verschiebung des CP/M-Systems (siehe nächstes Kapitel) wird dieser Sprungbefehl nicht mit verschoben. Durch ein Anwenderprogramm ist er dann nicht mehr auffindbar. Bei einer Vortex-Floppydisk gibt es diesen Sprungbefehl zwar auch, dort befindet er sich aber von vornherein an einer anderen Stelle.

Dieser Sprungbefehl ist nur zur internen Verwendung durch das BIOS selbst bestimmt. In Anwenderprogrammen kann die dadurch angesprungene Routine vollständig durch andere Betriebssystemroutinen ersetzt werden: Im wesentlichen entspricht er einem 'call BE9Bh'-Befehl mit nachfolgendem RET.

Restaurierung der BIOS-Tabelle

Im Gegensatz zum Standard-CP/M wird im CPC 464 die BIOS-Sprungtabelle direkt hinter dem BDOS nach jedem Warmstart restauriert. Das Ersetzen einer bestimmten BIOS-Routine durch eine eigene und diese durch einen geänderten Sprung in der Tabelle ins System einzubinden funktioniert also nur bis zum nächsten Warmstart.

Eine dauerhafte Installierung können Sie erreichen, indem Sie zusätzlich die Warmstartroutine so abändern, daß die BIOS-Tabelle nicht mehr restauriert wird. Dazu kopieren Sie die gewöhnliche Warmstartroutine in einen freien Bereich im RAM, streichen aber den Abschnitt, in dem die originale BIOS-Tabelle aus dem ROM ins RAM kopiert wird.

Der normale BIOS-Einsprung zur Warmstartroutine führt bei der Schneider-Floppydisk zur Adresse C2BEh. An dieser Stelle steht 'call AD33h'. Dieser Call-Befehl entspricht wie oben erwähnt im wesentlichen einem 'call BE8Bh'-Befehl: Es wird ein lokaler Stapel angelegt und der alternative Registersatz restauriert. An der Stelle C2C1h finden Sie also die Adresse der eigentlichen Warmstartroutine: C22Bh. Im Bereich von C279h bis C27Fh findet schließlich mit Hilfe eines LDIR-Befehles die Restaurierung der BIOS-Sprungtabelle statt, die Sie streichen sollen.

Bei der Vortex-Floppydisk funktioniert das Ganze genau gleich. Nur die Adressen lauten natürlich anders.

Eine Anwendung der zusätzlichen BIOS-Einsprünge

Mit Hilfe der Routine zum Anlegen einer Autostart-Zeichenkette können Sie sehr einfach eine Art 'Mini-Submit' realisieren. Das Listing dazu zeigt Bild 8-9. Es ist so kurz, daß Sie es am besten nicht mit dem Assembler, sondern mit Hilfe des DDT.COM-Programms eingeben und nach ^C mit 'SAVE 1 RUN.COM' abspeichern.

Nach dem Aufruf befindet im Speicherbereich ab 0080h eine Kopie der Zeichenkette, die nach dem Namen RUN als Parameter eingegeben wurde. Nach dem Anlegen eines lokalen Stapels werden im ersten Arbeitschritt alle enthaltenen Ausrufezeichen durch Wagenrücklaufzeichen 0Dh ersetzt. Zusätzlich wird am Ende der Eingabe ein weiterer Wagenrücklauf angehängt.

Das Ergebnis wird dann mit Hilfe des Aufrufs 'call BEA4h' als Autostartstring ins System eingebunden und das Programm korrekt beendet.

```
0100  LXI  H,0000        0116  MVI  M,0D
0103  DAD  SP            0118  DCR  B
0104  SHLD 01FE          0119  JNZ  0111
0107  LXI  SP,01FE       011C  INX  H
010A  LXI  H,0080        011D  MVI  M,0D
010D  MVI  A,21          011F  LXI  H,0080
010F  MOV  B,M           0122  MVI  A,FF
0110  INR  M             0124  CALL BEA4
0111  INX  H             0127  LHLD 01FE
0112  CMP  M             012A  SPHL
0113  JNZ  0118          012B  RET
```

Bild 8-9: *Das Programm RUN.COM*

Nach dem Programmende werden die dem RUN-Programm übergebenen Parameter nachwirkend als gewöhnliche Eingaben betrachtet. Ausrufezeichen gelten dabei - wie beim Assembler - als logische Zeilenenden, die im Autostartstring wirklich voneinander getrennt werden. Durch die Angabe von

```
RUN STAT!DIR *.COM!DIR *.ASM
```

werden - ohne weitere Eingabe - die Befehle 'STAT', 'DIR *.COM' und 'DIR *.ASM' nacheinander abgearbeitet.

```
RUN ASM PROGRAMM.AAZ!LOAD PROGRAMM!ERA PROGRAMM.HEX
```

assembliert das Programm PROGRAMM.ASM, wandelt es in eine COM-Datei und löscht die unerwünschte Hex-Datei in einem Zug.

RUN.COM ist also trotz der Kürze ein vollständiges Submit-Programm mit integriertem Editor. Nicht möglich sind auf diese Weise in diesem Submit-Programm nur Variablen wie $1, $2 und so weiter und der Aufruf des XSUB.COM-Programmes. Dafür lassen sich auch einzelne Tastaturabfragen, die nicht mit der BDOS-Funktion 9 angefordert wurden, automatisch abarbeiten. So können Sie zum Beispiel das FORMAT- oder SETUP-Programm mit einer einzigen Eingabe bedienen. RUN.COM ist auch wesentlich schneller als das gewöhnliche SUBMIT.

9 Für Profis: Verändern des Betriebssystems

9.1 Reservieren von Speicherbereichen

Bisher kannten Sie noch keine Möglichkeit, einen Speicherbereich absolut vor dem Zugriff des gewöhnlichen Betriebssystems zu schützen und trotzdem alle CP/M-Möglichkeiten uneingeschränkt weiter zu verwenden. Tricks wie das Verändern des Pointers auf den BDOS-Einsprung sind nur mit Vorsicht zu genießen: Dadurch funktioniert zum Beispiel der XSUB-Befehl nicht mehr ordnungsgemäß. Zum dauerhaften Einfügen eigener Ergänzungen wäre aber gerade das nötig.

Glücklicherweise gibt es aber eine Möglichkeit, das gesamte BDOS im Speicher zu verschieben. Dadurch wird im darüberliegenden Bereich Speicherplatz frei.

Der Befehl MOVCPM.COM

In den CPC-Computern ist das BDOS standardmäßig ab der Adresse 9F06h und der Console-Command-Processor CCP ab der Adresse 9700h lauffähig. Mit Hilfe des MOVCPM-Befehls können Sie das aber ändern. Bei jedem danach folgenden Warmstart lädt sich das BDOS dann automatisch in den passenden Speicherbereich. Dem MOVCPM-Befehl können - getrennt durch Leerzeichen - 2 Parameter übergeben werden.

Der erste Parameter gibt die Speichergröße in 256-Byte-Einheiten an, für die das neue BDOS konfiguriert wird. Leider beginnt aber die Zählung nicht mit der Nummer 0, sondern mit einem bestimmten Versatz. Für das Standard-BDOS, das an der Adresse 9F06 beginnt, muß hier der Wert '179' übergeben werden. Jede Verminderung um den Wert 1 läßt das BDOS an einer 256 Bytes niedrigeren Adresse beginnen, jede Erhöhung an einer um 256 Bytes höheren Adresse. Eine feinere Abstufung ist leider nicht möglich. 'MOVCPM 178 *' konfiguriert also das BDOS so, daß es ab der Adresse 9E06h lauffähig ist.

Zusammen mit dem BIOS der Schneider-Floppydisk ist 179 der maximal erlaubte Wert. Dasselbe gilt für die Vortex-Floppydisk unter VDOS 1.0. Unter VDOS 2.0 ist hier der maximal erlaubte Wert sogar nur 178.

In einem Standard-CP/M-System darf als erster Parameter ein Sternchen '*' angegeben werden. In diesem Fall sucht sich das MOVCPM selbständig die

größte existierende RAM-Adresse und Konfiguriert das BDOS dazu passend. In den CPC Computern ist ein Teil des RAM-Bereiches aber für die CPC-Firmware reserviert. Das MOVCPM-Programm erkennt das leider nicht, so daß ein so konfiguriertes BDOS diese Bereiche überschreiben würde. Das führt natürlich zu einem Systemsabsturz.

Für den 2. Parameter gibt es nur 2 Möglichkeiten: Entweder kann er fehlen oder er ist ein Stern '*'. Wenn der 2. Parameter fehlt, wird das neue Betriebssystem sofort gestartet ohne daß irgendwelche Vorbereitungen getroffen werden, um es später auf der Diskette abzuspeichern. Bei einem Eingabefehler haben Sie also keine Korrekturmöglichkeit mehr und müssen den Rechner ausschalten und wieder vorn vorn beginnen. In der Regel sollten Sie diese Variante also vermeiden.

Ist als 2. Parameter ein '*' vorhanden, werden das neue BDOS und der neue CCP zunächst nur im RAM zwischengespeichert und nicht gestartet. Unabhängig von der Adresse, an der das neue Betriebssystem ablauffähig ist, beginnt es in diesem Fall an der Adresse 0980h. Durch 'SAVE 34 <Name>' können Sie es in einer Datei zwischenspeichern und für eine spätere Verwendung aufbewahren. Das MOVCPM-Programm macht dazu selbständig einen Namensvorschlag, in dem die CP/M-Größe enthalten ist: Bei der durch 'MOVCPM 179 *' erzeugten CP/M-Version zum Beispiel CPM44.COM.

Durch 'DDT CPM44.COM' können Sie die neue CP/M-Version untersuchen und eventuell bestimmte Teile verändern. Der CCP beginnt an der Adresse 0980h und das BDOS an der Adresse 1186h.

Das MOVCPM-Programm verändert nicht die Betriebssystem-Version, die sich auf der Diskette befindet, sondern enthält eine eigene Kopie des BDOS und des CCPs. Das MOVCPM-Programm muß also an Ihre CP/M-Version angepaßt sein. Im RAM befindet sich in den 6 Bytes unmittelbar vor dem Anfang des aktiven BDOS (beim Standard-CP/M in den CPC-Computern also im Bereich 9F00h-9F05h) die codierte CP/M-Versionsnummer. Nach dem Aufruf überprüft das MOVCPM, ob diese Bytes mit den im Programm intern gespeicherten übereinstimmen. Wenn keine Übereinstimmung festgestellt wurde, wird die Fehlermeldung 'Synchronisation Error' ausgegeben. In der Regel sollte diese Fehlermeldung bei Ihnen aber nicht auftreten.

Abspeichern der neuen CP/M-Version auf den Systemspuren

Wenn die neue CP/M-Version nach jedem Warmstart automatisch verwendet werden soll, muß sie sich auf den Systemspuren befinden. Das können Sie mit dem Befehl SYSGEN.COM durchführen.

Dabei können Sie entscheiden, ob Sie ein Betriebssystem von den Systemspuren einer anderen Diskette auf die Systemspuren der Zieldiskette kopieren, oder ob Sie ein neu erstelltes Betriebssystem im RAM des Computers verwenden wollen. Beim kopieren von einer Diskette müssen Sie bei der ersten Eingabeanforderung den Diskettennamen an, von der das Betriebssystem genommen werden soll und befolgen die danach erscheinenden Befehle.

Wenn Sie bei der ersten Eingabeanforderung ENTER eintippen, wird ein Betriebssystem aus dem RAM auf den Systemspuren abgespeichert. SYSGEN erwartet es dazu im Speicherbereich ab der Adresse 0980h. Diese SYSGEN-Variante können Sie also nur unmittelbar nach einem MOVCPM-Befehl oder nach einem 'DDT CPMnn.COM'-Befehl verwenden.

Nach der SYSGEN-Variante, die ein Betriebssystem von einer Diskette auf eine andere kopierte, befindet sich das kopierte Betriebssystem wie nach einem MOVCPM-Befehl im Speicher. Durch anschließendes 'SAVE 34 CPMnn.COM' und 'DDT CPMnn.COM' können Sie also auch ein schon vorhandenes System bearbeiten.

9.2 Verändern von CCP und BDOS

Wenn sich das BDOS und der CCP nach MOVCPM im Speicher befinden, könnten Sie bequem Änderungen anbringen. In der Regel ist eine Veränderung des BDOS aber nicht sinnvoll: Sie werden wohl kaum einen bisher unentdeckten Fehler finden und diesen auch noch verbessern können.

Anders ist es aber mit dem CCP. Dieser besitzt eine speziellen Schnittstelle, in den eine Änderung eingefügt werden darf: der automatisch ausgeführte Befehl. Nach einem MOVCPM-Befehl hat die Schnittstelle den Aufbau nach Bild 9-1.

Adresse	Länge	Bedeutung
0980h	3	CCP-Einsprung mit Beachtung des Befehlspuffers
0983h	3	CCP-Einsprung mit Löschung des Befehlspuffers
0986h	1	maximale Befehlspufferlänge*(127)
0987h	1	Länge des aktuellen Befehls (0)
0988h	16	1. Teil des Befehlspuffers: enthält Leerzeichen
0998h	112	2. Teil des Befehlspuffers: enthält Copyright-Notiz von Digital Research
0A07h		nicht veränderbarer Bereich des CCP

Bild 9-1: *Änderungsbereich des CCP*

Der Bereich 0988h bis 0A07h steht zum Einfügen eines CP/M-Programmaufrufes zur Verfügung. Die darin enthaltene Copyright-Notiz von Digital Research darf ohne Bedenken überschrieben werden. Die Speicherstelle 0987h enthält die Anzahl der gültigen Bytes des Programmaufrufes. Der Wert 0 bedeutet, daß kein Aufruf vorliegt. Ein hier eingetragener Befehl darf nur Großbuchstaben enthalten - sonst wird er nicht richtig interpretiert.

Wird der CCP über die erste Einsprungadresse aufgerufen, führt er als allererstes den im Befehlspuffer enthaltenen Befehl aus. Auf dem Bildschirm erscheinen dabei keinerlei Meldungen über den Aufruf. Ein Programm, das nicht selbst eine Meldung ausgibt oder eine Eingabe anfordert, läuft so vom Benutzer völlig unbemerkt ab. Wenn kein Befehl vorliegt, untersucht der CCP ob sich im Laufwerk A eine aktive Submit-Datei befindet. Erst wenn auch keine Submit-Datei vorhanden ist wird die Meldung 'A>' oder 'B>' ausgegeben und eine Tastatureingabe verlangt.

Auf diese Weise können Sie einen Autostart eines Programms installieren, der völlig unabhängig vom Computertyp ist und nicht nur (wie der 'call BEA4h'-Befehl) nur auf den Schneider-Computern funktioniert.

Wenn der CCP über die 2. Einsprungadresse aufgerufen wird, wird der automatische Befehl nicht beachtet.

In üblichen CP/M-Systemen ruft die BIOS-Kaltstartroutine den CCP in der nichtlöschenden Version auf und die BIOS-Warmstartroutine die löschende Version des CCP. Dadurch wird ein eingetragener Programmaufruf nur ein einziges Mal, gleich nach dem Einschalten aufgerufen. Damit kann etwa ein Programm gestartet werden, das das Computersystem zusätzlich initialisiert. Denkbar wäre auch - wie beim CPC 6128 unter CP/M 3.1 - ein Befehl 'SUBMIT PROFILE', so daß gleich eine ganze Programmliste in einer PROFILE.SUB-Datei abgearbeitet wird.

Unverständlicherweise wird im BIOS des CPC 464/664 auch beim Warmstart der CCP in der nichtlöschenden Variante aufgerufen. Ein im CCP eingetragenes Programm wird also hier nach jedem Warmstart ausgeführt. Der praktische Einsatz eines automatischen Befehls ist damit auf diese Art unmöglich: Jedes automatisch gestartete Programm, das mit einem Warmstart endet wird ja sofort noch einmal aufgerufen. Der Computer wird so völlig blockiert.

Durch einen Trick läßt sich aber doch ein automatischer Befehl in den CCP einbauen: Das wichtige ist nur, daß der automatische Befehl nicht auf den Systemspuren der Diskette enthalten ist. Wenn sich der CCP im Speicher befindet, kann ein Befehl aber ohne Schaden nachträglich eingefügt werden. Die Kaltstartroutine im ersten Sektor der Diskette wird wirklich nur bei einem Kaltstart aufgerufen. Sie muß also so verändert werden, daß sie nach dem Übertragen des BDOS und des CCP in den Speicher zusätzlich einen frei zu wählenden Text in den CCP-Bereich kopiert.

Dieses Verfahren kann übrigens auch in einem Anwenderprogramm verwendet werden, das nach dem Abschluß gleich ein neues Programm aufrufen soll. Dabei ist aber Vorsicht geboten: Wenn sich irgendein Programm analog zum XSUB.COM vor dem BDOS eingehängt hat, kann die Anfangsadresse des CCP nicht mehr ohne weiteres bestimmt werden. Diese Situation tritt häufig bei gekauften Anwenderprogrammen auf, aus denen andere Programme als 'Unterprogramme' aufgerufen werden können, und die nach Abschluß des 'Unterprogramms' erneut gestartet werden.

9.3 Erweitern des Betriebssystems

Nach einer Verschiebung des Betriebssystems wird Speicher frei, der für residente Erweiterungen zur Verfügung steht. Der freie Speicher liegt jedoch nicht - wie man vermuten könnte - direkt unterhalb der im MOVCPM-Befehl angegebenen Adresse. Den genauen Speicheraufbau nach der Verschiebung zeigt Bild 9-2.

In den CPC-Computern besteht das BIOS aus 2 logischen Teilen: Dem Arbeitsbereich und der standardisierten BIOS-Sprungtabelle. Bei einer Verschiebung des Betriebssystems im Speicher wandert die BIOS-Sprungtabelle mit dem BDOS mit. Der Arbeitsbereich des BIOS bleibt jedoch immer am oberen Speicherende fixiert. Der entstehende freie Speicherplatz liegt also zwischen den beiden Teilen. Bei einer mit 'MOVCPM 179 *' erzeugten CP/M-Version schrumpft der freie Speicherplatz zu Null und die beiden Teile treffen sich an der Adresse AD32h/AD33h. Bei einer Verschiebung wird unterhalb dieser Trennstelle Speicher in 256-Byte-Schritten frei. Etwa bei einer durch 'MOVCPM 177 *' erzeugten Version steht Speicherplatz von der Adresse AB33h bis einschließlich AD32h für Betriebssystemerweiterungen und ähnliches uneingeschränkt zur Verfügung. Diese Aufteilung gilt im Allgemeinen sowohl bei Verwendung der Schneider-Floppydisk als auch bei Verwendung der Vortex-Floppydisk mit einer Ausnahme: Bei der Version VDOS 2.0 beginnt der BIOS-Arbeitsbereich schon an der Adresse AC33h statt an der Adresse AD33h. Der unbelegte Speicherplatz ist also um 256 Bytes kürzer als bei den anderen Versionen.

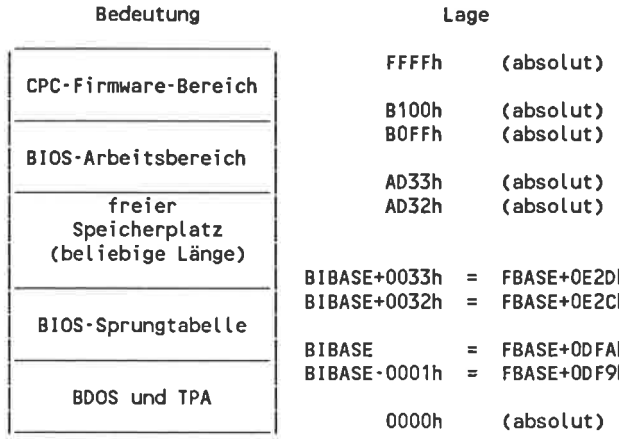


Bild 9-2: *Speicheraufteilung im CPC-BIOS*

9.4 Eine einfache Betriebssystemerweiterung

Ein Fehler im VDOS 2.0-ROM

Das derzeit (Dezember 1985) in den Vortex-Floppydiskcontrollern enthaltene ROM enthält noch einen Fehler. In der Standard-BIOS-Sprungtabelle ist der Einsprung des Puncher-Kanals mit dem des Reader-Kanals vertauscht.

Der Stanzerkanal führt zu verschiedenen Lesegeräten und der Leserkanal zu Stanzerkanälen! Wenn das BDOS also etwa eine Lochstreifenausgabe verlangt, wartet die angesprochene BIOS-Routine bis zufällig ein Zeichen zum Lesen anliegt: Die Folge ist, daß sich der Computer 'aufhängt'.

Wenn Ihre Experimente mit dem Reader-/Puncherkanal also nicht wunschgemäß funktionieren, sollten Sie nicht gleich an Ihren Programmierkünsten zweifeln, sondern zuerst testen, ob in Ihrem Laufwerk ein fehlerhaftes ROM eingebaut ist.

Am einfachsten gehen Sie dazu so vor: Mit 'STAT PUN:=UP2:' weisen Sie dem Puncherkanal das reale UP2-Gerät zu. In den CPC-Computern ist das die Bildschirmausgabe. Mit 'PIP PUN:=<Textdatei>' senden Sie dann eine Textdatei an den Puncherkanal. Wenn die Datei jetzt nicht genau wie beim TYPE-Befehl auf dem Bildschirm ausgeben wird, besitzen Sie die fehlerhafte ROM-Version.

Für die Dauer einzelner Experimente können Sie diesen Fehler aber durch den Start des Reparaturpogramms aus Bild 9-3 beseitigen. In der rechten Spalte sehen Sie die spezielle Version für eine durch 'MOVCPM 177 *' erzeugte CP/M-Version und in der linken Spalte die allgemeine Version. BIBASE ist dabei die Anfangsadresse der Standard-BIOS-Sprungtabelle. Am besten tippen Sie es mit Hilfe des DDT-Programms ein.

```
mvi  a,FAh                    mvi  a,FAh
sta  BIBASE+0013h             sta  AB13h
mvi  a,F6h                    mvi  a,F6h
sta  BIBASE+0016              sta  AB16h
ret                           ret
```

Bild 9-3: *Programm REPAIR.COM für 'MOVCPM 177 *'-CP/M*

Die mit diesem einfachen Programm durchgeführte Reparatur bleibt leider nur bis zum nächsten Warmstart erhalten. Vor jedem Programm, das den Reader-/Puncherkanal verwendet, muß das Repair-Programm also erneut

aufgerufen werden. Das können Sie aber immerhin automatisch erledigen lassen: Durch das Einfügen des Repair-Aufrufes in den Befehlspuffer des CCPs wird die Reparatur bei jedem Warmstart durchgeführt. Dadurch wird das Betriebsystem aber leider langsamer. Eine dauerhafte Änderung ohne zusätzlichen Diskettenzugriff ist nur mit erheblichem Aufwand und Änderung der Warmstartroutine möglich.

Implementierung zusätzlicher Ein-/Ausgabegeräte

In den CPC-Computern werden normalerweise die Geräteschnittstellen für die SIO1 und die SIO2 nicht genutzt. Es ist aber möglich, stattdessen selbstdefinierte Zusatzgeräte ins Betriebssystem zu integrieren, die dann auch wirklich verwendet werden können.

Möglich wäre es zum Beispiel, über die SIO-Geräte den Kassettenrekorder anzusprechen oder sogar ein selbstgebautes Telephon-Modem: In diesem Fall kann ohne weitere Software, nur mit Hilfe von PIP, eine komplette Mailboxenkommunikation abgewickelt werden. In diesem Beispiel werden aber andere Geräte eingebaut:

SIO1 als Bildschirm-Ein-/Ausgabegerät

Die SIO1 (das TTY:-Gerät) arbeitet jetzt genau wie das CRT:-Gerät im Konsolenkanal des Computers: Ausgaben über die SIO1 wirken auf den Bildschirm und Eingaben erfolgen über die Computertastatur. Nach der Zuweisung 'STAT CON:=TTY:' verhält sich der CPC jetzt also genau wie unter der normalen Einstellung. Bisher wäre er in diesem Fall abgestürzt.

Diese Möglichkeit ist natürlich nur eine Spielerei. Die entscheidende Verbesserung ist aber, daß das TTY-Gerät über alle 4 Kanäle angesprochen werden kann, das CRT-Gerät aber nicht. Deshalb können jetzt nach entsprechenden 'STAT ...=TTY:'-Zuweisungen wirklich alle Ein- und Ausgaben über Bildschirm und Tastatur geleitet werden. Insbesondere gilt das auch für den Drucker-Kanal. Beim Testen von Programmen können Sie sich so eine Menge Papier sparen.

SIO2 als Protokoll-Gerät

Eingaben über die SIO2 werden wie bei der SIO1 von der Tastatur geholt. SIO2-Ausgaben jedoch werden jetzt sowohl auf den Bildschirm als auch auf den Drucker geschickt. Wie das SIO1-Gerät steht auch das SIO2-Gerät auf allen logischen Kanälen zur Verfügung. In den verschiedenen Kanälen wird das SIO-Gerät allerdings verschieden bezeichnet: Im Konsolenkanal mit UC1:, im Readerkanal mit UR1:, im Puncherkanal mit UP1: und im Druckerkanal mit UL1:.

Nach der Zuweisung 'STAT CON:=UC1:' laufen alle Konsolenein-/ausgaben über das SIO2-Gerät. Sie können aber weiterhin Tastatureingaben machen und die Ausgaben erscheinen wie gewöhnlich auf dem Bildschirm. Zusätzlich werden die Ausgaben aber auf dem Drucker protokolliert. Im Vergleich zum normalen Protokoll, das mit ^P eingeschaltet wird, bietet diese Methode die folgenden Vorteile:

- Bei jedem Warmstart wird die Standard-Protokollfunktion abgeschaltet. Der Ablauf von Submit-Programmen läßt sich so nicht protokollieren: Nur die Ein-/Ausgaben des ersten aufgerufenen Programms sind sichtbar. Die neue Protokollfunktion bleibt jedoch solange aktiv, bis sie explizit mit 'STAT CON:=CRT:' oder 'STAT CON:=TTY:' abgeschaltet wird.
- Die normale Protokollfunktion wird durch mehrmaliges oder versehentliches Drücken der ^P-Taste abwechselnd ein- und ausgeschaltet. Eine Verwechslung ist so sehr leicht möglich. Bei der neuen Protokollfunktion ist das wegen der verschiedenen Namen nicht mehr möglich.
- Die normale Protokollfunktion wird vom BDOS-gesteuert: Bei Bedarf wird das anzuzeigende Zeichen einfach sowohl an den Konsolenkanal als auch an den Druckerkanal geschickt. Das erfolgt jedoch nur bei bestimmten BDOS-Funktionen zur Konsolenausgabe: Bei der BDOS-Funktion Nr. 6 'Direkte Konsolenein-/ausgabe' wird prinzipiell keine parallele Druckerausgabe durchgeführt. Programmabläufe, die diese Funktion verwenden, können so auf normal Art nicht protokolliert werden. Ein Beispiel dafür sind mit Turbo-Pascal erstellte Programme: Druckerausgaben sind hier nur durch eine explizite Zuweisung im Programmtext und nach Eröffnung einer 'Druckerdatei' möglich. Die Protokollfunktion durch das SIO2-Gerät wird jedoch durch das BIOS gesteuert, so daß das BDOS keinen Einfluß nehmen kann. Es kann so wirklich jede beliebige Bildschirmausgabe protokolliert werden.

Sinnvoll ist auch eine Zuweisung des SIO2-Gerätes an den Druckerkanal: Dann werden alle Druckerausgaben zugleich auf dem Bildschirm angezeigt. Dabei ist aber Vorsicht geboten: Wenn Sie gleichzeitig den Konsolenkanal an das SIO2-Gerät zugewiesen haben und die ^P-Taste drücken, werden alle Zeichen sowohl auf dem Bildschirm als auch auf dem Drucker doppelt ausgegeben.

Die praktische Durchführung der Implementation

Die zusätzlichen Ein-/Ausgabefunktionen liegen im Speicherbereich ab der Adresse AB33h. Deshalb sind sie nur zusammen mit einer durch 'MOVCPM 177 *' erzeugten CP/M-Version lauffähig. Die Lage wird im Assemblerlisting aber nur in einer einzigen Zeile festgelegt. Durch eine Änderung

dieser Zeile können Sie das Programm für eigene Zwecke durchaus an einer anderen Stelle ablegen.

Die Einbindung der Routinen ins Betriebssystem erfolgt, indem an die Einsprungadressen der Kommunikationsschnittstelle im Bereich von BEA7h bis BEBEh Sprünge auf die neuen Routinen eingetragen werden. Das Programm EXTERM.COM besteht deshalb aus 2 Teilen: Der 2. Teil enthält die eigentlichen Ein-/Ausgaberoutinen. Der 1. Teil kopiert nur die Routinen in den endgültigen Speicherbereich und paßt die Zeiger der Kommunikationsschnittstelle an.

Leider sind die Einsprungadressen beim Schneider- und beim Vortex-Laufwerk nicht genau gleich, so daß das Programm Ihrer Computerkonfiguration angepaßt werden muß. Dazu braucht aber nur eine einzige Zeile im Assemblerlisting geändert werden: Wenn Sie ein Schneider-Laufwerk besitzen, müssen Sie im Definitionsteil des Listings statt 'vortex equ true', 'vortex equ false' eintragen.

Die Routinen stützen sich sehr stark auf die CPC-Firmwareeinsprünge. Details über die ca. 250 Firmwareroutinen haben jedoch nicht mehr viel mit der CP/M-Programmentwicklung zu tun. Wenn Sie sich dafür interessieren, sollten Sie zu Spezialliteratur über dieses Thema greifen.

Wichtig ist die Routine zum Feststellen des Ausgabestatus der SIO1: Gerade weil eine Bildschirmausgabe immer möglich ist, muß diese Routine immer den Wert 255 liefern.

Bei den Routinen zur Bildschirmansteuerung wurde der Hauptaufwand für die richtige Darstellung des Cursors bei einer Tastatureingabe verwendet: Dazu existiert ein Flag, in dem angegeben ist, ob der Cursor zur Zeit sichtbar ist. Vor einer Bildschirmausgabe wird er bei Bedarf ausgeschaltet und beim Warten auf einen Tastendruck wird er eingeschaltet. Beim Feststellen des Tastaturstatus ohne Warten auf ein Zeichen wird er jedoch nur eingeschaltet, wenn mehrere Statusabfragen hintereinander ohne zwischenzeitliche Bildschirmausgabe auftreten. Nur so vermeidet man, daß der Cursor bei jeder BDOS-gesteuerten Ausgabe, bei der auf Tastendruck eine Unterbrechnung möglich ist, kurz aufleuchtet.

Die originale Bildschirmausgaberoutine rollt den Bildschirm sofort nach oben, wenn das nächste Zeichen außerhalb des Bildschirms positioniert werden würde. Für ein stabiles Bildschirmbild darf deshalb die Position ganz rechts unten nicht beschrieben werden. Diese Einschränkung gibt es bei der neuen Routine nicht mehr. ansonsten ist die neue Bildschirmausgabe jedoch voll kompatibel zur originalen Bildschirmausgabe.

Die neuen Routinen verwenden jedoch ein eigenes Cursor-Flag, dessen Wert sich vom entsprechenden BIOS-internen Flag unterscheiden kann. Beim Umschalten von den originalen auf die neuen Ausgaberoutinen kann deshalb ein Cursorsymbol als 'Müll' auf dem Bildschirm stehenbleiben. Für besonders 'saubere' Programme sollten Sie deshalb dem Konsolenkanal ständig entweder das TTY-Gerät oder das UC1-Gerät zuordnen.

Das Listing der erweiterten Ein-/Ausgabefunktionen

```
                ;Erweiterte Ein-/Ausgabemöglichkeiten:
                ;SIO1 wird als Terminal und SIO2 als Drucker definiert
                ;=> alle Kanäle können auf Terminal+Drucker umgeleitet
                ; werden
                ;benötigt mit 'MOVCPM 177 *' erzeugte CP/M-Version!
                ;
                ;****** Startadressen
AB33 =          base    equ     0AB33h          ;Engültige Lage des Programms
0100            org     0100h           ;Jetzige Lage
                ;****** Unterscheidung, ob Vortex oder Schneider-Floppy
FFFF =          true    equ     0FFFFh
0000 =          false   equ     not true
FFFF =          vortex  equ     true            ;Bei Schneider: false
                ;
                ;****** Adressen der Kommunikationsschnittstelle
                ;(nur beim Vortex-Laufwerk)
                        if      vortex
BEAA =          insta2  equ     0BEAAh          ;SIO2 Eingabestatus
BEB0 =          otsta2  equ     0BEB0h          ;SIO2 Ausgabestatus
BEB3 =          incha1  equ     0BEB3h          ;SIO1 Zeichen empfangen
BEB9 =          otcha1  equ     0BEB9h          ;SIO1 Zeichen senden
                        endif
                ;(nur beim Schneider-Laufwerk)
                        if      not vortex
                incha1  equ     0BEAAh          ;SIO1 Zeichen empfangen
                otcha1  equ     0BEB0h          ;SIO1 Zeichen senden
                insta2  equ     0BEB3h          ;SIO2 Eingabestatus
                otsta2  equ     0BEB9h          ;SIO2 Ausgabestatus
                        endif
                ;(sowohl bei Vortex als auch bei Schneider)
BEA7 =          insta1  equ     0BEA7h          ;SIO1 Eingabestatus
BEAD =          otsta1  equ     0BEADh          ;SIO1 Ausgabestatus
BEB6 =          incha2  equ     0BEB6h          ;SIO2 Zeichen empfangen
BEBC =          otcha2  equ     0BEBCh          ;SIO2 Zeichen senden
                ;
                ;****** CPC-Firmwareeinsprünge
BB06 =          kmwait  equ     0BB06h          ;KM WAIT CHAR
BB09 =          kmread  equ     0BB09h          ;KM READ CHAR
BB0C =          kmretn  equ     0BB0Ch          ;KM CHAR RETURN
BB5A =          txtout  equ     0BB5Ah          ;TXT OUTPUT
BB81 =          txtcon  equ     0BB81h          ;TXT CURSOR ON
BB84 =          txtcof  equ     0BB84h          ;TXT CURSOR OFF
BD2B =          mcprnt  equ     0BD2Bh          ;MC PRINT CHAR
BD2E =          mcbusy  equ     0BD2Eh          ;MC BUSY PRINTER
```

```
                ;
                ;****** Programm aus jetziger in endgültige Lage Übertragen
                ;(Blocktransfer)
0100 013F00             lxi     b,ende-start    ;Programmlänge
0103 1133AB             lxi     d,base          ;Engültige Lage
0106 213601             lxi     h,start         ;Jetzige Lage
0109 EDB0               dw      0B0EDh          ;Entspricht Z80-LDIR
                ;(BIOS-Einsprungadressen anpassen)
010B 2150AB             lxi     h,consta+versatz;Konsolenstatus
010E 22A8BE             shld    insta1+1        ;SIO1
0111 22ABBE             shld    insta2+1        ;SIO2
0114 2164AB             lxi     h,coninp+versatz;Konsoleneingabe
0117 22B4BE             shld    incha1+1        ;SIO1
011A 22B7BE             shld    incha2+1        ;SIO2
011D 213FAB             lxi     h,conout+versatz;Konsolenausgabe
0120 22BABE             shld    otcha1+1        ;SIO1
0123 214DAB             lxi     h,conost+versatz;Konsolenausgabestatus
0126 22AEBE             shld    otsta1+1
0129 2138AB             lxi     h,listot+versatz;Druckerausgabe
012C 22BDBE             shld    otcha2+1        ;SIO2
012F 2133AB             lxi     h,listst+versatz;Druckerstatus
0132 22B1BE             shld    otsta2+1        ;SIO2
0135 C9                 ret
                ;
                start:
                ;****** die einzufügenden Routinen
                ;(SIO2 -> Drucker bereit?)
0136 CD2EBD     listst: call    mcbusy          ;Drucker bereit?
0139 9F                 sbb     a               ;ja -> 0FFh
013A C9                 ret
                ;(SIO2 -> Zeichen an Drucker)
013B 79         listot: mov     a,c             ;Zeichen ins a-Register
013C CD2BBD             call    mcprnt          ;Ausdrucken
013F D238AB             jnc     listot+versatz  ;solange bis erfolgreich
                ;(SIO1 -> Bildschirmausgabe ;zugleich als Druckerecho)
0142 2171AB     conout: lxi     h,curflg+versatz;Cursor-Flag
0145 7E                 mov     a,m
0146 36FF               mvi     m,0FFh          ;Flag: Cursor unsichtbar
0148 A7                 ana     a               ;Cursor sichtbar?
0149 CC84BB             cz      txtcof          ;Cursor ausschalten
014C 79                 mov     a,c             ;Zeichen ins a-Register
014D C35ABB             jmp     txtout          ;Zeichen an Bildschirm
                ;(SIO1 -> Ausgabestatus Bildschirm; klappt immer)
0150 3EFF       conost: mvi     a,0FFh          ;Kennzeichen o.k.
0152 C9                 ret
                ;(SIO1 -> Taste gedrückt?)
0153 2171AB     consta: lxi     h,curflg+versatz;Cursor-Flag
0156 7E                 mov     a,m
0157 D601               sui     01h             ;Cursor sichtbar machen?
0159 CC81BB             cz      txtcon          ;Nur, wenn curflg Null wird
015C CE00               aci     00h             ;Übertrag -> bleibt Null
015E 77                 mov     m,a             ;Flag: Cursor
015F CD09BB             call    kmread          ;Taste gedrückt?
0162 DC0CBB             cc      kmretn          ;Wenn ja, für später aufheben
0165 9F                 sbb     a               ;Taste da -> 0FFh
0166 C9                 ret
```

```
                ;(SIO1 -> Taste holen)
0167 2171AB     coninp: lxi     h,curflg+versatz;Cursor-Flag
016A 7E                 mov     a,m
016B 3600               mvi     m,00h           ;Flag: Cursor sichtbar
016D A7                 ana     a               ;Cursor angezeigt?
016E C481BB             cnz     txtcon          ;Cursor einschalten
0171 C306BB             jmp     kmwait          ;Auf Tastendruck warten
                ;
                ;****** Zwischenspeicher
0174 FF         curflg: db      0FFh
                ende:
                ;
                ;****** Differenz jetzige <-> engültige Lage
A9FD =          versatz equ     base-start
0175                    end
```

9.5 Der Kaltstartlader

Die Funktionsweise des Kaltstartladers

Auf dem ersten Sektor der Spur 0 der Diskette im Laufwerk A befindet sich ein kurzes Maschinenprogramm: der Kaltstartlader. Dieser wird sofort nach der Wahl des CP/M-Modus in den RAM-Speicherbereich von 0100h bis 02FFh geladen und durch einen Sprung zur Adresse 0100h gestartet. In den CPC-Computern führt dieses Programm die verschiedenen Voreinstellungen aus, die mit SETUP oder INSTALL gewählt wurden und sofort nach dem Einschalten zur Verfügung stehen.

Zur Zeit, in der der Kaltstartlader aktiv ist, ist das CP/M-System überhaupt noch nicht initialisiert. Nur der I/O-Byte-Standartwert in der Speicherstelle 0003h und der Code für Laufwerk A/User 0 ist schon gesetzt. Außer diesen Werten befindet sich im RAM nur die CPC-spezifische Firmware. Nicht vorhanden sind aber die Diskettenbeschreiber (DPH und DPB) und vor allem das BDOS. Außer mit Hilfe von Tricks kann der Kaltstartlader also keine Dateien verarbeiten. Bereits initialisiert sind die speziellen CPC-BIOS-Routinen im Bereich von BE80h bis BEBEh. Der Kaltstartlader kann also immerhin Daten im Direktzugriff von der Diskette holen. Weiterhin darf er alle Standard-BIOS-Routinen aufrufen. Damit sind auch Bildschirmausgaben erlaubt. Die BIOS-Sprungtabelle befindet sich zu diesem Zeitpunkt aber noch nicht im RAM an der endgültigen Position, sondern noch im ROM.

Der Kaltstartlader erhält vom aufrufenden Programm im BC-Register die Anfangsadresse der BIOS-Sprungtabelle. Der Stackpointer zeigt auf die Adresse hinter der letzten freien Speicherstelle im RAM: Ein Unterprogramm-Stapel wird also genau am oberen Speicherende angelegt. Da die RAM-BIOS-Tabelle und das BDOS noch nicht initialisiert sind, hat die letzte freie Speicherstelle hier in der Regel die Adresse AD32h. Das muß aber nicht unbedingt sein: Beim VDOS 2.0 der Vortex-Floppydisk ist das die Adresse AC32h. Die Kaltstartroutine muß mit einem Sprung zur Warmstart-Einsprungadresse enden. Angesprochen werden muß aber nicht unbedingt die im ROM eingebaute Warmstartroutine, sondern es kann auch eine selbstgeschriebene verwendet werden.

Verändern des Kaltstartladers

Der Standard-Kaltstartlader kopiert als allererstes die BIOS-Sprungtabelle in einen genau bestimmten Speicherbereich. Das ist jedoch nicht zwingend notwendig, sondern nur ein 'verwaltungstechnischer Trick': BIOS-Routinen können im folgenden mit absoluten Sprüngen aufgerufen werden, ohne die Lage der Einsprungadresse erst berechnen zu müssen.

Als nächstes wird der 2. Sektor der Diskette in den Speicherbereich von 0300h bis 04FFh geladen. Dieser enthält die mit SETUP (Schneider) oder INSTALL (Vortex) gewählten Parameter. In einem eigenen Kaltstartlader können Sie die Auswertung einfach weglassen: In diesem Fall bleiben die alten Standardwerte gültig.

Im Gegensatz zum Schneider-Laufwerk muß im Kaltstartlader des Vortex-Laufwerkes unbedingt ein 'call BE86h' ausgeführt werden: Sonst wird das System nicht richtig initialisiert. Ansonsten sind die beiden Kaltstartlader zu den beiden Laufwerken analog aufgebaut - wenn es auch im Detail Unterschiede gibt. Der Kaltstartlader der Vortex-Floppydisk gibt zum Beispiel Meldungen über die gewählten Voreinstellungen aus. Diese sind aber nur 'Verzierung' und können weggelassen werden.

Die 5 Sektoren der ersten Systemspur mit den logischen Nummer 3 bis 7 sind normalerweise unbelegt. Der Kaltstartlader könnte also so umgebaut werden, daß er nicht nur den Parametersektor, sondern auch diese 5 fortlaufend in den Speicher liest. Dem Kaltstartlader stehen dann volle 3,5 KByte zur Verfügung.

Beachten Sie, daß im Standard-Kaltstartlader der Gebrauch der gewöhnlichen Disketten-BIOS-Routinen vermieden wird: Wenn es irgendeine Möglichkeit gibt, werden die speziellen CPC-BIOS-Routinen verwendet.

```
0100  LXI  H,010E          0120  POP  H
0103  LXI  D,0080          0121  POP  D
0106  LXI  B,0028          0122  POP  B
0109 ED B0 ..              0123  INR  H
010B  JMP  0080            0124  INR  H
010E  LXI  SP,6000         0125  INR  C
0111  LXI  H,0100          0126  MOV  A,C
0114  MVI  E,00            0127  CPI  4A
0116  MVI  D,00            0129  JNZ  008C
0118  MVI  C,41            012C  INR  D
011A  PUSH B               012D  MOV  A,D
011B  PUSH D               012E  CPI  02
011C  PUSH H               0130  JNZ  008A
011D  CALL BE89            0133  JMP  0000
```

Bild 9-4: *Programm READSYS.COM: Liest Systemspuren*

Untersuchen des Kaltstartladers

Mit Hilfe des Programms aus Bild 9-4 können Sie die Systemspuren einer Diskette im Laufwerk A in den Speicher kopieren. Dort ist zum Beispiel mit DDT eine einfache Untersuchung möglich. Wegen der geringen Länge geben Sie das Programm am besten gleich mit Hilfe des DDT-Programms ein, ohne den Assembler zu bemühen.

Der erste Teil des Readsys-Programms kopiert das eigentliche Ladeprogramm in den Speicherbereich von 0080h bis 00FFh und startet es. Dabei wird der Z80-Befehl LDIR verwendet. Da der 8080-Debugger diesen nicht kennt, müssen Sie mit Hilfe der S-Funktion des Debugger direkt die Hexcodes EDh und B0h eingegeben.

Die vorliegende Version funktioniert nur bei einer 3"-Systemdiskette. Zum Lesen einer Vortex-Diskette müssen Sie die Befehle an den Adressen 0118h und 0127h zu 'MVI C,01' und 'CPI 0A' ändern.

Das Programm an der Adresse 0080h legt zunächst den Stapel in das zentrale RAM und lädt dann mit Hilfe einer CPC-spezifischen BIOS-Funktion Sektoren in den Speicher. Nach jedem Lesebefehl wird die Pufferadresse um den Wert 0200h erhöht, so daß die Systemsektoren fortlaufend im Speicher abgelegt werden. Nach dem Programmende befinden sich die Systemspuren im Speicher ab der Adresse 0100h und Sie können sie mit 'SAVE 36 CPM.SYS' abspeichern. Mit DDT können Sie die erhaltene Datei untersuchen. Beachten Sie aber, daß im Kaltstartlader viele Z80-spezifische Objektcodes auftreten. Das nachfolgenden Listing wurde deshalb zur besseren Übersicht ebenfalls im Z80-Code gehalten.

Wenn Sie im Kaltstartlader oder im CP/M-System Veränderungen angebracht haben, möchten Sie das neue System sicher wieder auf der Diskette abspeichern. Dazu müssen Sie zunächst nur den 'call BE89h'-Befehl (Sektor lesen) in einen call 'BE8Ch'-Befehl (Sektor schreiben) umwandeln, um den Bereich ab 0100h abzuspeichern. Zusätzlich müssen Sie aber noch darauf achten, daß der mühsam erstellte Kaltstartlader nicht vom SAVE-Programm überschrieben wird. Vor dem Abspeichern müssen Sie den Kaltstartlader also mit Hilfe des DDT in einen sicheren Bereich kopieren und die Diskettenpufferadresse entsprechend ändern.

Das Listing des Schneider-Kaltstartladers

```
******* Kaltstartlader-Einsprung
0100  LD   H,B          ;bc zeigt auf BIOS-Sprungtabelle
0101  LD   L,C
0102  LD   DE,0500      ;Ziel beim kopieren
0105  LD   BC,0033
0108  LDIR              ;BIOS-Tabelle an Adresse 0500 kopieren
010A  LD   C,42         ;Sektor Nr. 42
010C  LD   DE,0000      ;Laufwerk A/Spur 0
010F  LD   HL,0300      ;Pufferadresse
0112  CALL BE89         ;Sektor lesen=Konfigurationssektor
0115  JR   NC,06=011D   ;Fehler beim Lesen aufgetreten
0117  CALL 0150         ;System-Voreinstellungen bearbeiten
011A  JP   0503         ;Warmstart


******* Fehler: Konfigurationssektor konnte nicht gelesen werden
011D  CALL 0259         ;Nachfolgenden Text ausgeben

0120 0D 0A 46 61 69 6C 65 64 20 74 6F 20 6C 6F 61 64 ..Failed to load
0130 20 74 68 65 20 63 6F 6E 66 69 67 75 72 61 74 69  the configurati
0140 6F 6E 20 73 65 63 74 6F 72 0D 0A 0A             on sector...

014C  INC  H            ;hier wird Programm nach Ausgabe fortgesetzt
014D  JP   0503         ;Warmstart


******* Bildschirmmodus wählen
0150  CALL 0259         ;Nachfolgenden Text ausgeben

0153 04 02 24 ..$

0156  LD   HL,(0300)    ;Hier wird Programm nach Ausgabe fortgesetzt
0159  LD   DE,EDCB      ;Vergeichscode
015C  ADD  HL,DE        ;Addition zu Wert aus Konfigurationssektor
015D  LD   A,H
015E  OR   L
015F  JR   Z,26=0187    ;kein Prüfsummenfehler


******* Prüfsummenfehler im Konfigurationssektor
0161  CALL 0259         ;Nachfolgenden Text ausgeben

0164           0D 0A 49 6C 6C 65 67 61 6C 20 63 6F     ..Illegal co
0170 6E 66 69 67 75 72 61 74 69 6F 6E 20 73 65 63 74 nfiguration sect
0180 6F 72 0D 0A 0A 24                               or...$

0186  RET              ;Rücksprung ins Hauptprogramm


******* Weitere System-Voreinstellungen
0187  LD   HL,(0302)    ;Wartezeit nach dem Einschalten
018A  LD   (023D),HL
018D  LD   HL,(0304)    ;Nachlaufzeit nach dem Ausschalten
0190  LD   (023F),HL
```

```
0193  LD   A,(0306)      ;Zeit zum Spurwechsel
0196  LD   (0243),A
0199  LD   HL,023D       ;Adresse der Zeitkonstanten
019C  CALL BE83          ;Floppy-Disk Zeitkonstanten
019F  LD   A,(0307)      ;Input/Output-Byte
01A2  LD   (0003),A
01A5  LD   A,(0308)      ;BIOS-Meldungen erlaubt?
01A8  CALL BE80          ;Meldungen ein-/ausschalten
01AB  LD   A,(0309)      ;Fast/Slow-Modus
01AE  CALL BE9E          ;Fast/Slow-Modus wählen
01B1  LD   HL,030A       ;Tabelle der SIO-Daten
01B4  CALL BEA1          ;SIO initialisieren
01B7  LD   HL,0364       ;Adresse des Initialisierungstextes
01BA  CALL 025F          ;Zeichenkette ausgeben


****** Printer-Power-Up-String ausgeben
01BD  LD   A,(HL)        ;Länge des Printer-Power-Up-Strings
01BE  INC  HL
01BF  OR   A
01C0  JR   Z,0C=01CE     ;kein Power-Up-String
01C2  LD   B,A
01C3  LD   C,(HL)        ;erstes Zeichen
01C4  INC  HL
01C5  PUSH HL
01C6  PUSH BC
01C7  CALL 050F          ;BIOS-Funktion Drucker-Ausgabe
01CA  POP  BC
01CB  POP  HL
01CC  DJNZ F5=01C3       ;kein weiteres Zeichen


******* Tastenbelegung einstellen
01CE  EX   DE,HL         ;de enthält Adresse des nächsten Zeichens
01CF  LD   HL,BB27       ;Adresse 'KM SET TRANSLATE'
01D2  CALL 0246          ;Routine ausführen
01D5  LD   HL,BB2D       ;Adresse 'KM SET SHIFT'
01D8  CALL 0246          ;Routine ausführen
01DB  LD   HL,BB33       ;Adresse 'KM SET CONTROL'
01DE  CALL 0246          ;Routine ausführen
01E1  EX   DE,HL         ;(hl) ist nächstes zu bearbeitendes Zeichen


******* Expansions-Strings belegen
01E2  LD   A,(HL)
01E3  INC  HL
01E4  OR   A             ;Anzahl der Expansionsstrings
01E5  JR   Z,50=0237     ;kein Expansionsstring
01E7  LD   D,A
01E8  LD   B,(HL)        ;Nummer des Expansionsstrings
01E9  INC  HL
01EA  LD   C,(HL)        ;Länge des Expansionsstrings
01EB  INC  HL
01EC  PUSH HL
01ED  PUSH DE
01EE  PUSH BC
01EF  CALL BB0F          ;'KM SET EXPAND'
01F2  POP  BC
```

```
01F3  POP  DE
01F4  POP  HL
01F5  JR   C,3A=0231      ;Alles in Ordnung


******* Expansions-Puffer voll
01F7  CALL 0259           ;folgenden Text ausgeben

01FA                                  0D 0A 45 78 70 61           ..Expa
0200 6E 73 69 6F 6E 20 62 75 66 66 65 72 20 66 75 6C nsion buffer ful
0210 6C 20 6F 72 20 69 6C 6C 65 67 61 6C 20 74 6F 6B l or illegal tok
0220 65 6E 20 73 70 65 63 69 66 69 65 64 0D 0A 0A 24 en specified...$

0230  RET                 ;zurück ins Hauptprogramm


******* Hier weiter nach korrektem Expansionsstring
0231  LD   B,00           ;c enthält Länge des Expansionsstrings
0233  ADD  HL,BC          ;hl zeigt auf nächsten Expansionsstring
0234  DEC  D
0235  JR   NZ,B1=01E8     ;kein weiterer Expansionsstring


******* hl zeigt auf Initial-Command-Buffer-String, 1.Byte ist Länge
0237  LD   A,(0316)       ;0=Clear on Input
023A  JP   BEA4           ;String initialisieren und Rücksprung


******* Zeitkonstanten für Floppy-Disk
023D 00 00 00 00 AF 1E 00 01 03 .........


******* Keyboard-Translations, hl zeigt auf Routinenadresse
0246  LD   A,(DE)         ;nächstes zu behandelnde Zeichen
0247  LD   C,A            ;Anzahl der geänderten Tasten
0248  INC  DE
0249  OR   A
024A  RET  Z              ;keine Taste geändert
024B  LD   A,(DE)
024C  LD   B,A            ;neue Übersetzung der Taste
024D  INC  DE
024E  LD   A,(DE)         ;Tastennummer
024F  INC  DE
0250  PUSH HL
0251  CALL 026C           ;Routine (hl) aufrufen
0254  POP  HL
0255  DEC  C
0256  JR   NZ,F3=024B     ;keine weitere Übersetzung
0258  RET


******* Text ausgeben, der hinter dem Call-Befehl folgt
0259  EX   SP,HL          ;hl enthält jetzt erstes Textzeichen
025A  CALL 025F           ;Zeichenkette ab hl bis '$' ausgeben
025D  EX   SP,HL          ;(sp) enthält jetzt erstes Zeichen hinter Text
025E  RET                 ;Programm hinter dem Text fortsetzen
```

```
******* Zeichenkette ab hl bis '$' ausgeben
025F  LD   A,(HL)        ;erstes Zeichen
0260  INC  HL
0261  CP   24            ;Endemarkierung
0263  RET  Z
0264  LD   C,A
0265  PUSH HL
0266  CALL 050C          ;BIOS-Routine Konsolenausgabe
0269  POP  HL
026A  JR   F3=025F       ;zum nächsten Zeichen


******* gehört zur Tastenbelegungsroutine
026C  JP   (HL)


******* Dieser Bereich wird nicht benötigt
026D bis 02FF


******* Der Konfigurationssektor wird erst zur Ausführungszeit in
       diesen Bereich geladen
0300 35 12 32 00 FA 00 0C 81 00 00 44 6A E1 44 6A E1 5.2.......Dj.Dj.
0310 0D 00 0D 00 0D 00 00 00 00 00 00 00 00 00 00 00 ................
0320 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0330 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0340 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0350 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0360 00 00 00 00 1C 40 40 40 1C 61 7A 7A 1D 40 40 43 .....§§§.azz.§§C
0370 50 2F 4D 20 32 2E 32 20 2D 20 41 6D 73 74 72 61 P/M 2.2 - Amstra
0380 64 20 43 6F 6E 73 75 6D 65 72 20 45 6C 65 63 74 d Consumer Elect
0390 72 6F 6E 69 63 73 20 70 6C 63 0A 0D 24 00 00 00 ronics plc..$...
03A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

03B0 bis 04FF: Der Konfigurationssektor könnte auch noch diesen
               Bereich benützen


******* Zur Ausführungszeit wird in diesen Bereich die
       BIOS-Sprungtabelle kopiert
0500 bis 0532h
```

10 Besondere Befehle für den Assembler-programmierer

10.1 Die Befehle SUBMIT/XSUB

Das Funktionsprinzip

Mit Hilfe des SUBMIT-Befehls können Programme ohne weitere Anwendereingaben automatisch gestartet werden. Zum Abarbeiten der Programmliste PROGLIST.SUB muß

```
SUBMIT PROGLIST <Parameter1> <Parameter2>...
```

eingegeben werden. PROGLIST.SUB kann dabei mit Hilfe eines beliebigen Texteditors erstellt worden sein.

Das SUBMIT-Programm lädt dann die Datei PROGLIST.SUB in den Speicher und übersetzt sie in eine vom Computer besser lesbare Form. In der Programmliste enthaltene Zeichenfolgen '$1', '$2' und so weiter werden dabei durch die Texte <Parameter1>, <Parameter2>... ersetzt. Ein doppeltes Dollarzeichen '$$' wird nicht durch einen weiteren Parameter, sondern durch ein einzelnes Dollarzeichen ersetzt.

Das Ergebnis wird im Laufwerk A in einer Datei mit dem Namen $$$.SUB abgespeichert. Sollte sich auf der Diskette im Laufwerk A schon eine Datei dieses Namens befinden, wird diese gelöscht. Damit ist die Arbeit des SUBMIT-Programms schon beendet und die Kontrolle über den Computer wird dem CCP zurückgegeben.

Die erste Aktion des CCP nach einem beliebigen Aufruf ist nicht eine Tastaturabfrage, sondern die Frage, ob sich auf der Diskette im Laufwerk A eine Datei mit dem Namen $$$.SUB befindet. Diese Abfrage wird nicht nur nach dem Ende des SUBMIT-Programms durchgeführt, sondern nach jedem beliebigen Programmende. Wirklich auf der Diskette sucht der CCP aber nur nach einem Warmstart. Endete ein Programm ohne Warmstart, richtet sich der CCP nach einem intern gespeicherten Vermerk über die $$$.SUB-Datei, der beim letzten Warmstart angelegt wurde. Veränderungen der $$$.SUB-Datei werden also nur nach einem Warmstart registriert.

Findet der CCP eine $$$.SUB-Datei, wird auf die manuelle Eingabe einer Befehlszeile verzichtet: Aus der Datei wird dann die aktuelle Zeile geholt

und so betrachtet, als ob sie als Befehlszeile direkt eingegeben worden wäre. Aus der Datei wird die gelesene Befehlszeile sofort gelöscht. So wird beim nächsten CCP-Aufruf die nächste akutelle Zeile als Befehlszeile betrachtet. War die gelöschte Zeile die letzte Zeile der Datei, wird die $$$.SUB-Datei gelöscht.

Der Aufbau einer $$$.SUB-Datei

Das Prinzip der automatischen Befehle wird oft in Anwenderprogrammen verwendet, die sich gegenseitig aufrufen: Das Anwenderprogramm imitiert in diesem Fall SUBMIT.COM und legt eine $$$.SUB-Datei so an, daß sie zunächst das ausgewählte Programm und danach wieder das aufrufende Programm selbst startet.

Wenn Sie diesen Trick in eigenen Programmen verwenden wollen, müssen Sie wissen, wie eine $$$.SUB-Datei aufgebaut ist. Dazu sollten Sie zunächst mit einem beliebigen Texteditor die Datei TEST.SUB erstellen, die so aufgebaut ist:

```
DDT $$$$$$.SUB (1)
DIR *.$1 (2)
DIR *.$2 (3)
DIR *.$3 (4)
.die letzte Zeile (5)
```

Beim Aufruf dieser Datei werden hinter dem 'DDT' jeweils 2 Dollarzeichen zu einem einzigen zusammengefaßt. Tatsächlich untersucht wird also die Datei '$$$.SUB' mit nur 3 Fragezeichen. Die Zeilennummer in Klammern sollten Sie mit eingeben, obwohl sie beim Abarbeiten der Programme sinnlos sind. Nach dem Aufruf mit

```
SUBMIT TEST COM BAK ASM
```

wird zunächst das DDT-Programm gestartet, danach 'DIR *.COM', dann 'DIR *.BAK' und zuletzt 'DIR *.ASM'. Die Zeile 5 wird nur als Kommentar ausgegeben. Vorher können Sie im DDT-Programm mit 'D0100,04FF' die zur Programmliste gehörende $$$.SUB-Datei untersuchen. Daraus können Sie deren Aufbau lesen:

- Für jede Befehlszeile ist ein vollständiger 128-Byte-Record reserviert.
- Im ersten Byte des Records steht die Länge der Befehlszeile.
- Im Bereich ab dem 2. Bytes steht die Befehlszeile in Klarschrift.
- Das erste Byte hinter dem Ende der Zeile hat den Wert 0.
- Der Rest des Records hat einen zufälligen Inhalt.
- Die Befehlszeile enthält nur Großbuchstaben, auch wenn die Eingabe in der Datei TEST.SUB ursprünglich aus Kleinbuchstaben bestand.

- Die Befehlszeilen liegen in der $$$.SUB-Datei in umgekehrter Reihenfolge vor als in der zugehörigen Programmlistendatei.

Durch die Umkehrung ist die erste Befehlszeile also im letzten Record der Datei zu finden. Da der Record außer dieser Zeile keine weiteren Daten mehr enthält erleichtert das die Verwaltung sehr: Zum Löschen der gerade bearbeiteten Befehlszeile braucht nur der letzte Record aus der Datei entfernt werden. Das gelingt ohne Umkopieren, indem der Recordzähler der Datei um den Wert 1 vermindert wird und die Datei ohne weitere Veränderung wieder geschlossen wird: Dadurch ist der bisherige letzte Record nicht mehr zu lesen und der vorletzte (mit der nächsten zu bearbeitenden Befehlszeile) wird der neue letzte.

Ein Experiment zum gegenseitigen Programmaufruf

Das folgende Programm CALL.COM macht nichts anderes als vom Anwender einen Programmnamen samt Übergabeparametern zu fordern und das angegebene Programm zu starten. Ein sinnvoller Einsatz ist zum Beispiel im Stapelbetrieb möglich, wenn mitten in eine SUBMIT-Datei ein vom Anwender noch anzugebendes Programm eingeschoben werden soll.

Im Unterschied zum schon füher beschriebenen RUN.COM-Programm baut CALL.COM nur auf dem Standard-CP/M auf und ist so auf jedem beliebigen CP/M-Computer ablauffähig.

Wenn der Versuch, eine Datei $$$.SUB zu eröffnen erfolgreich ist, ist schon eine alte $$$.SUB-Datei vorhanden, die nicht gelöscht werden darf. In diesem Fall wird deren Größe bestimmt und die eingegebene Befehlszeile am Ende angehängt. Nach dem nächsten Programmende wird also das angegebene Programm gestartet und beim übernächsten Programmende die $$$.SUB-Datei ganz normal weiterbearbeitet. Wurde $$$.SUB nicht gefunden, wird sie neu erzeugt.

Die eingegebenen Daten stehen automatisch genau in der Anordnung zur Verfügung, die auch die $$$.SUB-Datei benötigt. Nur die Kleinbuchstaben müssen in Großbuchstaben umgewandelt und am Ende eine Null angehängt werden.

Das Listing zu CALL.COM

```
                ;CP/M-Programm von einem anderen CP/M-Programm
                ;  aufrufen lassen
                ;
                ;****** Konstanten
0005 =          bdos    equ     0005h
0080 =          dma     equ     0080h
0009 =          type    equ     09h
000A =          input   equ     0Ah
000F =          open    equ     0Fh
0010 =          close   equ     10h
0016 =          make    equ     16h
0022 =          write   equ     22h
0023 =          size    equ     23h
001A =          setdma  equ     1Ah
4000 =          stack   equ     4000h
0100            org     0100h
                ;
                ;****** lokalen Stapel anlegen
0100 310040             lxi     sp,stack
                ;
                ;****** Eingabe anfordern
                ;(Eingabe anfordern)
0103 118301     again:  lxi     d,inpmsg        ;Text: Eingabe anfordern
0106 0E09               mvi     c,type
0108 CD0500             call    bdos
010B 118000             lxi     d,dma
010E 3E7E               mvi     a,7Eh           ;maximale Eingabelänge
0110 12                 stax    d
0111 0E0A               mvi     c,input
0113 CD0500             call    bdos            ;Zeile in Puffer lesen
0116 218100             lxi     h,dma+1         ;Tatsächlich benötigte Länge
0119 7E                 mov     a,m
011A A7                 ana     a               ;nochmal auf Länge 0 testen
011B CA0301             jz      again
                ;(Eingabe in Großbuchstaben umformen
011E 47                 mov     b,a             ;Schleifenzähler
011F 23         loop:   inx     h               ;nächstes Zeichen
0120 7E                 mov     a,m
0121 FE7B               cpi     'ä'             ;Größer als 'z'
0123 D22E01             jnc     ok
0126 FE61               cpi     'a'             ;Kleiner als 'a'
0128 DA2E01             jc      ok
012B D620               sui     'a'-'A'         ;Korrektur
012D 77                 mov     m,a
012E 05         ok:     dcr     b               ;noch ein Zeichen
012F C21F01             jnz     loop
0132 23                 inx     h
0133 3600               mvi     m,0             ;Markierung Zeilenende
                ;
                ;****** Datei '$$$.SUB' auf letztes Zeichen eröffnen
                ;(Datei eröffnen)
0135 11E901     parada: lxi     d,efcb
0138 0E0F               mvi     c,open          ;Versuch, zu eröffnen
013A CD0500             call    bdos
013D 3C                 inr     a               ;Vergleich mit 0FFh
013E C25801             jnz     datda           ;Datei schon da
```

```
0141 11E901             lxi     d,efcb
0144 0E16               mvi     c,make          ;Datei erzeugen
0146 CD0500             call    bdos
0149 3C                 inr     a               ;Vergleich mit OFFh
014A C26001             jnz     openok          ;kein Fehler aufgetreten
                ;(Fehlermeldung ausgeben)
014D 11B901     error:  lxi     d,errmsg
0150 0E09               mvi     c,type
0152 CD0500             call    bdos
0155 C30000             jmp     0000h
                ;(letzten Record bestimmen
0158 11E901     datda:  lxi     d,efcb
015B 0E23               mvi     c,size
015D CD0500             call    bdos            ;Dateigröße
                ;(Datei beschreiben und schließen)
0160 118100     openok: lxi     d,dma+1         ;Adresse der Zeilenlänge
0163 0E1A               mvi     c,setdma        ;neuer Datenpuffer
0165 CD0500             call    bdos
0168 11E901             lxi     d,efcb
016B 0E22               mvi     c,write         ;Eingabe steht schon im
016D CD0500             call    bdos            ;Diskettenpuffer
0170 A7                 ana     a
0171 C24D01             jnz     error           ;Fehler beim Schreiben
0174 11E901             lxi     d,efcb
0177 0E10               mvi     c,close         ;Datei schließen
0179 CD0500             call    bdos
017C 3C                 inr     a
017D CA4D01             jz      error           ;Fehler aufgetreten
0180 C30000             jmp     0000h           ;Programmende
                ;
                ;****** Meldetexte
0183 4269747465inpmsg:  db      'Bitte Programmname und Übergabe'
01A2 706172616D         db      'parameter eingeben: ',13,10,'$'
01B9 2A2A2A2A2Aerrmsg:  db      '***** Fehler beim Schreiben aufgetreten '
01E1 2A2A2A2A2A         db      '*****',13,10,'$'
                ;
                ;****** Dateibeschreiber für '$$$.SUB'
01E9 01         efcb:   db      1               ;Laufwerk A
01EA 2424242020         db      '$$$     SUB'   ;11 Zeichen Dateiname
01F5 00000000           db      0,0,0,0         ;Extentbytes
01F9                    ds      16              ;beliebiger Inhalt
0209 00000000           db      0,0,0,0         ;verschiedene Zähler
020D                    end
```

Eine Verbesserung des SUBMIT-Programms

In einer Datei <Name>.SUB kann ohne Bedenken ein weiterer SUBMIT-Aufruf verwendet werden. Beim Aufruf der zusätzlichen SUBMIT-Datei wird wie üblich eine $$$.SUB-Datei angelegt, die die codierten Programmaufrufe enthält.

Dabei wird aber der Rest der alten $$$.SUB-Datei gelöscht. Die noch nicht abgearbeiteten Zeilen des ersten SUBMIT-Aufrufes gehen verloren. Ein zusätzliches SUBMIT in einer SUBMIT-Datei entspricht also einem Goto-

Befehl in einem Basic-Programm: Im angesprungenen Programmteil kann nicht mehr festgestellt werden, aus welchem Programmteil dieser aufgerufen wurde.

Nützlich wäre aber ein SUBMIT-Befehl, der nach Art eines Gosub-Befehls in Basic funktioniert: Nachdem die aufgerufene Programmliste abgearbeitet ist sollte die Bearbeitung der ursprünglichen $$$.SUB-Datei fortgesetzt werden. Dabei sollte es möglich sein, die SUBMIT-Aufrufe beliebig zu verschachteln.

Im letzten Beispiel wurde gezeigt, wie so etwas mit einer einzigen zusätzlichen Befehlszeile funktioniert: Diese wird einfach am Ende der aktiven $$$.SUB-Datei angehängt. Dieses Verfahren läßt sich auch für ganze Befehlskolonnen erweitern: Hinter das ursprüngliche Dateiende der $$$.SUB-

daß der als erstes abzuarbeitende Befehl ganz zum Schluß steht. Die 'Unterprogrammdatei' wird dann genau so abgearbeitet, als ob es sich um eine gewöhnliche Datei handeln würde. Wenn die letzte Zeile des Unterprogramms abgearbeitet ist, befindet sich die $$$.SUB-Datei wieder in genau demselben Zustand, in dem sie vor dem Unterprogrammaufruf war.

Tatsächlich kann man das originale SUBMIT-Programm mit geringem Aufwand so umbauen, daß es diese Bedingungen erfüllt. Im Bild 10-1 sind die Ausschnitte gezeigt, die im originalen SUBMIT-Programm das Löschen der alten $$$.SUB-Datei und das Erzeugen der neuen erledigen. Diese Teile beziehen sich bei Ihrer Arbeit auf den Dateikontollblock der $$$.SUB-Datei, der sich an der Adresse 05BBh befindet.

```
022D  LXI  H,05E4     025D  LXI  H,05EA     04FE  LXI  B,05BB
0230  MOV  M,B       0260  MOV  M,B       0501  CALL 022D
0231  DCX  H         0261  DCX  H         0504  LXI  H,05DB
0232  MOV  M,C       0262  MOV  M,C       0507  MVI  M,00
0233  LHLD 05E3      0263  LHLD 05E9      0509  LXI  B,05BB
0236  XCHG           0266  XCHG           050C  CALL 025D
0237  MVI  C,13      0267  MVI  C,16      050F  LDA  05DE
0239  CALL 058A      0269  CALL 058D      0512  CPI  FF
023C  RET            026C  STA  05DE      0514  JNZ  051D
                     026F  RET
```

Bild 10-1: *Ein Ausschnitt aus dem Original-SUBMIT-Programm*

Das Hauptprogramm von SUBMIT.COM an der Adresse 04FEh löscht zunächst mit Hilfe des Unterprogrammes an der Adresse 022Dh die eventuell schon vorhandene $$$.SUB-Datei, und setzt in der Adresse 0507h die aktuelle Recordnummer auf dem Wert 00h. Mit Hilfe des Unterprogramms an der Adresse 025Dh wird $$$.SUB neu erzeugt. Trat dabei ein Fehler auf, wird das Programm an der Adresse 0517h fortgesetzt, war alles in Ord-

nung, an der Adresse 051Dh. Die nicht gezeigten Unterprogramme an den Adressen 058Ah und 058Dh enthalten nur Sprungbefehle an die Adresse 0005h.

Diese 3 Teile müssen so geändert werden, daß bei einer schon vorhandenen $$$.SUB-Datei diese nicht gelöscht wird, sondern ohne gelöscht zu werden richtig fortgesetzt wird. Eine Lösung für dieses Problem zeigt Bild 10-2.

```
022D  MVI  C,23      025D  PUSH D       04FE  LXI  D,05BB
022F  CALL 025D      025E  CALL 0005    0501  MVI  C,0F
0232  MVI  C,22      0261  POP  D       0503  CALL 025D
0234  CALL 0005      0262  RET          0506  INR  A
0237  ANA  A         0263  NOP          0507  JNZ  022D
0238  JMP  0514      0264  NOP          050A  STA  05DB
023B  NOP                    .          050D  MVI  C,16
023C  NOP                    .          050F  CALL 0005
                             .          0512  ANI  FC
                       026F  NOP        0514  JZ   051D
```

Bild 10-2: *Die geänderten Bereiche des SUBMIT-Programms*

Im neuen SUBMIT-Programm wird an der Adresse 04FEh die Datei $$$.SUB zunächst versuchsweise eröffnet. War diese Operation erfolgreich, gibt es die Datei schon und der Dateizeiger muß auf die Position direkt hinter dem Dateiende gerichtet werden. Das wird im Programmteil an der Adresse 022Dh erledigt. Dazu wird mit Hilfe der BDOS-Funktion 23h die Dateigröße berechnet und im Dateibeschreiber abgelegt. Mit der BDOS-Funktion 22h werden Daten wahlfrei in den so bestimmten Record geschrieben, wodurch der Dateibeschreiber für ein sequentielles Beschreiben vorbereitet ist.

War das Eröffnen nicht erfolgreich, gibt es die Datei noch nicht und sie wird neu erzeugt. Dabei auftretende Fehler werden durch die Originalroutinen behandelt. Beachten Sie aber, daß der Befehl 'JZ 051D' an der Adresse 0514h durch einen JNZ-Befehl ersetzt wurde. Das Unterprogramm an der Adresse 025Dh ruft eine BDOS-Funktion auf, ohne das DE-Registerpaar zu verändern.

Die Änderungen nehmen Sie am besten vor, indem Sie das SUBMIT-Programm mit 'DDT SUBMIT.COM' in den Speicher laden und in die in den Bildern gezeigten Bereiche die neuen Inhalte eintragen. Das Ergebnis müssen Sie mit 'SAVE 5 DO.COM' abspeichern. DO ist der neue Name der geänderten SUBMIT-Version. So vermeiden Sie Verwechslungen mit der bisherigen Version. Das DO-Programm ist im Normalfall vollständig kompatibel zum originalen SUBMIT-Programm. Die korrekte Behandlung von SUBMIT-Unterprogrammen schränkt die bisherigen Möglichkeiten in keiner Weise ein. Selbstverständlich funktioniert DO nicht nur auf Schneider-, sondern auf beliebigen CP/M 2.2-Computern.

Der XSUB-Befehl

Mit SUBMIT können nur Programme aufgerufen werden. Die Eingaben, die das Programm verlangt, werden nicht automatisch erledigt und müssen von Ihnen weiterhin in Handarbeit eingetippt werden. Mit Hilfe des XSUB-Programmes aber kann auch das der Computer selbsttätig erledigen.

Beim Start lädt sich das XSUB-Programm direkt unter die Obergrenze der TPA, aber so daß der CCP nicht überschrieben wird. Durch Verbiegen des BDOS-Einsprunges (der gleichzeitig als Endemarkierung des freien Speichers dient) schützt es sich vor dem Überschrieben werden.

Ein Anwenderprogramm, das glaubt eine BDOS-Routine aufzurufen, ruft danach in Wirklichkeit eine Routine des XSUB-Programms auf. XSUB testet dann zunächst, welche BDOS-Funkion verlangt wird. Im allgemeinen leitet XSUB den Auftrag aber an das normale BDOS weiter. Die einzigen Ausnahmen sind die folgenden:

- Die BDOS-Funktion Nr. 10 verlangt normalerweise die Eingabe einer Zeile von der Tastatur. XSUB schaut anstelle dessen nach, ob sich auf der Diskette eine Datei $$$.SUB befindet. Wenn ja, liest es wie beim gewöhnlichen SUBMIT die aktuelle Zeile aus der Datei präsentiert diese aber dem aufrufenden Programm als Eingabezeile. Erst wenn keine $$$.SUB-Datei mehr zur Verfügung steht kann eine Eingabe von der Tastatur erfolgen.
- Zum Einlesen einer $$$.SUB-Zeile benötigt das XSUB-Programm einen Diskettenpuffer. Die BDOS-Funktion Nr. 26 SETDMA wird deshalb von XSUB 'abgehört', damit es immer über die aktuelle Lage des Diskettenpuffers Bescheid weiß. Vor dem Einlesen einer $$$.SUB-Zeile wird der Diskettenpuffer in einen lokalen Bereich gelegt und danach wieder auf die alte Adresse gesetzt.

Leider werden ungepufferte Tastatureingaben über die BDOS-Funktionen 1 oder 6 nicht von XSUB abgefangen. Eingaben, die mit diesen Funktionen programmiert wurden, können so nicht automatisch erledigt werden.

Damit das XSUB-Programm bei einem Warmstart nicht gelöscht wird, werden auch Warmstarts unterdrückt. Dazu wird der Warmstart-Einsprung an der Adresse 0000h geändert. Leider wird dabei die BIOS-Sprungtabelle nicht an den Ort der neuen Warmstartroutine kopiert. Ein Anwenderprogramm hat dann (außer bösen Tricks) keine Möglichkeit mehr, die Lage der BIOS-Sprungtabelle festzustellen. Anwenderprogramme, die betrieben werden während XSUB-aktiv ist, dürfen also keine direkten BIOS-Einsprünge verwenden.

Die Veränderung der $$$.SUB-Datei durch ein Anwenderprogramm wird vom Betriebssystem nur nach einem Warmstart registriert. Eine SUBMIT-Datei darf also bei eingeschaltetem XSUB weder mit SUBMIT noch mit DO eine SUBMIT-Unterdatei aufrufen.

Ein Zusatz zum XSUB-Befehl

Nachdem der XSUB-Modus einmal eingeschaltet wurde, bleibt er normalerweise bis zum Ende der $$$.SUB-Datei aktiv. Vor einem Unterdatei-Aufruf mit DO darf XSUB aber nicht eingeschaltet sein. Wenn man in solchen Fällen trotzdem nicht auf den XSUB-Modus verzichten möchte, ist eine Routine nötig, mit der man den XSUB-Modus für bestimmte Befehle wieder abschalten kann und erst bei Bedarf wiedereinschalten. Das ist auch sinnvoll, wenn einige Programme automatisch abgearbeitet werden sollen, Zwischendurch aber wieder Anwendereingaben möglich sein sollen.

Bei einem Warmstart werden der Warmstart- und der BDOS-Einsprung auf die Originalwerte gesetzt und damit der XSUB-Modus gelöscht. Der normale Warmstart-Einsprung über die Adresse 0000h wird aber von der XSUB-Routine abgefangen.

Nicht abgefangen wird aber der Warmstarteinsprung mit Hilfe der BDOS-Funktion Nr. 0: Das ist die Warmstartanforderung der höchsten Priorität. Mit dieser Funktion können Sie auch hier noch die 'Notbremse' ziehen.

Das Programm XSUBEND.COM besteht also nur aus einem einzigen BDOS-Aufruf:

```
0100  MVI  C,00
0102  CALL 0005
```

Dieses Programm können Sie mit Hilfe des DDT und 'SAVE 1 XSUBEND.COM' leicht eingeben.

10.2 Der PIP-Befehl

Außer den 4 Standardkanälen CON:, RDR:, PUN: und LST: gibt es im PIP-Programm noch die Sonderkanäle INP: und OUT:. Im allgemeinen sind diese Geräte jedoch nicht implementiert: Der INP-Kanal liefert immer den Wert 1Ah und ein an den OUT-Kanal geschicktes Zeichen wird nicht beachtet.

Die beiden Kanäle sind aber für besondere nicht-standardgemäße Anwendungen reserviert. In den CPC-Computern wäre es zum Beispiel sinnvoll, diesen Kanälen eine Kassetten-Ein-/Ausgabe zuzuordnen. Für die Implementierung dieser Geräte steht im PIP-Programm eine interne standardisierte Anwenderschnittstelle zur Verfügung:

- Bei einer Eingabeanforderung über den INP-Kanal ruft das PIP-Programm ein Unterprogramm an der Adresse 0103h auf. Dort muß ein Sprung zur eigentlichen, nicht-standardgemäßen Eingaberoutine stehen. Das gelesene Zeichen muß die Routine in der Speicherstelle 0109h ablegen, bevor es die Kontrolle an das aufrufende Programm zurückgibt.
- Soll ein Zeichen auf den OUT-Kanal ausgegeben werden, wird ein Unterprogramm an der Adresse 0106h aufgerufen. Im CPU-Register C steht zu diesem Zeitpunkt der Code des auszugebenden Zeichens. In die Adresse 0106h muß ein Sprung zur eigentlichen Eingaberoutine eingetragen werden.
- Der Bereich von 010Ah bis 01FFh wird vom Standard-PIP nicht benutzt. Dort können die eigentlichen Ein-/Ausgaberoutinen abgelegt werden.

Am besten laden Sie zu einer Änderung das PIP-Programm mit 'DDT PIP.COM', fügen die Ergänzungen ein und speichern das Ganze unter einem anderen Namen mit SAVE ab.

11 Verbindungen zwischen Basic und CP/M

11.1 Dateiformate im Basic-System

Ascii-Dateien

Unter AMSDOS erstellte ASCII-Dateien sind zu unter CP/M erstellten ASCII-Dateien absolut äquivalent. Sie können also ohne weiteres ASCII-Dateien, die sie unter AMSDOS erstellt haben, unter CP/M weiterverwenden. Probieren Sie das einmal aus:

Schreiben Sie das folgende Basic-Programm und starten Sie es:

```
10 OPENOUT"ascii.dat"
20 FOR i%=1 TO 100
30 PRINT#9,"Dies ist die Zeile";i%
40 NEXT
50 CLOSEOUT
```

Gehen sie in den CP/M-Modus und geben Sie ein 'TYPE ASCII.DAT'. Sie sehen - es funktioniert. Umgekehrt können Sie genauso jede unter CP/M, zum Beispiel mit ED erstellte Ascii-Datei mit

```
10 OPENIN"ascii.dat"
20 WHILE NOT(EOF)
30 INPUT#9,a$:PRINT a$
40 WEND
50 CLOSEIN
```

auch während des Arbeitens mit Basic betrachten oder weiterverarbeiten.

Wenn Sie ein gutes, aber in Basic geschriebenes Textverarbeitungsprogramm haben, können Sie also damit ohne weiteres Ihre 8080-Assemblerprogramme tippen und brauchen sich nicht mehr mit dem schlechten ED-Programm zufriedenzustellen. Das gilt sogar für Basic-Programme: Wenn diese mit der Option ',a' abgespeicert wurden, werden sie als normale Ascii-Texte auf der Diskette abgelegt. Genausogut kann der Basic-Interpreter Basic-Zeilen im Ascii-Format einlesen.

Wenn Sie sich daran halten, können Sie Ihre Basic-Programm mit jedem beliebigen Textverarbeitungsprogramm erstellen. Gelegentlich müssen Sie die Datei aber vorher noch mit PIP umspeichern: Mit der PIP-Option 'Z' werden alle High-Bits der Zeichen gelöscht und mit der Option 'N2' erhalten alle Zeilennummern. Da auch in 8080-Assemblerprogrammen Zeilen-

nummern erlaubt sind, können Sie umgekehrt sogar Ihre Assemblerlistings als Basic-Programm eintippen und dann mit 'SAVE "<name>",a' abspeichern. Wenn Sie keinen guten Texteditor besitzen ist das durchaus eine Alternative.

Wenn Sie im Basic-Betriebssystem ASCII-Dateien auf die Diskette ausgeben, werden die Daten einfach unverändert auf die Diskette geschrieben. Genauso verfahren auch alle BDOS-Funktionen. Das Basic-Betriebssystem betrachtet so in der Regel jede Datei, die nicht ganz speziellen Bedingungen genügt als Ascii-Datei. Es ist deshalb ohne weiteres möglich,etwa eine COM-Datei mit OPENIN zum Lesen zu eröffnen. Natürlich wird dann beim Ausgeben auf dem Bildschirm nur Unsinn angezeigt.

Basic-Programme

Im Gegensatz zu Ascii-Dateien müssen Basic-Programme noch Informationen enthalten, die im Listing nicht erscheinen: Über die Lage des Programms im Speicher, die Programmlänge und ob das Programm geschützt ist oder nicht.

Im Directory der Diskette ist kein Platz, diese Informationen unterzubringen. Die Entwickler des Disketten-ROMs haben deshalb zu einem Trick gegriffen: Ähnlich zum File-Header bei auf Kassette aufgezeichneten Programmen wird jedem Basic-Programm bei der Aufzeichnung ein weiterer 128-Byte-Record spendiert. Dieser enthält die zusätzlichen Informationen und erscheint nicht im Listing. Die ersten 64 Bytes dieses Records sind sogar völlig identisch zum Fileheader, der sich bei einer Kassettenaufzeichnung ergeben würde.

Damit wird aber auch eine Routine nötig, die verhindert, daß der erste Block einer Ascii-Datei irrtümlich als File-Header eines Basic-Programms interpretiert wird und umgekehrt. Dieses Problem tritt zum Beispiel auf, wenn entschieden werden soll, ob ein zu ladendes Basic-Programm nun mit der Option ',a' abgespeichert wurde oder nicht. Da in einer Ascii-Datei nirgends Platz ist, ein entsprechendes Kennzeichen unterzubringen, wurde zu einer anderen Methode gegriffen: Bei jeder Dateieröffnung wird zunächst die Summe über die ersten 67 Bytes des ersten Records gebildet und mit der 16-Bit-Zahl im 68./69. Byte verglichen. Basic-Programme werden nun genau so abgespeichert, daß sich eine Gleichheit ergibt.

Ergibt sich beim Eröffnen einer Datei keine Übereinstimmung, so handelt es sich mit Sicherheit um eine Ascii-Datei. Stimmen die beiden Prüfsummen aber überein, so liegt mit einer Wahrscheinlichkeit von 65535 / 65536 = 99.9985% (die Prüfsumme könnte ja doch nur zufällig stimmen) eine Basic- oder eine Bin-Datei vor.

Maschinenspracheprogramme und binäre Dateien

Die Aufzeichnung einer binären Datei entspricht völlig der Aufzeichnung eines Basic-Programms: Den eigentlichen Programmdaten wird ebenfalls der auf 128 Bytes erweiterte Fileheader vorangestellt, nur mit dem Unterschied, daß darin das Kennzeichen für den Dateityp auf 'Bin' gesetzt ist.

Der Aufbau des Fileheaders bei Diskettendateien

Durch ein Experiment können Sie sich leicht von der Existenz dieses Fileheaders überzeugen:

- Füllen sie (im Basic-Betriebssystem) einen Speicherbereich mit lauter auffallenden Zeichen, zum Beispiel durch:
  ```
  FOR i%=&8000 TO &83FF:POKE i%,CHR$("#"):NEXT
  ```
- Speichern Sie diesen Bereich als Bin-Datei auf Diskette ab:
  ```
  SAVE "LEER.BIN",b,&8000,1024
  ```
- Wenn Sie jetzt das Inhaltsverzeichnis der Diskette betrachten, können Sie schon den ersten Hinweis auf den Header-Record entdecken: Die Datei belegt nicht (wie zu vermuten wäre) 1 KByte, sondern 2 KByte.
- Gehen Sie jetzt in den CP/M-Modus und sehen sich die Datei mit dem DDT-Programm an. Die Zeichen '#' liegen jetzt im Speicherbereich von 0180h bis 057Fh. Im Bereich von 0100h bis 017f können Sie direkt den langgesuchten Header betrachten. Bild 11-1 zeigt dessen Aufbau.

Alle hier nicht erwähnten Bytes im Bereich von 0100h bis 017fh werden bei der Diskettenaufzeichnung nicht benutzt und können von Ihnen beliebig verwendet werden. (Vielleicht, um einen Copyrightvermerk einzutragen, den man in Basic nicht lesen kann?)

0100h-010Fh	Diese Bytes enthalten den Programmnamen LEER.BIN
0112h	Hier steht der Code für den Dateityp. 00h für ein Basic-Programm, 02h für ein Maschinenprogramm und 01h für ein geschütztes Basic-Programm.
0115h-0116h	Enthält die Adresse, an der die Daten ursprünglich standen. In diesem Falle also den Wert 800h.
0118h-0119h	In diesen Speicherzellen steht die Programmlänge. Hier also der Wert 400h.
011Ah-011Bh	Diese Bytes enthalten eine eventuell angegebene Startadresse für ein Maschinenprogramm.
0143h-0144h	Diese Bytes enthalten die Summe der Bytes im Speicherbereich von 0100h bis 0142h.

Bild 11-1: *Aufbau des Fileheaders für binäre und Basic-Dateien*

11.2 Transfer von Daten zwischen Basic und CP/M

Basic-Programme und Ascii-Dateien

Ascii-Dateien sind in Basic und unter CP/M in genau der gleichen Weise organisiert. In Basic mit OPENOUT ganz normal abgespeicherte Ascii-Dateien können ohne weitere Vorkehrungen unter CP/M auf die unter CP/M übliche Art weiterverarbeitet werden. Dasselbe gilt auch für den umgekehrten Weg.

Die Übertragung von codierten Basic-Programmen zwischen Basic und CP/M ist nicht sinnvoll: Die Basic-Tokens haben nur für den Basic-Interpreter einen Sinn.

Es ist aber möglich, Basic-Programme mit der Option ',a' abzuspeichern. In diesem Fall wird das Basic-Programm zu einer reinen Ascii-Datei und kann unter CP/M ohne jede Vorkehrung verarbeitet werden. Wenn sie nicht gerade die exotischsten Basic-Befehle verwendet haben, ist dann sogar die Übertragung von Basic-Progrmmen zwischen verschiedenen Computertypen möglich.

Übertragung von Maschinenspracheprogrammen

Umwandeln einer BIN-Datei in eine COM-Datei

Die Umwandlung eines in Basic erstellten Maschinenprogrammes in eine COM-Datei ist nur sinnvoll, wenn das Maschinenprogramm ab der Adresse 0100h ablauffähig ist. In der Regel kommt diese Möglichkeit also nur in Betracht, wenn Sie einen guten und im Basic-System betriebenen Assembler besitzen und ein Programm speziell für die spätere Verwendung unter CP/M entwickeln. Das gilt natürlich auch für Programme, die Sie früher entwickelt haben und sich nur durch Änderung einiger weniger Assemblerzeilen für die Verwendung unter CP/M anpassen lassen.

1. Möglichkeit:

Speichern Sie das zu übertragende Maschinenprogramm wie für BIN-Dateien üblich mit 'SAVE "<Name>.BIN",b' oder einer gleichwertigen Funktion ab.

Wenn Sie die so entstandene Datei unter CP/M laden, stört nur der 128 Bytes lange Fileheader. Diesen können Sie so entfernen:

- Starten Sie den DDT.
- Geben Sie den Namen der BIN-Datei an: 'I<Name>.BIN'
- Laden Sie die Datei mit 'RFF80'. FF80h entspricht dem Wert -0080h und gibt den Versatz an, um den die Datei beim Laden verschoben wird. Der Fileheader beginnt jetzt also an der Adresse 0080h und die eigentlichen Daten an der Adresse 0100h. Durch 'SAVE <Länge> <Name>.COM' speichern Sie den Bereich ab 0100h ab, also genau die Daten die Sie brauchen- ohne den lästigen Fileheader.

2. Möglichkeit:

Eine COM-Datei unterscheidet sich im Aufzeichnungsformat nicht von einer Ascii-Datei. Bei entsprechender Wahl der zu speichernden Daten können Sie also eine COM-Datei durch OPENOUT... erzeugen:

- Laden Sie die als COM-Datei zu speichernde BIN-Datei in einen beliebigen unbelegten Speicherbereich von der Adresse <Anfang> bis <Ende>.
- Jetzt geben Sie diese Programmzeile ein:

```
OPENOUT"<Name>.COM":FOR i=<Anfang>TO <Ende>:
PRINT#9,CHR$(PEEK(i));:NEXT:CLOSEOUT
```

Die so entstandene Datei mit der Bezeichnung '.COM' ist genau so aufgebaut, wie eine auf 'klassischem Wege' erzeugte COM-Datei.

Umwandeln einer COM-Datei in eine BIN-Datei

Einer COM-Datei fehlt im Vergleich zur BIN-Datei der Dateiheader. Diesen können Sie aber mit einem Trick ganz leicht erzeugen:

- Eine COM-Datei hat die Länge <Länge> und soll beim Einladen als BIN-Datei später an der Adresse <Ziel> abgelegt werden.
- Speichern Sie jetzt im Basic eine leere BIN-Datei, aber mit der richtigen Länge ab: 'SAVE "<Name>.BIN",b,<Ziel>,<Länge>'.
- Gehen Sie jetzt in den CP/M-Modus und starten Sie den DDT zur Untersuchung der BIN-Datei: 'DDT<Name>.BIN'. Im Speicher befindet sich dann ab der Adresse 0100h schon der gesuchte Fileheader.
- Jetzt laden Sie die COM-Datei mit Hilfe des R<Versatz>-Befehles an die Adresse 0180h: Das COM-Programm befindet sich zusammen mit dem Fileheader im Speicher.
- Jetzt unterbrechen Sie das DDT-Programm und speichern die BIN-Datei mit dem CP/M-Befehl SAVE ab. In Basic kann diese Datei dann ganz normal verwendet werden.

12 Nützliche Routinen zum Arbeiten unter CP/M

12.1 Allgemeines zu den Anwendungsprogrammen

Die folgenden Programme sind jeweils in 2 Versionen abgedruckt: als Assembler-Listing und als Basic-Lader. Wenn Sie Assembler beherrschen, sollten Sie auf alle Fälle das Assemblerlisting abtippen: Dann können Sie ohne großen Aufwand nach Lust und Laune Änderungen anbringen. Zum Abtippen brauchen Sie die Programme nicht unbedingt bis in letzte Detail verstehen: Sie müssen nur das Assemblerprogramm starten können.

Wenn Sie sich nicht an die Assemblerprogramme wagen, können Sie wahlweise auch die Basic-Version eintippen. Beim Start eines dieser Basic-Programme werden die Zahlen aus den Datazeilen in das Format einer COM-Datei umgewandelt und auf der Diskette abgelegt. Hat das fehlerfrei geklappt, brauchen Sie das Basic-Programm nicht mehr. Zum Anwenden des Programms genügt die erzeugte COM-Datei.

Die 17. Zahl einer Data-Zeile eines Basic-Laders ist die Summe der ersten 16 Zahlen. Wenn Sie beim Abtippen einen Fehler gemacht haben, stimmen die Summen nicht mehr überein. Beim Abarbeiten wird das vom Lader bemerkt und eine Fehlermeldung ausgegeben: Sie können den Basic-Lader jetzt verbessern. Nicht automatisch erkannt wird aber eine vollständig vergessene Zeile oder das Vertauschen zweier benachbarter Zeilen: In diesem Fall wird ohne Warnung eine fehlerhafte COM-Datei erzeugt. Wenn das Programm nicht läuft, sollten Sie also zunächst nur nach vergessenen Zeilen suchen, ohne sie im Detail zu betrachten. Auch wenn die letzte Zeile eines Programmes lauter Nullen enthält, dürfen Sie sie nicht weglassen. In der Regel enthalten diese Zeilen bestimmte Voreinstellungen für das COM-Programm.

Wenn Sie zu einem Programm nicht den Basic-Lader, sondern das Assemblerlisting eingeben wollen, sollten Sie die ersten Übersetzungen durch 'ASM <Name>.AZZ' aufrufen. Dadurch erreichen Sie eine schnellere Assemblierung. Dafür wird das Ergebnis aber nicht auf der Diskette abgelegt. Für den Anfang schadet das aber nicht, denn kaum ein Programm ist von vornherein frei von Tippfehlern.

Mit welcher Methode Sie ein Programm eintippen, bleibt vollständig Ihnen überlassen. Wenn Sie einen guten Texteditor besitzen, können Sie den Ba-

sic-Lader ruhig damit eintippen und dessen gute Editierhilfen ausnützen. Das erzeugte Basic-Programm läßt sich trotzdem ganz normal laden.

In Ermangelung eines Texteditors können Sie auch Assemblerprogramme so eintippen, als ob es Basic-Programme wären. Das Ergebnis müssen Sie mit 'SAVE "<name>.ASM",a' abspeichern. Die so entstandene Datei verarbeitet der CP/M-Assembler problemlos und ohne weitere Vorkehrungen. Die beim Eintippen in Basic entstehenden Zeilennummern stören ihn nicht.

Das Programm UNLOAD.BAS

Selbstverständlich wurden die Basic-Lader nicht von Hand erstellt. Zur Unterstützung wurde das Programm UNLOAD.BAS verwendet.

UNLOAD.BAS ist ein Beispiel für ein Basic-Programm, das 'selbständig' ein anderes Basic-Programm schreibt. Es lädt eine HEX-Datei in den Speicher und formt die darin enthaltenen Daten so um, daß sie wie Basic-Datazeilen aussehen. Diese Zeilen werden unter dem Namen eines Basic-Programms abgespeichert. Dieses neue Programm wird auf dieselbe Weise um einige Zeilen ergänzt, die die DATA-Zeilen verarbeiten und als COM-Datei ablegen: Dann ist der BASIC-Lader für ein COM-Programm fertig.

Die HEX-Datei, die UNLOAD.BAS als Quelldatei benötigt, muß in jeder Zeile Daten für genau 16 Bytes der COM-Datei enthalten. Nur die letzte Zeile darf weniger als 16 Bytes enthalten. Die Einhaltung dieser Regel wird aber nicht überprüft: Wenn eine HEX-Zeile mehr Daten als notwendig enthält, werden die überschüssigen weggelassen. Enthält eine HEX-Zeile nicht ausreichend Daten, wird sie mit Nullen auf die Normlänge gestreckt. Wenn diese Fälle mitten in einer HEX-Datei auftreten, sind die entstandenen BASIC-Lader sinnlos. Nur am Ende schadet das Strecken einer Zeile nicht.

Diese Bedingungen werden in der Regel von den durch den CP/M-Assembler erstellten HEX-Dateien eingehalten. Das gilt jedoch nur, wenn das Assembler-Quellisting in der Mitte keine DS-Befehle enthält. Denn die DS-Bereiche werden in der HEX-Datei nicht vermerkt. Am Ende eines Programms schaden die DS-Befehle dagegen nicht. Wenn Sie vorhaben, ein selbsterstelltes Assemblerprogramm mit UNLOAD.BAS weiterzuverarbeiten, sollten Sie alle DS-Befehle am Ende des Programms zusammenfassen. Wenn das nicht möglich ist, müssen Sie die DS-Befehle durch 'DW 0,0,0...' oder "DB '.....'" ersetzen.

Das Listing von UNLOAD.BAS

```
10000 'Uebersetzt eine HEX-Datei in einen Basic-Lader; mit diesem kann ohne
10010 '  Maschinensprachekennnisse eine COM-Datei erstellt werden
10020 '
10030 'Vorbereitung
10040 INPUT"Programmname(ohne Namensextension)";n$
10050 OPENIN n$+".hex"
10060 OPENOUT n$+".bas"
10070 '
10080 'Jetzt wird das eigentliche Basic-Lader-Programm erzeugt
10090 PRINT#9,"100 'Basic-Lader fuer die Datei '"UPPER$(n$)".COM'"
10100 PRINT#9,"110 '"
10110 PRINT#9,"120 OPENOUT"CHR$(34)UPPER$(n$)".COM"CHR$(34)
10120 PRINT#9,"130 '"
10130 PRINT#9,"140 z%=10000"
10140 PRINT#9,"150 READ d$:WHILE d$<>"CHR$(34)"Ende"CHR$(34)
10150 PRINT#9,"160 d%=VAL("CHR$(34)"&"CHR$(34)"+d$):PRINT#9,CHR$(d%);:p%=d%"
10160 PRINT#9,"170 FOR i%=1 TO 15
10170 PRINT#9,"180 READ d$:d%=VAL("CHR$(34)"&"CHR$(34)"+d$)"
10180 PRINT#9,"190 PRINT#9,CHR$(d%);:p%=p%+d%"
10190 PRINT#9,"200 NEXT"
10200 PRINT#9,"210 READ q%"
10210 PRINT#9,"220 IF p%<>q% THEN PRINT"CHR$(34)"Tippfehler in Zeile "
      CHR$(34)"z%:CLOSEOUT:END"
10220 PRINT#9,"230 z%=z%+10"
10230 PRINT#9,"240 READ d$:WEND"
10240 PRINT#9,"250 CLOSEOUT"
10250 PRINT#9,"260 PRINT"CHR$(34)"Alles O.K."CHR$(34)
10260 PRINT#9,"270 END"
10270 PRINT#9,"280 '"
10280 '
10290 'Die Hex-Datei kommt in Form von Datas ins erzeugte Basic-Programm
10300 '
10310 'Schleife: 1 Zeile aus der HEX-Datei lesen
10320 z%=10000
10330 INPUT#9,q$:WHILE NOT(EOF)
10340 '
10350 'Hexzeile auf Normlänge und Ausgabezeile vorbereiten
10360 IF LEN(q$)<43 THEN q$=LEFT$(q$,LEN(q$)-2)+STRING$(45-LEN(q$),"0")
10370 PRINT#9,USING"#####";z%;:z%=z%+10
10380 PRINT#9," DATA ";
10390 '
10400 'Datas mit Pruefsumme ausgeben
10410 p%=0
10420 FOR i%=10 TO 41 STEP 2
10430 p%=p%+VAL("&"+MID$(q$,i%,2)):PRINT#9,MID$(q$,i%,2)",";
10440 NEXT
10450 PRINT#9,USING"####";p%
10460 '
10470 INPUT#9,q$:WEND
10480 '
10490 'Endemarkierung ausgeben
10500 PRINT#9,USING"#####";z%;
10510 PRINT#9," DATA Ende"
10520 CLOSEOUT
10530 CLOSEIN
10540 END
```

12.2 Programme für beliebige CP/M-Computer

Die beiden folgenden Programme sind reine Standard-CP/M-Programme und deshalb auf jedem beliebigen CP/M-Computer lauffähig. Dafür werden aber zusätzliche Möglichkeiten der Schneider-Computer nicht ausgenützt.

12.2.1 Der Menue-Generator

Anhand einer Programmliste erstellt der Menue-Gemerator ein Auswahlmenue und zeigt es auf dem Bildschirm an. Mit einem Tastendruck kann ein beliebiges Programm aus der Programmliste aufgerufen werden. Nach Ausführung dieses Programms wird automatisch das Menue erneut angezeigt. Durch einige zusätzliche Extras, die das Programm bietet, kann der Computer so konfiguriert werden, daß er wirklich von reinen 'Nur-Anwendern' bedient werden kann, die vom Umgang mit CP/M keine Ahnung haben.

Dem Menue-Generator wird erst beim Aufruf der Name einer Datei übergeben, in der - ähnlich einer SUBMIT-Datei - die Namen der aufzurufenden Programme stehen. Der Menue-Generator kann deshalb ohne Änderung verschiedene Menues erzeugen. Die aufzurufenden Programme brauchen in keiner Weise für die Verwendung durch den Menue-Generator vorbereitet werden: Auch bei den einfachsten CP/M-Programmen wird bei der Beendigung automatisch das Menue wieder aufgerufen.

```
A>type haupt.men
Hauptmenue
Experimentieren mit relativen Dateien
RAND2
Dateien kopieren
PIP
Disketteninhaltsverzeichnis anzeigen
DO INHALT
Datei '$$$.SUB' untersuchen
DO SUBTEST
Alle hier benötigten Dateien auflisten
DO LISTEN
Untermenue 'Dateipflege' aufrufen
MENUE UNTER.MEN
Nur als Gag, aber möglich: Rekursiver Menue-Aufruf
MENUE HAUPT.MEN
Beliebigen Befehl eingeben
CALL
```

Bild 12-1: *Eine Programmliste für den Menue-Generator*

Die Funktionsweise des Menue-Generators

Bild 12-1 zeigt am Beispiel der Datei HAUPT.MEN, wie eine Programmliste für den Menue-Generator aufgebaut sein kann. Bild 12-2 zeigt den Bildschirm des Computers so, wie er nach dem Aufruf des Menue-Generators mit 'MENUE HAUPT.MEN' zu sehen ist. Dieses Beispiel sollten Sie ruhig einmal ausprobieren, um alle Möglichkeiten des Menue-Generators zu verstehen.

```
A>menue haupt.men
                              Hauptmenue                          Ebene: 1
------------------------------------------------------------------------------

        (1)    Experimentieren mit relativen Dateien

        (2)    Dateien kopieren

        (3)    Disketteninhaltsverzeichnis anzeigen

        (4)    Datei '$$$.SUB' untersuchen

        (5)    Alle hier benötigten Dateien auflisten

        (6)    Untermenue 'Dateipflege' aufrufen

        (7)    Nur als Gag, aber möglich: Rekursiver Menue-Aufruf

        (8)    Beliebigen Befehl eingeben

        (E)    Menue-Ende


Bitte wählen Sie: e                          (c) 1985 by Isar-Amper-Soft
```

Bild 12-2: *So präsentiert sich der Menue-Generator*

Die Programmliste für den Menue-Generator muß also so aufgebaut sein:

- In der ersten Zeile steht der Name des Menues. Dieser wird in der Kopfzeile des Bildschirmbildes angezeigt. Er darf eine beliebige Länge haben, bei der Ausgabe werden aber nur die ersten 72 Zeichen beachtet. Steuerzeichen sollten in dieser Zeile nicht enthalten sein, da sie vom Programm nicht unterdrückt werden.
- Ab der 2. Programmlistenzeile sind paarweise jeweils 2 Zeilen für einen Menuepunkt zuständig: Die erste Zeile eines Paares enthält nur den Text, der als Kommentar auf dem Bildschirm dargestellt wird. Für diese Zeile gelten dieselben Beschränkungen wie für den Namen des Menues. Erst die 2. Zeile eines Paares enthält den eigentlichen Text

des Programmaufrufes. Diese Befehlszeile muß so aufgebaut sein wie bei der direkten Eingabe über die Tastatur.

- Im Menue finden maximal 9 Programmaufrufe Platz. Die Programmliste kann zwar länger sein, die überschüssigen Zeilen werden aber nicht beachtet. Eine Leerzeile wird als vorzeitiges Ende der Programmliste betrachtet.

Vor der Ausgabe des Bildes wird die maximale Zeilenlänge der zu zeigenden Zeilen bestimmt. Die Ausgabe erfolgt dann auf die Bildschirmmitte zentriert. Es kommt also nicht vor, daß eine kurze Zeile ganz am linken oder eine lange Zeile am rechten Rand 'klebt'.

Die Steuerung der verschiedenen Programmaufrufe wird vollständig über $$$.SUB-Dateien abgewickelt. Eine eventuell schon vorher vorhandene $$$.SUB-Datei wird nicht gelöscht. Dadurch können zum Beispiel die Aufrufe des Menue-Generators beliebig verschachtelt werden: Durch 'MENUE <Untermenue>' wird ein in der Datei <Untermenue> beschriebenes Menue angezeigt. Die dort getätigten Programmaufrufe werden einfach an die $$$.SUB-Datei angehängt. Deshalb erscheint nach dem Abschluß des Untermenues wieder das Hauptmenue. Diese Aufrufe können ruhig über mehrere Stufen gehen. Obwohl es wahrscheinlich in der Praxis nicht sinnvoll ist, können sogar rekursive Menue-Aufrufe getätigt werden, etwa wenn ein Untermenue statt eines weiteren Untermenues wieder das Hauptmenue aufruft. Der Menuegenerator 'verhaspelt' sich dabei nicht. Damit der Anwender immer weiß, wieviele Untermenuestufen aktiv sind, wird eine entsprechende Zahl auf dem Bildschirm ausgegeben. Dazu wird ein Zähler bei jedem Untermenueaufruf um den Wert 1 erhöht und mit an das Untermenue übergeben.

```
A>type inhalt.sub
DIR
STAT
:BITTE EINE TASTE DRUECKEN
PIP

A>type subtest.sub
XSUB
:WEGEN EINES FEHLERS IM DDT-PROGRAMM DARF DDT
:  NICHT MIT EINEM TASTENDRUCK ABGEBROCHEN WERDEN !!!
DDT $$$$$$.SUB
D0100,04FF
GO
XSUBEND
```

Bild 12-3: *Zwei im Menue nützliche Programme*

Mit Hilfe des schon früher erstellten Programms DO.COM können aus einem Menue sogar SUBMIT-Programme aufgerufen werden. Das originale SUBMIT.COM-Programm kann dazu leider nicht verwendet werden, denn es würde die Datei $$$.SUB mit den noch benötigten Menue-Aufrufen zerstören. Als Beispiel sehen Sie im Bild 12-3 die Programme SUBTEST.COM und INHALT.COM: Mit deren Hilfe können Sie im oben gezeigten Menue-Beispiel die $$$.SUB-Datei untersuchen und ein erweitertes Inhaltsverzeichnis ausgeben. Dabei finden auch die Befehle XSUB und XSUBEND Verwendung.

Beim DDT.COM müssen Sie aber vorsichtig sein: Nachdem Sie DDT mit einem Tastendruck abgebrochen haben, befindet sich im Tastenpuffer noch ein nicht gelesenes Zeichen. Das ist ein Fehler im DDT-Programm. Beim nächsten durch $$$.SUB gesteuerten Programmaufruf wird dieses Zeichen fälschlicherweise ausgelesen. Für den CCP ist das ein Signal, die $$$.SUB-Datei zu löschen. Egal aus welcher Untermenuestufe Sie den DDT aufgerufen haben, befinden Sie sich dann wieder im Eingabe-Modus des CP/M-Systems.

Wenn beim Aufruf kein Laufwerksname für die Programmliste vorgegeben wurde, sucht der Menue-Generator die Programmliste immer auf dem Laufwerk A. Das gilt auch, wenn das Laufwerk B das Bezugslaufwerk ist. Dadurch können Sie auch während eines Menuedurchlaufes die Diskette im Laufwerk B auswechseln. Das Auswechseln der Diskette im Laufwerk A ist ohnehin nicht sinnvoll, da sich auf dieser ja die aktive $$$.SUB-Datei liegt.

Ordnungsgemäß beenden können Sie die angezeigte Menue-Stufe durch Eingabe von 'e' oder 'E'. Dann wird sinnvollerweise das noch aktive Menue der nächsthöheren Stufe aufgerufen. Obwohl nicht auf dem Bildschirm angezeigt, können Sie aber auch ^C eingeben. In diesem Fall wird der Menue-Generator vollständig abgebrochen: Die $$$.SUB-Datei und der XSUB-Modus werden gelöscht und eine Fehlermeldung ausgeben. Danach können Sie wieder Eingaben über die Tastatur tätigen, ohne sich erst durch alle Untermenuestufen an die Oberfläche zu hangeln.

Die Zugriffe auf den Menue-Generator vor und nach den Programmaufrufen können sehr beschleunigt werden, wenn die dazu nötigen Dateien möglichst nahe am Inhaltsverzeichnis der Diskette abgelegt werden. Auf einer vollständig leeren Diskette können Sie das erreichen, indem Sie die als allererstes 'SAVE 1 DUMMY' eingeben. Dadurch wird der allererste Block der Diskette belegt. Als nächstes sollten Sie das MENUE.COM-Programm abspeichern, dann die vom Menuegenerator benötigten Programmlisten und erst danach die Anwendungsprogramme. Wenn Sie jetzt die Datei DUMMY löschen, wird der erste Block der Diskette wieder frei. Beim

Menueaufruf wird die $$$.SUB-Datei in diesem Block abgelegt. Sowohl das Menue-Programm selbst als auch die $$$.SUB-Datei liegen damit auf der Inhaltsverzeichnisspur, so daß das Abarbeiten der Menues ohne Spurwechsel möglich ist.

Sinnvoll ist auch, in die Menue-Programmliste das früher besprochene Programm CALL.COM zu integrieren. Dann können auch außerplanmäßige Programmaufrufe erledigt werden, ohne das Menue extra zu verlassen.

Das Assembler-Listing zum Menue-Generator

```
                ;Menuegesteuertes gegenseitiges Aufrufen von
                ;       CP/M-Programmen
                ;
000C =          cls     equ     12              ;Code für Bildschirm löschen
                ;****** BDOS-Konstanten
0000 =          boot    equ     0000h
0005 =          bdos    equ     0005h
005C =          fcb     equ     005Ch
0006 =          conio   equ     06h             ;Direkte Konsoleneín-/ausgabe
0009 =          type    equ     09h             ;Zeichenkette ausgeben
000F =          open    equ     0Fh             ;Datei eröffnen
0010 =          close   equ     10h             ;Datei schließen
0013 =          delete  equ     13h             ;Datei löschen
0014 =          rdsequ  equ     14h             ;sequentielles Lesen
0016 =          make    equ     16h             ;Datei erzeugen
001A =          setdma  equ     1Ah             ;Datenpufferadresse wählen
0022 =          wrrand  equ     22h             ;wahlfreies Schreiben
0023 =          size    equ     23h             ;Dateigröße bestimmen
0100            org     0100h
                ;
                ;****** Initialisierung: Lokalen Stapel anlegen
0100 210000             lxi     h,0
0103 39                 dad     sp
0104 22D405             shld    oldstk
0107 31D405             lxi     sp,oldstk
                ;
                ;****** Rekursionstiefe bei verschachtelten Menüs feststellen
010A 3A6D00             lda     006Dh           ;Verschachtelungstiefe
010D D631               sui     '1'             ;Ascii-> direkte Codierung
010F FE09               cpi     09h             ;im Bereich 1 bis 9?
0111 DA1501             jc      nozero          ;nein
0114 AF                 xra     a               ;dann Stufe 1
0115 C631       nozero: adi     '1'             ;directe Codierung -> Ascii
0117 325605             sta     ebene           ;abspeichern
                ;
                ;****** Programmliste einlesen
011A 115C00             lxi     d,fcb           ;hier steht Name der Proglist.
011D 1A                 ldax    d               ;Laufwerkscode
011E A7                 ana     a               ;Vergleich mit 0
011F C22401             jnz     select          ;bestimmtes Laufwerk vorgegeb.
0122 3C                 inr     a               ;sonst Laufwerk A
0123 12                 stax    d               ;(statt Bezugslaufwerk)
0124 0E0F       select: mvi     c,open          ;Programmliste eröffnen
0126 CD0500             call    bdos
0129 3C                 inr     a               ;Vergleich mit FFH
012A CA4703             jz      nodat           ;Datei nicht gefunden
                ;(Datenrecords fortlaufend im Speicher ablegen)
012D 11D605             lxi     d,text          ;Hier wird Datei abgelegt
0130 D5         nxtrec: push    d
0131 0E1A               mvi     c,setdma        ;Datenpufferadresse festlegen
0133 CD0500             call    bdos
0136 115C00             lxi     d,fcb           ;Einen Record einlesen
0139 0E14               mvi     c,rdsequ
013B CD0500             call    bdos
013E D1                 pop     d               ;alte DMA-Adresse
013F 218000             lxi     h,0080h         ;neue berechnen
```

```
0142 19                 dad     d
0143 EB                 xchg
0144 A7                 ana     a               ;War letztes Lesen erfolgreich?
0145 CA3001             jz      nxtrec
0148 3E1A               mvi     a,1Ah           ;Dateiende-Kennzeichen anfügen
014A 12                 stax    d
                ;
                ;****** Anfangsadressen und Längen der Programmlistenzeilen
                ;(Textpointerbereich löschen)
014B 215805             lxi     h,point
014E 063C               mvi     b,3Ch           ;60 Zeichen lang
0150 AF                 xra     a               ;mit Null füllen
0151 77         clear:  mov     m,a
0152 23                 inx     h
0153 05                 dcr     b
0154 C25101             jnz     clear
                ;(mit neuen Werten füllen)
0157 215805             lxi     h,point         ;Ablageadresse der Textpointer
015A 11D605             lxi     d,text          ;Lage des Textes
015D 0613               mvi     b,13h           ;maximal 19 Zeilen
015F 73         nxtln:  mov     m,e             ;Adresse der Zeile
0160 23                 inx     h               ;in der Pointertabelle ablegen
0161 72                 mov     m,d
0162 23                 inx     h
0163 0EFF               mvi     c,0FFh          ;Zeilenlänge mit -1 vorbelegen
0165 0C         nxtcha: inr     c               ;Zeilenlänge erhöhen
0166 1A                 ldax    d               ;Zeichen aus der Zeile
0167 13                 inx     d
0168 FE1A               cpi     1Ah             ;vorzeitiges Textende?
016A CA8601             jz      txtend
016D FE0D               cpi     0Dh             ;Zeilenende?
016F C26501             jnz     nxtcha
0172 1A                 ldax    d
0173 FE0A               cpi     0Ah             ;Zeilenvorschub übergehen
0175 C27901             jnz     nolf
0178 13                 inx     d
0179 3E48       nolf:   mvi     a,48h           ;72 ist maximale Zeilenlänge
017B B9                 cmp     c
017C D28001             jnc     short
017F 4F                 mov     c,a             ;Überschuß nicht beachten
0180 71         short:  mov     m,c             ;Zeilenlänge in Pointertabelle
0181 23                 inx     h
0182 05                 dcr     b               ;noch eine Zeile?
0183 C25F01             jnz     nxtln
                ;
                ;****** Kopfzeile ausgeben
                ;(Verschachtelungstiefe bei rekursiven Aufrufen ausgeben)
0186 11C403     txtend: lxi     d,rectxt        ;Kommentar zur Schachteltiefe
0189 0E09               mvi     c,type          ;ausgeben
018B CD0500             call    bdos
018E 3A5605             lda     ebene           ;Wert der Schachteltiefe
0191 5F                 mov     e,a             ;Zeichen ausgeben
0192 CD7203             call    zeich
0195 1E0D               mvi     e,0Dh           ;Wagenrücklauf
0197 CD7203             call    zeich
                ;(Titel des Menüs ausgeben)
019A 215A05             lxi     h,point+2       ;Länge der ersten Zeile
019D 3E48               mvi     a,48h           ;Maximallänge
```

```
019F 96              sub     m               ;Differenz
01A0 1F              rar                     ;Halbieren
01A1 47              mov     b,a
01A2 CD7E03          call    space           ;Zeile zentrieren
01A5 215805          lxi     h,point         ;Zeiger auf Kopfzeile
01A8 CD9003          call    string          ;Zeile ausgeben
01AB CDB203          call    crlf
01AE 1E2D            mvi     e,'-'           ;Zeile unterstreichen
01B0 064F            mvi     b,4Fh
01B2 CD8003          call    fill            ;viele gleiche Zeichen ausgeben
01B5 CDB203          call    crlf            ;1 Leerzeile
01B8 CDB203          call    crlf
                     ;
                     ;****** Längste Menuezeile bestimmen
01BB 01FF00          lxi     b,00FFh
01BE 215D05          lxi     h,point+5       ;Länge der ersten Menüzeile
01C1 0C        nxtmen: inr   c               ;Anzahl der Menüzeilen
01C2 5E              mov     e,m             ;Länge der ersten Menüzeile
01C3 23              inx     h
01C4 23              inx     h
01C5 23              inx     h
01C6 7E              mov     a,m             ;Zugehörige Programmzeile
01C7 23              inx     h
01C8 23              inx     h
01C9 23              inx     h
01CA A7              ana     a               ;Länge 0?
01CB CADB01          jz      endmen          ;Ende des Menüs
01CE 7B              mov     a,e             ;Textzeile
01CF A7              ana     a
01D0 CADB01          jz      endmen          ;Ende des Menüs
01D3 B8              cmp     b               ;bisher längste Zeile
01D4 DAC101          jc      nxtmen          ;Neue ist kürzer
01D7 47              mov     b,a             ;neue längste Zeile
01D8 C3C101          jmp     nxtmen
01DB 79        endmen: mov   a,c             ;Anzahl der Menüzeilen
01DC 325705          sta     maxi            ;Speichern
01DF 3E48            mvi     a,48h           ;erlaubte maximale Länge
01E1 90              sub     b               ;Differenz zur tatsächlichen
01E2 1F              rar                     ;Halbieren
01E3 47              mov     b,a             ;soweit wird eingerückt
                     ;
                     ;****** Zeilen des Menüs ausgeben
01E4 1601            mvi     d,1             ;aktuelle Zeile
01E6 215B05          lxi     h,point+3       ;Pointer auf erste Textzeile
01E9 C5        mentxt: push  b
01EA CD7E03          call    space           ;Zeile zentrieren
01ED C1              pop     b
01EE 1E28            mvi     e,'('
01F0 CD7203          call    zeich
01F3 3E30            mvi     a,'0'           ;Ascii-Umwandlung
01F5 82              add     d               ;Zeilennummer
01F6 5F              mov     e,a
01F7 CD7203          call    zeich           ;Ausgeben
01FA 1E29            mvi     e,')'
01FC CD7203          call    zeich
01FF C5              push    b
0200 0604            mvi     b,4             ;Abstand zum Text
0202 CD7E03          call    space
```

```
0205 C1              pop     b
0206 CD9003          call    string          ;Text der Menüzeile ausgeben
0209 23              inx     h               ;zugehörige Programmzeile
020A 23              inx     h               ;überspringen
020B 23              inx     h
020C CDB203          call    crlf            ;Leerzeile
020F CDB203          call    crlf
0212 7A              mov     a,d             ;aktuelle Zeile
0213 14              inr     d               ;nächste
0214 B9              cmp     c               ;letzte Menüzeile?
0215 C2E901          jnz     mentxt          ;nächste Menüzeile
                     ;
                     ;****** Abschlußzeilen des Bildschirms ausgeben
0218 CD7E03          call    space           ;Einrücken
021B D5              push    d
021C 11DC03          lxi     d,endmsg        ;Endemeldung
021F 0E09            mvi     c,type
0221 CD0500          call    bdos
0224 D1              pop     d
0225 CDB203    leer: call    crlf            ;2 Leerzeilen
0228 CDB203          call    crlf
022B 7A              mov     a,d             ;aktuelle Doppelzeile
022C 14              inr     d
022D FE0A            cpi     0Ah             ;Unteres Bildschirmende erreich
022F C22502          jnz     leer
0232 11F003          lxi     d,waimsg        ;Eingabeanforderung
0235 0E09            mvi     c,type
0237 CD0500          call    bdos
                     ;
                     ;****** Warten, bis gültige Taste gedrückt wird
023A 1EFF      wait: mvi     e,0ffh          ;Direkte Konsoleneingabe
023C 0E06            mvi     c,conio
023E CD0500          call    bdos
0241 FE03            cpi     03h             ;Totalabbruch aller Menüstufen
0243 CA5D03          jz      kill
0246 5F              mov     e,a             ;Ausgabe vorbereiten
0247 0E06            mvi     c,conio
0249 FE45            cpi     'E'             ;dicse Rekursionsstufe zu Ende
024B CA5203          jz      ende
024E FE65            cpi     'e'
0250 CA5203          jz      ende
0253 D631            sui     31h             ;Ascii-> direkte Codierung
0255 215705          lxi     h,maxi          ;höchste erlaubte Eingabe
0258 BE              cmp     m
0259 D23A02          jnc     wait            ;keine gültige Taste
025C F5              push    psw
025D CD0500          call    bdos            ;gedrückte Taste anzeigen
0260 F1              pop     psw
                     ;****** gewählte Programmzeile zur Ausführung bringen
                     ;(Pointer der auszuführenden Zeile suchen)
0261 3C              inr     a               ;erste ist nie Programmzeile
0262 87              add     a               ;mal 2
0263 6F              mov     l,a
0264 87              add     a               ;mal 4
0265 85              add     l               ;mal 6
0266 5F              mov     e,a             ;relative Adresse des Pointers
0267 1600            mvi     d,0
0269 215805          lxi     h,point         ;absolute Pointeradresse
```

```
026C 19              dad     d
026D E5              push    h               ;Pointer der Programmzeile
                ;(Datei '$$$.SUB' eröffnen oder erzeugen)
026E 11B104          lxi     d,fcbsub        ;Versuch, zu eröffnen
0271 0E0F            mvi     c,open
0273 CD0500          call    bdos
0276 3C              inr     a               ;Ist $$$.SUB schon vorhanden?
0277 CA8502          jz      nosub
027A 11B104          lxi     d,fcbsub        ;Wenn ja, dann Nummer des erste
027D 0E23            mvi     c,size          ;freien Records bestimmen
027F CD0500          call    bdos
0282 C39102          jmp     subda
0285 11B104    nosub: lxi    d,fcbsub        ;'$$$.SUB' erzeugen
0288 0E16            mvi     c,make
028A CD0500          call    bdos
028D 3C              inr     a
028E CA4103          jz      wrerr           ;Disk Write Error
0291 11D504    subda: lxi    d,dma           ;Neue DMA-Adresse
0294 0E1A            mvi     c,setdma
0296 CD0500          call    bdos
                ;(nächsten Menüaufruf nach Programmende vorbereiten)
0299 11DE04          lxi     d,dma+9         ;freie Stelle in der DMA
029C 218200          lxi     h,0082h         ;Name der Programmliste
029F 7E        ldir: mov     a,m             ;Name in DMA übertragen
02A0 FE21            cpi     21h
02A2 DAAB02          jc      noldir
02A5 12              stax    d
02A6 23              inx     h
02A7 13              inx     d
02A8 C39F02          jmp     ldir
02AB EB        noldir: xchg                  ;Menue-Aufruf ergänzen
02AC 3620            mvi     m,' '
02AE 23              inx     h
02AF 3A5605          lda     ebene           ;Rekursionstiefe
02B2 77              mov     m,a
02B3 23              inx     h
02B4 3600            mvi     m,00h           ;Zeilenende
02B6 7D              mov     a,l
02B7 D6D6            sui     (dma+1) and Offh;Zeilenlänge berechnen
02B9 32D504          sta     dma             ;In DMA ablegen
02BC 11B104          lxi     d,fcbsub        ;DMA in '$$$.SUB' schreiben
02BF 0E22            mvi     c,wrrand        ;wahlfreier Zugriff
02C1 CD0500          call    bdos
02C4 A7              ana     a
02C5 C24103          jnz     wrerr
02C8 2AD204          lhld    fcbsub+21h      ;Recordnummer erhöhen
02CB 23              inx     h
02CC 22D204          shld    fcbsub+21h
                ;(Aufruf des gewählten Programms in '$$$.SUB' schreiben)
02CF E1              pop     h               ;Pointer auf Programmzeile
02D0 5E              mov     e,m             ;Adresse der Programmzeile
02D1 23              inx     h
02D2 56              mov     d,m
02D3 23              inx     h
02D4 46              mov     b,m             ;Länge der Programmzeile
02D5 21D504          lxi     h,dma           ;Ziel der Programmzeile
02D8 70              mov     m,b
02D9 23              inx     h
```

```
02DA 1A       ldir2:  ldax    d               ;Blocktransfer
02DB 77               mov     m,a
02DC 23               inx     h
02DD 13               inx     d
02DE 05               dcr     b
02DF C2DA02           jnz     ldir2
02E2 3600             mvi     m,00h           ;Endekennzeichen
                ;(Test, ob Programmaufruf ein weiteres Menü ist
02E4 21AB04           lxi     h,mennam        ;Text 'MENUE '
02E7 11D704           lxi     d,dma+2         ;Doppelpunkt bei Laufwerksangabe
02EA 1A               ldax    d
02EB 13               inx     d               ;Zeichen nach Laufwerksangabe
02EC FE3A             cpi     ':'
02EE CAF302           jz      drivea          ;keine Laufwerksangabe
02F1 1B               dcx     d               ;Name beginnt an erster Stelle
02F2 1B               dcx     d
02F3 0606     drivea: mvi     b,06h           ;6 Buchstaben
02F5 1A       test:   ldax    d               ;Zeichen im Aufruf
02F6 FE60             cpi     60h
02F8 DAFD02           jc      gross
02FB D620             sui     20h             ;in Großbuchstaben umwandeln
02FD BE       gross:  cmp     m               ;Vergleich mit 'MENUE '
02FE 23               inx     h
02FF 13               inx     d
0300 C22103           jnz     norec           ;kein Rekursiver Aufruf
0303 05               dcr     b
0304 C2F502           jnz     test
                ;(Rekursiver Aufruf eines Untermenüs: Tiefe markieren)
0307 EB               xchg                    ;hl Zeigt in DMA-Bereich
0308 7E       jump:   mov     a,m             ;nächses Wort überspringen
0309 23               inx     h               ;(Name der Programmliste)
030A FE21             cpi     21h
030C D20803           jnc     jump
030F 2B               dcx     h
0310 3620             mvi     m,20h           ;Leerzeichen anhängen
0312 23               inx     h
0313 3A5605           lda     ebene           ;Rekursionsstufe
0316 3C               inr     a               ;nächstes Menü tiefer verschachteln
0317 77               mov     m,a
0318 23               inx     h
0319 3600             mvi     m,00h           ;Endekennzeichen
031B 7D               mov     a,l
031C D6D6             sui     (dma+1) and 0FFh;Länge bestimmen
031E 32D504           sta     dma             ;In Programmzeile eintragen
                ;(neu Programmzeile in '$$$.SUB' abspeichern)
0321 11B104   norec:  lxi     d,fcbsub        ;Record wahlfrei schreiben
0324 0E22             mvi     c,wrrand
0326 CD0500           call    bdos
0329 A7               ana     a
032A C24103           jnz     wrerr           ;Disk Write Error
032D 11B104           lxi     d,fcbsub
0330 0E10             mvi     c,close         ;'$$$.SUB' schließen
0332 CD0500           call    bdos
0335 3C               inr     a
0336 CA4103           jz      wrerr           ;Fehler
0339 1E0C             mvi     e,cls           ;Bildschirm löschen
033B CD7203           call    zeich
033E C30000           jmp     boot            ;Warmstart
```

```
                        ;
                        ;****** Schreibfehler aufgetreten
0341 112904     wrerr:  lxi     d,errtxt        ;Fehlermeldung ausgeben
0344 C34A03             jmp     error
                        ;
                        ;****** Fehler: Programmliste nicht gefunden
0347 115D04     nodat:  lxi     d,prgtxt
034A 0E09       error:  mvi     c,type
034C CD0500             call    bdos
034F C30000             jmp     boot
                        ;
                        ;****** Programmende
0352 CD0500     ende:   call    bdos
0355 1E0C               mvi     e,cls           ;Bildschirm löschen
0357 CD7203             call    zeich
035A C30000             jmp     boot
                        ;
                        ;****** Abbruch aller Programme: '$$$.SUB' und 'XSUB' löschen
035D 11B104     kill:   lxi     d,fcbsub        ;Dateibeschreiber '$$$.SUB'
0360 0E13               mvi     c,delete        ;Löschen
0362 CD0500             call    bdos
0365 118004             lxi     d,break         ;Abbruchmeldung
0368 0E09               mvi     c,type
036A CD0500             call    bdos
036D 0E00               mvi     c,00h           ;Warmstart höchster Priorität
036F CD0500             call    bdos            ;(löscht aktiven XSUB-Modus)
                        ;
                        ;****** einzelnes Zeichen an Konsole ausgeben
0372 C5         zeich:  push    b
0373 D5                 push    d
0374 E5                 push    h
0375 0E06               mvi     c,conio         ;direkte Konsolenausgabe
0377 CD0500             call    bdos
037A E1                 pop     h
037B D1                 pop     d
037C C1                 pop     b
037D C9                 ret
                        ;
                        ;****** Text einrücken
037E 1E20       space:  mvi     e,20h
                        ;
                        ;****** viele gleiche Zeichen an Bildschirm ausgeben
0380 0E06       fill:   mvi     c,conio
0382 E5                 push    h
0383 C5         sploop: push    b
0384 D5                 push    d
0385 CD0500             call    bdos
0388 D1                 pop     d
0389 C1                 pop     b
038A 05                 dcr     b
038B C28303             jnz     sploop
038E E1                 pop     h
038F C9                 ret
                        ;
                        ;****** Zeile aus Programmliste ausgeben
0390 C5         string: push    b
0391 D5                 push    d
0392 5E                 mov     e,m             ;hl zeigt auf Variablenpointer
```

```
0393 23              inx     h
0394 56              mov     d,m             ;de enthält Stringadresse
0395 23              inx     h
0396 46              mov     b,m             ;b enthält Stringlänge
0397 23              inx     h
0398 E5              push    h               ;wird noch benötigt
0399 78              mov     a,b
039A A7              ana     a
039B CAAE03          jz      stend
039E EB              xchg                    ;hl enthält Stringadresse
039F 0E06            mvi     c,conio
03A1 5E      stloop: mov     e,m             ;Zeichen aus String
03A2 23              inx     h
03A3 C5              push    b
03A4 E5              push    h
03A5 CD0500          call    bdos
03A8 E1              pop     h
03A9 C1              pop     b
03AA 05              dcr     b
03AB C2A103          jnz     stloop
03AE E1      stend:  pop     h
03AF D1              pop     d
03B0 C1              pop     b
03B1 C9              ret
             ;
             ;****** Wagenrücklauf und Zeilenvorschub
03B2 C5      crlf:   push    b
03B3 D5              push    d
03B4 E5              push    h
03B5 11C103          lxi     d,neuln
03B8 0E09            mvi     c,type
03BA CD0500          call    bdos
03BD E1              pop     h
03BE D1              pop     d
03BF C1              pop     b
03C0 C9              ret
             ;
             ;****** Bildschirm-Ausgabestrings
03C1 0D0A24  neuln:  db      13,10,'$'
03C4 0C09090909rectxt: db    cls,9,9,9,9,9,9,9,9,'       Ebene: $'
03DC 2845292020endmsg: db    '(E)    Menue-Ende',13,10,'$'
03F0 0909090909waimsg: db    9,9,9,9,9,9,'    (c) 1985 by Isar-Amper-'
0411 536F66740D      db      'Soft',13,'Bitte wählen Sie: $'
0429 0D0A2A2A2Aerrtxt: db    13,10,'***** Diskette oder Inhaltsverzeich'
044E 6E69732076      db      'nis voll *****$'
045D 2A2A2A2A2Aprgtxt: db    '***** Datei nicht gefunden *****',13,10,'$'
0480 0D0A2A2A2Abreak:  db    13,10,'***** Menue - Programm abgebrochen '
04A5 2A2A2A2A2A      db      '*****$'
04AB 4D454E5545mennam: db    'MENUE '
             ;
             ;****** Variablenbereich
04B1 01      fcbsub: db      1               ;Laufwerk A
04B2 2424242020      db      '$$$     SUB'   ;11 Zeichen Programmname
04BD 00000000        db      0,0,0,0         ;Extentbytes usw.
04C1 0000000000      dw      0,0,0,0         ;Blockreservierungen
04C9 0000000000      dw      0,0,0,0         ;16 beliebige Bytes
04D1 00000000        db      0,0,0,0         ;Recordnummern sequ+wahlfrei
04D5 00413A4D45dma:  db      0,'A:MENUE '    ;Datenpuffer
```

```
                ;
                ;****** Dieser Bereich braucht nicht abgespeichert zu werden
04DE                    ds      120
0556            ebene:  ds      1               ;Rekursionstiefe
0557            maxi:   ds      1               ;Anzahl der Menuezeilen
0558            point:  ds      3Ch             ;Variablenpointer (Texte)
0594                    ds      64              ;Raum für Stack
05D4            oldstk: ds      2               ;alter Stackpointer
05D6            text:   end
```

Der Basic-Lader zum Menuegenerator

```
100 'Basic-Lader fuer die Datei 'MENUE.COM'
110 '
120 OPENOUT"MENUE.COM"
130 '
140 z%=10000
150 READ d$:WHILE d$<>"Ende"
160 d%=VAL("&"+d$):PRINT#9,CHR$(d%);:p%=d%
170 FOR i%=1 TO 15
180 READ d$:d%=VAL("&"+d$)
190 PRINT#9,CHR$(d%);:p%=p%+d%
200 NEXT
210 READ q%
220 IF p%<>q% THEN PRINT"Tippfehler in Zeile "z%:CLOSEOUT:END
230 z%=z%+10
240 READ d$:WEND
250 CLOSEOUT
260 PRINT"Alles O.K."
270 END
280 '
10000 DATA 21,00,00,39,22,D4,05,31,D4,05,3A,6D,00,D6,31,FE,1291
10010 DATA 09,DA,15,01,AF,C6,31,32,56,05,11,5C,00,1A,A7,C2,1308
10020 DATA 24,01,3C,12,0E,0F,CD,05,00,3C,CA,47,03,11,D6,05, 926
10030 DATA D5,0E,1A,CD,05,00,11,5C,00,0E,14,CD,05,00,D1,21,1058
10040 DATA 80,00,19,EB,A7,CA,30,01,3E,1A,12,21,58,05,06,3C,1104
10050 DATA AF,77,23,05,C2,51,01,21,58,05,11,D6,05,06,13,73,1112
10060 DATA 23,72,23,0E,FF,0C,1A,13,FE,1A,CA,86,01,FE,0D,C2,1588
10070 DATA 65,01,1A,FE,0A,C2,79,01,13,3E,48,B9,D2,80,01,4F,1464
10080 DATA 71,23,05,C2,5F,01,11,C4,03,0E,09,CD,05,00,3A,56,1036
10090 DATA 05,5F,CD,72,03,1E,0D,CD,72,03,21,5A,05,3E,48,96,1199
10100 DATA 1F,47,CD,7E,03,21,58,05,CD,90,03,CD,B2,03,1E,2D,1375
10110 DATA 06,4F,CD,80,03,CD,B2,03,CD,B2,03,01,FF,00,21,5D,1575
10120 DATA 05,0C,5E,23,23,23,7E,23,23,23,A7,CA,DB,01,7B,A7,1326
10130 DATA CA,DB,01,B8,DA,C1,01,47,C3,C1,01,79,32,57,05,3E,1803
10140 DATA 48,90,1F,47,16,01,21,5B,05,C5,CD,7E,03,C1,1E,28,1264
10150 DATA CD,72,03,3E,30,82,5F,CD,72,03,1E,29,CD,72,03,C5,1569
10160 DATA 06,04,CD,7E,03,C1,CD,90,03,23,23,23,CD,B2,03,CD,1585
10170 DATA B2,03,7A,14,B9,C2,E9,01,CD,7E,03,D5,11,DC,03,0E,1737
10180 DATA 09,CD,05,00,D1,CD,B2,03,CD,B2,03,7A,14,FE,0A,C2,1800
10190 DATA 25,02,11,F0,03,0E,09,CD,05,00,1E,FF,0E,06,CD,05,1047
10200 DATA 00,FE,03,CA,5D,03,5F,0E,06,FE,45,CA,52,03,FE,65,1635
10210 DATA CA,52,03,D6,31,21,57,05,BE,D2,3A,02,F5,CD,05,00,1590
10220 DATA F1,3C,87,6F,87,85,5F,16,00,21,58,05,19,E5,11,B1,1506
10230 DATA 04,0E,0F,CD,05,00,3C,CA,85,02,11,B1,04,0E,23,CD,1092
10240 DATA 05,00,C3,91,02,11,B1,04,0E,16,CD,05,00,3C,CA,41,1118
10250 DATA 03,11,D5,04,0E,1A,CD,05,00,11,DE,04,21,82,00,7E,1019
10260 DATA FE,21,DA,AB,02,12,23,13,C3,9F,02,EB,36,20,23,3A,1520
10270 DATA 56,05,77,23,36,00,7D,D6,D6,32,D5,04,11,B1,04,0E,1331
10280 DATA 22,CD,05,00,A7,C2,41,03,2A,D2,04,23,22,D2,04,E1,1437
10290 DATA 5E,23,56,23,46,21,D5,04,70,23,1A,77,23,13,05,C2,1115
10300 DATA DA,02,36,00,21,AB,04,11,D7,04,1A,13,FE,3A,CA,F3,1520
10310 DATA 02,1B,1B,06,06,1A,FE,60,DA,FD,02,D6,20,BE,23,13,1407
10320 DATA C2,21,03,05,C2,F5,02,EB,7E,23,FE,21,D2,08,03,28,1623
10330 DATA 36,20,23,3A,56,05,3C,77,23,36,00,7D,D6,D6,32,D5,1354
10340 DATA 04,11,B1,04,0E,22,CD,05,00,A7,C2,41,03,11,B1,04,1087
10350 DATA 0E,10,CD,05,00,3C,CA,41,03,1E,0C,CD,72,03,C3,00,1129
10360 DATA 00,11,29,04,C3,4A,03,11,5D,04,0E,09,CD,05,00,C3, 876
```

```
10370 DATA 00,00,CD,05,00,1E,0C,CD,72,03,C3,00,00,11,B1,04, 967
10380 DATA 0E,13,CD,05,00,11,80,04,0E,09,CD,05,00,0E,00,CD, 844
10390 DATA 05,00,C5,D5,E5,0E,06,CD,05,00,E1,D1,C1,C9,1E,20,1764
10400 DATA 0E,06,E5,C5,D5,CD,05,00,D1,C1,05,C2,83,03,E1,C9,2030
10410 DATA C5,D5,5E,23,56,23,46,23,E5,78,A7,CA,AE,03,EB,0E,1909
10420 DATA 06,5E,23,C5,E5,CD,05,00,E1,C1,05,C2,A1,03,E1,D1,1986
10430 DATA C1,C9,C5,D5,E5,11,C1,03,0E,09,CD,05,00,E1,D1,C1,2106
10440 DATA C9,0D,0A,24,0C,09,09,09,09,09,09,09,20,20,20, 440
10450 DATA 20,20,20,20,45,62,65,6E,65,3A,20,24,28,45,29,20, 915
10460 DATA 20,20,20,4D,65,6E,75,65,2D,45,6E,64,65,0D,0A,24,1086
10470 DATA 09,09,09,09,09,09,20,20,20,20,28,63,29,20,31,39, 500
10480 DATA 38,35,20,62,79,20,49,73,61,72,2D,41,6D,70,65,72,1337
10490 DATA 2D,53,6F,66,74,0D,42,69,74,74,65,20,77,7B,68,6C,1460
10500 DATA 65,6E,20,53,69,65,3A,20,24,0D,0A,2A,2A,2A,2A,2A, 891
10510 DATA 20,44,69,73,6B,65,74,74,65,20,6F,64,65,72,20,49,1424
10520 DATA 6E,68,61,6C,74,73,76,65,72,7A,65,69,63,68,6E,69,1729
10530 DATA 73,20,76,6F,6C,6C,20,2A,2A,2A,2A,2A,24,2A,2A,2A, 996
10540 DATA 2A,2A,20,44,61,74,65,69,20,6E,69,63,68,74,20,67,1304
10550 DATA 65,66,75,6E,64,65,6E,20,2A,2A,2A,2A,2A,0D,0A,24,1042
10560 DATA 0D,0A,2A,2A,2A,2A,2A,20,4D,65,6E,75,65,20,2D,20, 880
10570 DATA 50,72,6F,67,72,61,6D,20,61,62,67,65,62,72,6F,1591
10580 DATA 63,68,65,6E,20,2A,2A,2A,2A,2A,24,4D,45,4E,55,45,1070
10590 DATA 20,01,24,24,24,20,20,20,20,20,53,55,42,00,00,00, 535
10600 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,   0
10610 DATA 00,00,00,00,00,00,41,3A,4D,45,4E,55,45,20,00,00, 533
10620 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,   0
10630 DATA Ende
```

12.2.2 Druckersteuerung unter CP/M

Dieses Programm ermöglicht es Ihnen, die Schriftarten Ihres Druckers sehr komfortabel umzuschalten: Die Umschaltung erfolgt durch die Angabe von 'LETTER abcdef', wobei a...f für verschiedene Schriftartparameter steht. die Bedeutung der Parameter sehen Sie in Bild 12-4.

```
 Schriftartenwahl auf dem NLQ401
Aufruf: 1) 'LETTER ?'      : Helptexte
        2) 'LETTER abcd ef': (Defaultwerte fuer fehlende Parameter)
a: Hervorhebung 1       0 -> nicht unterstreichen
                        1 -> unterstreichen
b: Hervorhebung 2       0 -> keine Breitschrift
                        1 -> Breitschrift
c: Schriftart           0 -> Schnellschrift
                        1 -> Schmalschrift
                        2 -> Fettdruck
                        3 -> Near Letter Quality
d: Schriftvarianten:    0 -> normal
                        1 -> Doppeldruck
                        2 -> Hochgestellt
                        3 -> Tiefgestellt
e: Zeilenabstand        1 -> 0.5 Zeilen (1/12 ")
                        2 -> 1   Zeile  (1/6  ")
                        3 -> 1.5 Zeilen (1/4  ")
                        ...
                        6 -> 3   Zeilen (1/2  ")
f: Zeichensatz          0 -> deutsche Zeichen
                        1 -> amerikanische Zeichen
In der Schriftart NLQ werden die Schriftvarianten nicht ausgewertet.
```

Bild 12-4: *Die Aufrufparameter von LETTER.COM*

Nach der Eingabe von 'LETTER 0030 31' werden die Zeichen auf dem Drucker in der Schriftart NLQ, nicht unterstrichen und nicht vergrößert ausgegeben. Zusätzlich wird der Zeilenabstand auf 1/4Zoll eingestellt und die deutschen Sonderzeichen abgeschaltet.

Im allgemeinen sind alle durch die Parameter eingebbaren Kombinationen auch wirklich auf dem Drucker darstellbar. Die einzige Ausnahme ist die Schriftart NLQ: Wenn diese gewählt wurde, werden vom Drucker die Angaben zu den Schriftvarianten nicht ausgewertet.

Beim Aufruf müssen nicht alle Parameter angegeben werden: Bei einem fehlenden Parameter wird der Wert 0 verwendet. Eine Ausnahme ist die Zeilenhöhe. Bei dieser ist der Standardwert die Zahl 2, also 1/6 Zoll. Ein Aufruf ganz ohne Parameter, mit 'LETTER' alleine entspricht also dem Aufruf 'LETTER 0000 2'.

Wenn Sie einmal die Bedeutung eines Parameters vergessen haben, können Sie sich durch 'LETTER ?' einen Hilfszettel ausgeben lassen, der dieselben

Informationen enthält wie Bild 12-4. Dieser Hilfszettel wird auch automatisch ausgegeben, wenn Sie einen falschen Parameter angegeben haben.

Anpassung des Programms an verschiedene Drucker

Die vorliegende Version ist für den Drucker Schneider NLQ401 vollständig angepaßt. Da der NLQ401 keine besonders exotische Druckeransteuerung besitzt, werden die meisten Steuerzeichen aber auch von anderen Standard-Druckern verstanden. Bevor Sie das Programm verändern, sollten Sie deshalb erst ausprobieren, ob überhaupt eine Änderung nötig ist.

Wenn wirklich einige Steuerzeichen geändert werden müssen, braucht nur in die Steuerzeichentabelle am Ende des Programms eingegriffen zu werden. Die Änderungen können Sie entweder im Assemblerlisting oder mit Hilfe von 'DDT LETTER.COM' direkt im endgültigen Programm vornehmen. Dort beginnt sie an der Adresse 04BDh. Diese Möglichkeit bietet sich an, wenn Sie Assembler nicht verstehen und das Programm deshalb mit Hilfe des Basic-Laders eingegeben haben.

Die Tabelle besteht aus 21 Sätzen, von denen keiner ausgelassen und deren Reihenfolge nicht verändert werden darf. Auch wenn ein neu eingesetzter Tabellensatz kürzer ist als der alte, müssen die Sätze ohne Zwischenraum aufeinanderfolgen. Bei einer Verkürzung muß der nachfolgende Tabellenbereich verschoben werden. Die Reihenfolge der Sätze zeigt Bild 12-5.

Tabellensatz	Bedeutung
1	Unterstreichen abschalten
2	Unterstreichen einschalten
3	Breitschrift ausschalten
4	Breitschrift einschalten
5	Schnellschrift einschalten
6	komprinierte Zeichen
7	Fettdruck
8	Schönschrift (NLQ)
9	Schriftvarianten abschalten
10	Doppeldruch
11	Zeichen hochstellen
12	Zeichen tiefstellen
13	Standardzeilenhöhe wählen
14	Zeilenhöhe 1/12 Zoll (1/2)
15	Zeilenhöhe 1/6 Zoll (1)
16	Zeilenhöhe 1/4 Zoll (1 1/2)
17	Zeilenhöhe 1/3 Zoll (2)
18	Zeilenhöhe 5/12 Zoll (2 1/2)
19	Zeilenhöhe 1/2 Zoll (3)
20	deutschen Zeichensatz wählen
21	amerikanischen Zeichensatz

Bild 12-5: *Aufbau der Steuerzeichentabelle in LETTER.COM*

Jeder Satz enthält erst ab dem 2. Byte die eigentlichen Steuerzeichen. Im ersten Byte muß die Anzahl der Steuerzeichen dieses Tabellensatzes stehen.

Bei der Wahl einer bestimmten Schriftart muß sichergestellt werden, daß nicht gleichzeitig darstellbare Schriftarten mit höherer Priorität abgeschaltet werden. Beim Schneider-NLQ401 enthält deshalb etwa der Tabellensatz zum Einschalten der Schmalschrift auch die Steuerzeichen zum Abschalten der Schönschrift und zum Abschalten des Fettdrucks. Wenn Ihr Drucker eine Schriftart nicht bietet, müssen Sie eine unschädliche Steuerzeichenfolge einsetzen. Das ist zum Beispiel der Tabellensatz, der nur aus einem einzigen Byte mit dem Wert 0 besteht.

Das Assemblerlisting zu LETTER.COM

```
                ;Drucker-Schriftartumschaltung
                ;
                ;
                ;****** Konstanten
0005 =          bdos    equ     0005h
005D =          word1   equ     005dh
006D =          word2   equ     006dh
0009 =          write   equ     09h
0005 =          print   equ     05h
1000 =          stack   equ     1000h
0100            org     0100h
                ;
                ;****** Lokalen Stapel anlegen
0100 210000     start:  lxi     h,0000h
0103 39                 dad     sp              ;alter Stackpointer
0104 22FE0F             shld    stack-2         ;Zwischenspeichern
0107 31FE0F             lxi     sp,stack-2      ;neuer Stack
                ;
                ;****** Hauptprogramm
010A 3A5D00     repeat: lda     word1           ;1.Parameter
010D FE3F               cpi     '?'
010F CA9501             jz      help            ;Hilfstext
0112 010102             lxi     b,0201h         ;ab 1. Tabellensatz suchen
0115 CD4801             call    switch          ;(maximal 2 Einträge)
0118 3A5E00             lda     word1+1         ;2. Parameter
011B 010302             lxi     b,0203h         ;ab 3. Tabellensatz suchen
011E CD4801             call    switch          ;(maximal 2 Einträge)
0121 3A5F00             lda     word1+2         ;3. Parameter
0124 010504             lxi     b,0405h         ;ab 5. Tabellensatz suchen
0127 CD4801             call    switch          ;(maximal 4 Einträge)
012A 3A6000             lda     word1+3         ;4. Parameter
012D 010904             lxi     b,0409h         ;ab 9. Tabellensatz suchen
0130 CD4801             call    switch          ;(maximal 4 Einträge)
0133 3A6D00             lda     word2           ;5. Parameter
0136 010D07             lxi     b,070dh         ;ab 13. Tabellensatz suchen
0139 CD4801             call    switch          ;(maximal 7 Einträge)
013C 3A6E00             lda     word2+1         ;6. Parameter
013F 011402             lxi     b,0214h         ;ab 20. Tabellensatz suchen
0142 CD4801             call    switch          ;(maximal 2 Einträge)
0145 C39D01             jmp     finis           ;Programmende
                ;
                ;****** Zuständigen Tabellensatz suchen und ausgeben
0148 FE20       switch: cpi     ' '             ;Leerzeichen gilt als 0
014A C24F01             jnz     char
014D 3E30               mvi     a,30h
014F D630       char:   sui     '0'     ;Direkte Codierung berechnen
0151 DA7701             jc      error   ;wenn kleiner, dann Fehler
0154 B8                 cmp     b       ;Maximaler Wert
0155 D27701             jnc     error
0158 81                 add     c       ;Versatz
0159 21BC04             lxi     h,tabell;Tabellenanfang
015C 1600               mvi     d,00h
015E 5E         such:   mov     e,m     ;Satzlänge
015F 23                 inx     h
0160 19                 dad     d       ;Überspringen
0161 3D                 dcr     a
```

```
0162 C25E01           jnz     such    ;Solange, bis gefunden
0165 46               mov     b,m     ;Länge des gültigen Satzes
0166 23               inx     h
0167 0E05             mvi     c,print ;Steuercode Druckerausgabe
0169 5E       druck:  mov     e,m     ;Auszugebendes Zeichen
016A 23               inx     h
016B E5               push    h
016C C5               push    b
016D CD0500           call    bdos    ;Zeichen an Drucker
0170 C1               pop     b
0171 E1               pop     h
0172 05               dcr     b       ;noch ein Zeichen?
0173 C26901           jnz     druck
0176 C9               ret
                ;
                ;****** Fehlermeldung
0177 E1       error:  pop     h       ;Stapel korrigieren
0178 11A201           lxi     d,fehler;Meldetext 'Falscher Parameter'
017B 0E09             mvi     c,write ;Text ausgeben
017D CD0500           call    bdos
0180 215D00           lxi     h,word1 ;Eingabe löschen
0183 3E20             mvi     a,' '
0185 77               mov     m,a
0186 23               inx     h
0187 77               mov     m,a
0188 23               inx     h
0189 77               mov     m,a
018A 23               inx     h
018B 77               mov     m,a
018C 216D00           lxi     h,word2
018F 77               mov     m,a
0190 23               inx     h
0191 77               mov     m,a
0192 C30A01           jmp     repeat  ;Programm mit Standardeinstellung
                ;
                ;****** Hilfstexte ausgeben
0195 11BA01   help:   lxi     d,hilfen
0198 0E09     monit:  mvi     c,write
019A CD0500           call    bdos
                ;
                ;****** Programmende
019D 2AFE0F   finis:  lhld    stack-2 ;alter Stackpointer
01A0 F9               sphl            ;zurück
01A1 C9               ret             ;Sprung ins System
                ;
                ;****** Meldetexte
01A2 0D0A204661fehler: db     13,10,' Falscher Parameter ! '
01BA 0D0A182053hilfen: db     13,10,24,' Schriftartenwahl auf dem '
01D7 274E4C5134       db      '''NLQ401'' ',24,13,10,'Aufruf:'
01EA 2031292027       db      ' 1) ''LETTER ?''      : Helptexte',13,10
020B 2020202020       db      '       2) ''LETTER abcd ef'':  '
0229 2844656661       db      '(Defaultwerte fuer fehlende Parameter)',13,10
0251 613A204865       db      'a: Hervorhebung 1'
0262 0930202D3E       db      9,'0 -> nicht unterstreichen',13,10
027E 0909093120       db      9,9,9,'1 -> unterstreichen',13,10
0296 623A204865       db      'b: Hervorhebung 2'
02A7 0930202D3E       db      9,'0 -> keine Breitschrift',13,10
02C1 0909093120       db      9,9,9,'1 -> Breitschrift',13,10
```

```
02D7 633A205363         db      'c: Schriftart'
02E4 090930202D         db      9,9,'0 -> Schnellschrift',13,10
02FB 0909093120         db      9,9,9,'1 -> Schmalschrift',13,10
0312 0909093220         db      9,9,9,'2 -> Fettdruck',13,10
0325 0909093320         db      9,9,9,'3 -> Near Letter Quality',13,10
0342 643A205363         db      'd: Schriftvarianten:'
0356 0930202D3E         db      9,'0 -> normal',13,10
0364 0909093120         db      9,9,9,'1 -> Doppeldruck',13,10
0379 0909093220         db      9,9,9,'2 -> Hochgestellt',13,10
038F 0909093320         db      9,9,9,'3 -> Tiefgestellt',13,10
03A5 653A205A65         db      'e: Zeilenabstand'
03B5 0931202D3E         db      9,'1 -> 0.5 Zeilen (1/12 ")',13,10
03D0 0909093220         db      9,9,9,'2 -> 1   Zeile  (1/6  ")',13,10
03ED 0909093320         db      9,9,9,'3 -> 1.5 Zeilen (1/4  ")',13,10
040A 0909092E2E         db      9,9,9,'...',13,10
0412 0909093620         db      9,9,9,'6 -> 3   Zeilen (1/2  ")',13,10
042F 663A205A65         db      'f: Zeichensatz'
043D 090930202D         db      9,9,'0 -> deutsche Zeichen',13,10
0456 0909093120         db      9,9,9,'1 -> amerikanische Zeichen',13,10
0475 496E206465         db      'In der Schriftart ''NLQ'' werden die '
0498 5363687269         db      'Schriftvarianten nicht ausgewertet.$'

                        ;
                        ;****** Steuerzeichenfolgen (jeweils Länge + Parameter)
04BC 00         tabell: db      00                      ;Dummy, '0-ter' Satz
04BD 031B2D00           db      3,1bh,2dh,00h           ;nicht unterstreichen
04C1 031B2D01           db      3,1bh,2dh,01h           ;unterstreichen
04C5 031B5700           db      3,1bh,57h,00h           ;keine Breitschrift
04C9 031B5701           db      3,1bh,57h,01h           ;Breitschrift
04CD 06121B461B         db      6,12h,1bh,46h,1bh,78h,00h;Schnellschrift
04D4 060F1B461B         db      6,0fh,1bh,46h,1bh,78h,00h;Schmalschrift
04DB 051B451B78         db      5,1bh,45h,1bh,78h,00h   ;Fettdruck
04E1 031B7801           db      3,1bh,78h,01h           ;NLQ
04E5 041B481B54         db      4,1bh,48h,1bh,54h       ;keine Variante
04EA 041B471B54         db      4,1bh,47h,1bh,54h       ;Doppeldruck
04EF 031B5300           db      3,1bh,53h,00h           ;hochgestellt
04F3 031B5301           db      3,1bh,53h,01h           ;Tiefgestellt
04F7 051B410C1B         db      5,1bh,41h,0ch,1bh,32h   ;Zeilenabstand 1/6 "
04FD 051B41061B         db      5,1bh,41h,06h,1bh,32h   ;1/12 "
0503 051B410C1B         db      5,1bh,41h,0ch,1bh,32h   ;1/6 "
0509 051B41121B         db      5,1bh,41h,12h,1bh,32h   ;1/4 "
050F 051B41181B         db      5,1bh,41h,18h,1bh,32h   ;1/3 "
0515 051B411E1B         db      5,1bh,41h,1eh,1bh,32h   ;5/12 "
051B 051B41241B         db      5,1bh,41h,24h,1bh,32h   ;1/2 "
0521 021B36             db      2,1Bh,36h               ;amerikanisch
0524 021B37             db      2,1Bh,37h               ;deutsch
0527                    end
```

Der Basic-Lader zu LETTER.COM

```
100 'Basic-Lader fuer die Datei 'LETTER.COM'
110 '
120 OPENOUT"LETTER.COM"
130 '
140 z%=10000
150 READ d$:WHILE d$<>"Ende"
160 d%=VAL("&"+d$):PRINT#9,CHR$(d%);:p%=d%
170 FOR i%=1 TO 15
180 READ d$:d%=VAL("&"+d$)
190 PRINT#9,CHR$(d%);:p%=p%+d%
200 NEXT
210 READ q%
220 IF p%<>q% THEN PRINT"Tippfehler in Zeile "z%:CLOSEOUT:END
230 z%=z%+10
240 READ d$:WEND
250 CLOSEOUT
260 PRINT"Alles O.K."
270 END
280 '
10000 DATA 21,00,00,39,22,FE,0F,31,FE,0F,3A,5D,00,FE,3F,CA,1381
10010 DATA 95,01,01,01,02,CD,48,01,3A,5E,00,01,03,02,CD,48, 867
10020 DATA 01,3A,5F,00,01,05,04,CD,48,01,3A,60,00,01,09,04, 610
10030 DATA CD,48,01,3A,6D,00,01,0D,07,CD,48,01,3A,6E,00,01, 913
10040 DATA 14,02,CD,48,01,C3,9D,01,FE,20,C2,4F,01,3E,30,D6,1537
10050 DATA 30,DA,77,01,B8,D2,77,01,81,21,BC,04,16,00,5E,23,1405
10060 DATA 19,3D,C2,5E,01,46,23,0E,05,5E,23,E5,C5,CD,05,00,1264
10070 DATA C1,E1,05,C2,69,01,C9,E1,11,A2,01,0E,09,CD,05,00,1562
10080 DATA 21,5D,00,3E,20,77,23,77,23,77,23,77,21,6D,00,77,1062
10090 DATA 23,77,C3,0A,01,11,BA,01,0E,09,CD,05,00,2A,FE,0F,1108
10100 DATA F9,C9,0D,0A,20,46,61,6C,73,63,68,65,72,20,50,61,1522
10110 DATA 72,61,6D,65,74,65,72,20,21,20,0D,0A,18,20,53,63,1110
10120 DATA 68,72,69,66,74,61,72,74,65,6E,77,61,68,6C,20,61,1636
10130 DATA 75,66,20,64,65,6D,20,27,4E,4C,51,34,30,31,27,20,1087
10140 DATA 18,0D,0A,41,75,66,72,75,66,3A,20,31,29,20,27,4C, 991
10150 DATA 45,54,54,45,52,20,3F,27,20,20,20,20,20,20,3A,20, 804
10160 DATA 48,65,6C,70,74,65,78,74,65,0D,0A,20,20,20,20,20,1130
10170 DATA 20,20,20,32,29,20,27,4C,45,54,54,45,52,20,61,62, 949
10180 DATA 63,64,20,65,66,27,3A,20,20,28,44,65,66,61,75,6C,1228
10190 DATA 74,77,65,72,74,65,20,66,75,65,72,20,66,65,68,6C,1580
10200 DATA 65,6E,64,65,20,50,61,72,61,6D,65,74,65,72,29,0D,1427
10210 DATA 0A,61,3A,20,48,65,72,76,6F,72,68,65,62,75,6E,67,1460
10220 DATA 20,31,09,30,20,2D,3E,20,6E,69,63,68,74,20,75,6E,1102
10230 DATA 74,65,72,73,74,72,65,69,63,68,65,6E,0D,0A,09,09,1337
10240 DATA 09,31,20,2D,3E,20,75,6E,74,65,72,73,74,72,65,69,1338
10250 DATA 63,68,65,6E,0D,0A,62,3A,20,48,65,72,76,6F,72,68,1359
10260 DATA 65,62,75,6E,67,20,32,09,30,20,2D,3E,20,6B,65,69,1152
10270 DATA 6E,65,20,42,72,65,69,74,73,63,68,72,69,66,74,0D,1513
10280 DATA 0A,09,09,09,31,20,2D,3E,20,42,72,65,69,74,73,63, 973
10290 DATA 68,72,69,66,74,0D,0A,63,3A,20,53,63,68,72,69,66,1360
10300 DATA 74,61,72,74,09,09,30,20,2D,3E,20,53,63,68,6E,65,1177
10310 DATA 6C,6C,73,63,68,72,69,66,74,0D,0A,09,09,09,31,20,1102
10320 DATA 2D,3E,20,53,63,68,6D,61,6C,73,63,68,72,69,66,74,1494
10330 DATA 0D,0A,09,09,09,32,20,2D,3E,20,46,65,74,74,64,72, 888
10340 DATA 75,63,6B,0D,0A,09,09,09,33,20,2D,3E,20,4E,65,61, 871
10350 DATA 72,20,4C,65,74,74,65,72,20,51,75,61,6C,69,74,79,1547
10360 DATA 0D,0A,64,3A,20,53,63,68,72,69,66,74,76,61,72,69,1370
```

```
10370 DATA 61,6E,74,65,6E,3A,09,30,20,2D,3E,20,6E,6F,72,6D,1264
10380 DATA 61,6C,0D,0A,09,09,09,31,20,2D,3E,20,44,6F,70,70, 878
10390 DATA 65,6C,64,72,75,63,68,0D,0A,09,09,09,32,20,2D,3E, 985
10400 DATA 20,48,6F,63,68,67,65,73,74,65,6C,6C,74,0D,0A,09,1318
10410 DATA 09,09,33,20,2D,3E,20,54,69,65,66,67,65,73,74,65,1168
10420 DATA 6C,6C,74,0D,0A,65,3A,20,5A,65,69,6C,65,6E,61,62,1356
10430 DATA 73,74,61,6E,64,09,31,20,2D,3E,20,30,2E,35,20,5A,1036
10440 DATA 65,69,6C,65,6E,20,28,31,2F,31,32,20,22,29,0D,0A, 922
10450 DATA 09,09,09,32,20,2D,3E,20,31,20,20,20,5A,65,69,6C, 797
10460 DATA 65,20,20,28,31,2F,36,20,20,22,29,0D,0A,09,09,09, 544
10470 DATA 33,20,2D,3E,20,31,2E,35,20,5A,65,69,6C,65,6E,20,1049
10480 DATA 28,31,2F,34,20,20,22,29,0D,0A,09,09,09,2E,2E,2E, 515
10490 DATA 0D,0A,09,09,09,36,20,2D,3E,20,33,20,20,20,5A,65, 613
10500 DATA 69,6C,65,6E,20,28,31,2F,32,20,20,22,29,0D,0A,66, 906
10510 DATA 3A,20,5A,65,69,63,68,65,6E,73,61,74,7A,09,09,30,1316
10520 DATA 20,2D,3E,20,64,65,75,74,73,63,68,65,20,5A,65,69,1352
10530 DATA 63,68,65,6E,0D,0A,09,09,09,31,20,2D,3E,20,61,6D, 890
10540 DATA 65,72,69,6B,61,6E,69,73,63,68,65,20,5A,65,69,63,1585
10550 DATA 68,65,6E,0D,0A,49,6E,20,64,65,72,20,53,63,68,72,1300
10560 DATA 69,66,74,61,72,74,20,27,4E,4C,51,27,20,77,65,72,1361
10570 DATA 64,65,6E,20,64,69,65,20,53,63,68,72,69,66,74,76,1522
10580 DATA 61,72,69,61,6E,74,65,6E,20,6E,69,63,68,74,20,61,1545
10590 DATA 75,73,67,65,77,65,72,74,65,74,2E,24,00,03,1B,2D,1260
10600 DATA 00,03,1B,2D,01,03,1B,57,00,03,1B,57,01,06,12,1B, 362
10610 DATA 46,1B,78,00,06,0F,1B,46,1B,78,00,05,1B,45,1B,78, 730
10620 DATA 00,03,1B,78,01,04,1B,48,1B,54,04,1B,47,1B,54,03, 581
10630 DATA 1B,53,00,03,1B,53,01,05,1B,41,0C,1B,32,05,1B,41, 507
10640 DATA 06,1B,32,05,1B,41,0C,1B,32,05,1B,41,12,1B,32,05, 466
10650 DATA 1B,41,18,1B,32,05,1B,41,1E,1B,32,05,1B,41,24,1B, 557
10660 DATA 32,02,1B,36,02,1B,37,00,00,00,00,00,00,00,00,00, 217
10670 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,   0
10680 DATA Ende
```

12.3 CP/M-Programme speziell für die Schneider-Computer

Die beiden Programme in diesem Kapitel stützen sich auf die speziellen Firmware-Routinen der CPC-Computer. Deshalb sind sie nur auf dem CPC 464/664 unter CP/M 2.2 lauffähig. Dafür werden aber auch unter CP/M Dinge möglich, die bisher nur unter dem Basic-Betriebssystem möglich waren.

12.3.1 Deutscher Zeichensatz unter CP/M

Auch unter CP/M ist es möglich, die Firmwareroutinen zum Definieren eines eigenen Zeichensatzes zu verwenden. Die Standardmethode dazu ist, mit Hilfe von MOVCPM einen genügend großen freien Speicher zu erzeugen und die Zeichensatztabelle in diesem Bereich abzulegen.

Im deutschen Zeichensatz ist das Paragraphzeichen '§' das neu zu definierende Sonderzeichen mit dem niedrigsten Code. Damit die Neudefinition dieses Zeichens möglich wird, müßte die Firmware-Entsprechung des Basic-Befehls 'SYMBOL AFTER 64' angewendet werden. Als Nebeneffekt werden dadurch aber auch die Zeichen mit den Codes 65 bis 255 ins RAM gelegt, so daß die Zeichensatztabelle einen Umfang von vollen 1,5 KByte hat. Diese Methode ist so nur mit einer durch 'MOVCPM 172 *' erzeugten CP/M-Version möglich.

In diesem Programm wurde deshalb zu einer anderen Methode gegriffen: Alle Textausgaben laufen in den CPC-Computern systemintern über den Firmware-Einsprung TXT WRITE CHAR. Dieser Einsprung wird so verändert, daß er auf eine neue Routine zeigt, die zunächst untersucht, welches Zeichen ausgegeben werden soll.

Wenn ein ganz normales Zeichen ausgegeben werden soll, leitet die Routine die Bearbeitung ohne Eingriff an die normale TXT WRITE CHAR-Routine weiter. Wenn aber ein deutsches Sonderzeichen ausgegeben werden soll, wird dessen Code so übersetzt, daß der im Bereich von F7h bis FFh liegt. Jetzt braucht die Tabelle für die 9 deutschen Sonderzeichen also wirklich nur 9 Einträge erfassen und hat damit eine Länge von nur 72 Bytes. Nach der Code-Übersetzung schaltet die geänderte TXT WRITE CHAR-Routine diese Zeichensatztabelle ein und gibt die Bearbeitung erst dann an die normale TXT WRITE CHAR-Routine weiter. Nach der Zeichenausgabe wird die erweiterte Zeichensatztabelle wieder abgeschaltet. Die Codeübersetzung ist nur systemintern wirksam. An der 'Benutzeroberfläche' haben die deutschen Sonderzeichen weiterhin die Standard-Ascii-Codes, so daß mit normalen Druckern und Textprogrammen gearbeitet werden kann.

In den CPC-Computern ist der Speicherbereich von etwa BEC0h bis BFFFh für die Verwendung durch den Maschinenstapel reserviert. Im Normalfall nützt der Stapel diesen Bereich aber gar nicht aus. Ab der Adresse BEC0h ist so genügend Speicherplatz frei, um die kurze Zeichensatzroutine unterzubringen. Das Verschieben des CP/Ms mit MOVCPM erübrigt sich so.

In der Standardversion läuft das Programm in diesem Bereich einwandfrei. Wenn sie irgendwelche Hardwareerweiterungen verwenden, könnte dieser Bereich aber von anderen Zusatzgeräten verwendet werden - dann kommt es zu Problemen. In diesem Fall müssen Sie das Zeichensatzprogramm in einem anderen Adreßbereich unterbringen. Das Assemblerlisting kann dann durch Änderung einer einzigen Zeile verschoben werden. Ein guter Bereich für die Lage des deutschen Zeichensatzes ist zum Beispiel der Kassetten-Fileheaderbuffer. Wenn Sie nicht gerade Files von Diskette auf Kassette kopieren ist dieser Bereich ungenutzt.

Leider liegen im CPC 664/CPC 6128 einige Routinen an anderen Adressen als im CPC 464. Das Programm muß deshalb an die beiden Computer speziell angepaßt werden. Im Assemblerlisting gelingt das einfach durch eintragen von 'CPC646 equ true', wenn Sie einen CPC 464 verwenden oder 'CPC464 equ false', wenn Sie einen CPC 664/6128 verwenden. Im Basic-Lader ist eine Änderung nicht so einfach möglich. Deshalb liegt dieser in 2 Versionen vor.

Auch bei eingeschaltetem deutschen Zeichensatz arbeiten die Firmware-Routinen TXT READ CHAR und GRA WR CHAR mit dem originalen Zeichensatz - die eine erkennt beim Lesen die deutschen Zeichen nicht und die andere gibt weiterhin Ascii-Zeichen auf dem Bildschirm aus. In Standard-CP/M-Anwendungen dürfte das aber bedeutungslos sein.

Beachten Sie, daß einige der deutschen Zeichen in manchen Anwendungen eine besondere Bedeutung haben: Statt eckigen Klammern muß man beispielsweise im PIP-Befehl jetzt ein großes 'Ä' und ein großes 'Ü' verwenden. Die vollständige Tabelle der ersetzten Zeichen zeigt Bild 12-6.

deutsches Sonderzeichen	Ascii-Zeichen
§	@
Ä	[
Ö	\
Ü	]
^	↑
ä	{
ö	|
ü	}
ß	~

Bild 12-6: *Entsprechung deutsche/Ascii-Zeichen*

Deutscher Zeichensatz - das Assemblerlisting

```
                ;Deutscher Zeichensatz unter CP/M 2.2 für Schneider-Computer
                ;
                ;****** Computertyp auswählen
FFFF =          true    equ     0FFFFh
0000 =          false   equ     not true
FFFF =          cpc464  equ     true            ;Bei CPC664 'false' eintragen
                ;
                ;****** Adressen für CPC464
                        if      cpc464
B294 =          mtable  equ     0B294h          ;Angaben über gültige Matrix
134A =          charot  equ     0134Ah          ;Adresse der Bildschirmroutine
                        endif
                ;
                ;****** Adressen für CPC664
                        if      not cpc464
                mtable  equ     0B734h          ;Angaben über definierte Zeiche
                charot  equ     01347h          ;Adresse der Bildschirmroutine
                        endif
                ;
                ;****** Adressen sowohl für CPC464 als auch CPC664
BDD3 =          txtwrt  equ     0BDD3h          ;TXT WRITE CHAR
1000 =          oldstk  equ     1000h           ;lokaler Stapelbereich
B0ED =          ldir    equ     0B0EDh          ;Z80-Befehl 'ldir'
0005 =          bdos    equ     0005h
0080 =          dma     equ     0080h
0009 =          type    equ     09h
                ;
                ;****** Endgültige Lage des Programms bestimmen
BEC8 =          base    equ     0BEC8h          ;hier unbelegter Speicherplatz
0100                    org     0100h
                ;
                ;****** Lokalen Stapel initialisieren
0100 210000             lxi     h,0000h
0103 39                 dad     sp
0104 22FE0F             shld    oldstk-2
0107 31FE0F             lxi     sp,oldstk-2
                ;
                ;****** Wählen, ob einschalten oder ausschschalten
010A 218000             lxi     h,dma           ;Hier steht die Eingabe
010D 7E                 mov     a,m
010E 23                 inx     h
010F FE04               cpi     4               ;1.Byte=Eingabelänge=4
0111 C25B01             jnz     fehler
0114 23                 inx     h
0115 7E                 mov     a,m
0116 23                 inx     h
0117 FE45               cpi     'E'             ;Text 'EIN' erkannt
0119 CAA801             jz      ein
011C FE41               cpi     'A'             ;Text 'AUS' erkannt
011E C25B01             jnz     fehler
                ;
                ;****** Deutschen Zeichensatz abschalten
0121 214A13             lxi     h,charot        ;Betriebssystem-Vektor
0124 22D4BD             shld    txtwrt+1        ;restaurieren
0127 112D01             lxi     d,string1
012A C3BC01             jmp     finis           ;Textausgabe+Warmstart
```

```
012D 0D0A27405Bstring1:db      13,10,'''§ÄÖÜ^äöüß'' Deutscher Zeichensatz '
0151 696E616874        db      'inaktiv',13,10,'$'
                       ;
                       ;****** Fehlerbehandlung
015B 116101    fehler: lxi     d,string2
015E C3BC01            jmp     finis           ;Textausgabe+Warmstart
0161 0D0A556E67string2:db      13,10,'Ungültige Eingabe: Erlaubt ist nur '''
0187 4445555453        db      'DEUTSCH EIN'' und ''DEUTSCH AUS''',13,10,'$'
                       ;
                       ;****** Deutschen Zeichensatz einschalten
                       ;(neuen Betriebssystemvektor eintragen)
01A8 21C8BE    ein:    lxi     h,base
01AB 22D4BD            shld    txtwrt+1
                       ;(Verschieben des Programms an endgültige Position
01AE 11C8BE            lxi     d,base          ;Endgültige Lage
01B1 21F201            lxi     h,start         ;jetzige Lage
01B4 018400            lxi     b,ende-start    ;Programmlänge
01B7 EDB0              dw      ldir            ;Z80-Befehl 'LDIR'
01B9 11C601            lxi     d,string3
                       ;
                       ;****** Programmende, Rücksprung in CCP
01BC 0E09      finis:  mvi     c,type          ;Textausgabe
01BE CD0500            call    bdos
01C1 2AFE0F            lhld    oldstk-2
01C4 F9                sphl
01C5 C9                ret
01C6 0D0A27405Bstring3:db      13,10,'''§ÄÖÜ^äöüß'' Deutscher Zeichensatz '
01EA 616B746976        db      'aktiv',13,10,'$'
                       ;
                       start:
                       ;*************************************************************
                       ;Hier beginnt das eigentliche Programm
                       ;(Umkodieren: deutsche Zeichen in Bereich F7h bis FFh
01F2 FE40              cpi     '§'
01F4 CAE3BE            jz      code40+versatz  ;Codewandlung
01F7 FE5B              cpi     'Ä'
01F9 DA4A13            jc      charot          ;keine Änderung
01FC FE5F              cpi     '_'
01FE DAE5BE            jc      code5f+versatz  ;Codewandlung
0201 FE7B              cpi     'ä'
0203 DA4A13            jc      charot          ;keine Änderung
0206 FE7F              cpi     7Fh
0208 D24A13            jnc     charot          ;keine Änderung
020B C6CA      code7f: adi     0CAh            ;äöüß->Codes FCh-FFh
020D C61A      code40: adi     01Ah            ;§->Code F7h
020F C69D      code5f: adi     09Dh            ;ÄÖÜ^->Codes F8h-FBh
                       ;(selbstdefinierte Matrix einschalten)
0211 E5                push    h
0212 21F7FF            lxi     h,0FFF7h        ;Usermatrix ab Zeichen F7h
0215 2294B2            shld    mtable
0218 2104BF            lxi     h,matrix+versatz
021B 2296B2            shld    mtable+2        ;Adresse der Usermatrix
021E E1                pop     h
                       ;(normale Zeichenausgabe)
021F CD4A13            call    charot
```

```
                ;(Zeichenmatrix ausschalten)
0222 E5                 push    h
0223 210000             lxi     h,0000h
0226 2294B2             shld    mtable
0229 2296B2             shld    mtable+2
022C E1                 pop     h
022D C9                 ret
                ;(Zeichensatztabelle für deutsche Zeichen '§ÄÖÜ↑äöüß')
022E 3C603C6666matrix:  db      03ch,060h,03ch,066h,066h,03ch,006h,03ch
0236 C6386CC6FE         db      0c6h,038h,06ch,0c6h,0feh,0c6h,0c6h,000h
023E C6386CC6C6         db      0c6h,038h,06ch,0c6h,0c6h,06ch,038h,000h
0246 C600C6C6C6         db      0c6h,000h,0c6h,0c6h,0c6h,0c6h,07ch,000h
024E 10386CC600         db      010h,038h,06ch,0c6h,000h,000h,000h,000h
0256 6C00780C7C         db      06ch,000h,078h,00ch,07ch,0cch,076h,000h
025E 6C003C6666         db      06ch,000h,03ch,066h,066h,066h,03ch,000h
0266 6C0000C6C6         db      06ch,000h,000h,0c6h,0c6h,0c6h,07eh,000h
026E 1C36666C66         db      01ch,036h,066h,06ch,066h,066h,0cch,000h
                ;
                ende:
                ;****** Versatz jetzige <-> endgültige Lage
BCD6 =          versatz equ     base-start
0276                    end
```

Die CPC 464-Version des Basic-Laders zum Zeichensatz

```
100 'Basic-Lader fuer die Datei 'DEUTS464.COM'
110 '
120 OPENOUT"DEUTS464.COM"
130 '
140 z%=10000
150 READ d$:WHILE d$<>"Ende"
160 d%=VAL("&"+d$):PRINT#9,CHR$(d%);:p%=d%
170 FOR i%=1 TO 15
180 READ d$:d%=VAL("&"+d$)
190 PRINT#9,CHR$(d%);:p%=p%+d%
200 NEXT
210 READ q%
220 IF p%<>q% THEN PRINT"Tippfehler in Zeile "z%:CLOSEOUT:END
230 z%=z%+10
240 READ d$:WEND
250 CLOSEOUT
260 PRINT"Alles O.K."
270 END
280 '
10000 DATA 21,00,00,39,22,FE,0F,31,FE,0F,21,80,00,7E,23,FE,1287
10010 DATA 04,C2,5B,01,23,7E,23,FE,45,CA,A8,01,FE,41,C2,5B,1784
10020 DATA 01,21,4A,13,22,D4,BD,11,2D,01,C3,BC,01,0D,0A,27,1071
10030 DATA 40,5B,5C,5D,5E,7B,7C,7D,7E,27,20,44,65,75,74,73,1520
10040 DATA 63,68,65,72,20,5A,65,69,63,68,65,6E,73,61,74,7A,1610
10050 DATA 20,69,6E,61,6B,74,69,76,0D,0A,24,11,61,01,C3,BC,1347
10060 DATA 01,0D,0A,55,6E,67,7D,6C,74,69,67,65,20,45,69,6E,1296
10070 DATA 67,61,62,65,3A,20,45,72,6C,61,75,62,74,20,69,73,1460
10080 DATA 74,20,6E,75,72,20,27,44,45,55,54,53,43,48,20,45,1189
10090 DATA 49,4E,27,20,75,6E,64,20,27,44,45,55,54,53,43,48,1148
10100 DATA 20,41,55,53,27,0D,0A,24,21,C8,BE,22,D4,BD,11,C8,1438
10110 DATA BE,21,F2,01,01,84,00,ED,B0,11,C6,01,0E,09,CD,05,1461
10120 DATA 00,2A,FE,0F,F9,C9,0D,0A,27,40,5B,5C,5D,5E,7B,7C,1504
10130 DATA 7D,7E,27,20,44,65,75,74,73,63,68,65,72,20,5A,65,1480
10140 DATA 69,63,68,65,6E,73,61,74,7A,20,61,6B,74,69,76,0D,1557
10150 DATA 0A,24,FE,40,CA,E3,BE,FE,5B,DA,4A,13,FE,5F,DA,E5,2435
10160 DATA BE,FE,7B,DA,4A,13,FE,7F,D2,4A,13,C6,CA,C6,1A,C6,2384
10170 DATA 9D,E5,21,F7,FF,22,94,B2,21,04,BF,22,96,B2,E1,CD,2301
10180 DATA 4A,13,E5,21,00,00,22,94,B2,22,96,B2,E1,C9,3C,60,1659
10190 DATA 3C,66,66,3C,06,3C,C6,38,6C,C6,FE,C6,C6,00,C6,38,1854
10200 DATA 6C,C6,C6,6C,38,00,C6,00,C6,C6,C6,C6,7C,00,10,38,1854
10210 DATA 6C,C6,00,00,00,00,6C,00,78,0C,7C,CC,76,00,6C,00,1100
10220 DATA 3C,66,66,66,3C,00,6C,00,00,C6,C6,C6,7E,00,1C,36,1336
10230 DATA 66,6C,66,66,CC,00,00,00,00,00,00,00,00,00,00,00, 618
10240 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,   0
10250 DATA Ende
```

Die CPC 664-Version des Basic-Laders zum Zeichensatz

```
100 'Basic-Lader fuer die Datei 'DEUTS664.COM'
110 '
120 OPENOUT"DEUTS664.COM"
130 '
140 z%=10000
150 READ d$:WHILE d$<>"Ende"
160 d%=VAL("&"+d$):PRINT#9,CHR$(d%);:p%=d%
170 FOR i%=1 TO 15
180 READ d$:d%=VAL("&"+d$)
190 PRINT#9,CHR$(d%);:p%=p%+d%
200 NEXT
210 READ q%
220 IF p%<>q% THEN PRINT"Tippfehler in Zeile "z%:CLOSEOUT:END
230 z%=z%+10
240 READ d$:WEND
250 CLOSEOUT
260 PRINT"Alles O.K."
270 END
280 '
10000 DATA 21,00,00,39,22,FE,0F,31,FE,0F,21,80,00,7E,23,FE,1287
10010 DATA 04,C2,5B,01,23,7E,23,FE,45,CA,A8,01,FE,41,C2,5B,1784
10020 DATA 01,21,47,13,22,D4,BD,11,2D,01,C3,BC,01,0D,0A,27,1068
10030 DATA 40,5B,5C,5D,5E,7B,7C,7D,7E,27,20,44,65,75,74,73,1520
10040 DATA 63,68,65,72,20,5A,65,69,63,68,65,6E,73,61,74,7A,1610
10050 DATA 20,69,6E,61,6B,74,69,76,0D,0A,24,11,61,01,C3,BC,1347
10060 DATA 01,0D,0A,55,6E,67,7D,6C,74,69,67,65,20,45,69,6E,1296
10070 DATA 67,61,62,65,3A,20,45,72,6C,61,75,62,74,20,69,73,1460
10080 DATA 74,20,6E,75,72,20,27,44,45,55,54,53,43,48,20,45,1189
10090 DATA 49,4E,27,20,75,6E,64,20,27,44,45,55,54,53,43,48,1148
10100 DATA 20,41,55,53,27,0D,0A,24,21,C8,BE,22,D4,BD,11,C8,1438
10110 DATA BE,21,F2,01,01,84,00,ED,B0,11,C6,01,0E,09,CD,05,1461
10120 DATA 00,2A,FE,0F,F9,C9,0D,0A,27,40,5B,5C,5D,5E,7B,7C,1504
10130 DATA 7D,7E,27,20,44,65,75,74,73,63,68,65,72,20,5A,65,1480
10140 DATA 69,63,68,65,6E,73,61,74,7A,20,61,6B,74,69,76,0D,1557
10150 DATA 0A,24,FE,40,CA,E3,BE,FE,5B,DA,47,13,FE,5F,DA,E5,2432
10160 DATA BE,FE,7B,DA,47,13,FE,7F,D2,47,13,C6,CA,C6,1A,C6,2378
10170 DATA 9D,E5,21,F7,FF,22,34,B7,21,04,BF,22,36,B7,E1,CD,2119
10180 DATA 47,13,E5,21,00,00,22,34,B7,22,36,B7,E1,C9,3C,60,1474
10190 DATA 3C,66,66,3C,06,3C,C6,38,6C,C6,FE,C6,C6,00,C6,38,1854
10200 DATA 6C,C6,C6,6C,38,00,C6,00,C6,C6,C6,C6,7C,00,10,38,1854
10210 DATA 6C,C6,00,00,00,00,6C,00,78,0C,7C,CC,76,00,6C,00,1100
10220 DATA 3C,66,66,66,3C,00,6C,00,00,C6,C6,C6,7E,00,1C,36,1336
10230 DATA 66,6C,66,66,CC,00,00,00,00,00,00,00,00,00,00,00, 618
10240 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,   0
10250 DATA Ende
```

12.3.2 Verschiedene Tastenbelegungen unter CP/M

Mit Hilfe der CPC-Firmwareroutinen kann man auch unter CP/M die Tastaturbelegung frei wählen. Einen Befehl analog zum KEYDEF-Befehl in Basic gibt es in CP/M 2.2 aber nicht.

Dieses Programm enthält nun eine ganze Tabelle von umzudefinierenden Tasten: Beim Aufruf wird zunächst die Tastatur in den Ausgangszustand zurückgesetzt und dann die Tabelle verarbeitet. Danach steht sofort eine vollständig neue Tastenbelegung zur Verfügung. Die Belegungstabelle ist so aufgebaut, daß sie leicht geändert werden kann. Unter verschiedenen Namen können Sie sich so auf einer einzigen Diskette Belegungsprogramme mit jeweils unterschiedlichen Tabellen anlegen und jeweils die passende Version aufrufen. Die Umständliche SETUP-Prozedur oder der Systemreset mit Booten mit einer anderen Diskette entfällt so.

Die Belegungstabelle besteht aus 3 Teilen:

Im ersten Teil sind in maximal 32 Tabellensätzen die Zeichenketten der Funktionstasten angegeben. Jeder Tabellensatz enthält dazu die Nummer des Funktionsstrings (0 bis 31), die Länge und die einzelnen Zeichen des Strings. Unabhängig von der Länge der Funktionsstrings folgen die Tabellensätze unmittelbar aufeinander. Das Ende der Tabelle wird durch ein Byte mit dem Wert FFh markiert.

Daran schließt sich unmittelbar eine Tabelle über die Lage der einzelnen Tasten an. Jeder Tabellensatz besteht dabei aus 4 Bytes: Der Tastennummer und dem Ascii-Code der Belegung im Normal-, Shift- und Control-Modus. Die Verwendung eines Codes im Bereich von 80h bis 9Fh bedeutet, daß auf dieser Taste ein Funktionsstring mit einer Nummer zwischen 00h und 1Fh liegt. Der Code FFh markiert eine unbelegte Taste. Dieser Tabellenteil kann wieder beliebig viele Sätze enthalten. Das Ende wird durch ein Byte mit dem Wert FFh markiert.

Der letzte Tabellenteil enthält einen beliebig langen Text, der bei jedem Aufruf des Programms ausgegeben wird. Dieser Text wird durch ein Byte mit dem Wert '$' abgeschlossen. Sinnvollerweise sollte hier eine kurze Bezeichnung der Tastaturbelegung ausgegeben werden. Es ist aber auch möglich Steuerzeichen einzufügen, die etwa die Bildschirmfarben auf einen Standardwert setzen oder den Bildschirmmodus umschalten.

Die im Assemblerlisting verwendete Tastenbelegung ist nur ein Beispiel und kann von Ihnen ohne weiteres geändert werden. Im Basic-Lader wird über die Tabelle keine Prüfsumme berechnet, so daß ein Eingriff ohne weiteres möglich ist. Ein Tabellenelement können Sie sowohl als Hexadezi-

malzahl (ohne vorausgestelltes '&') als auch direkt als Ascii-Zeichen eingeben. Dem Ascii-Zeichen muß in diesem Fall aber ein Ausrufezeichen vorausgehen. Nicht dargestellt werden dürfen auf diese Weise alle Ascii-Zeichen, die in Basiczeilen eine besonderer Bedeutung haben. Das sind die Zeichen

, ; : ' " und | bzw. ö

In diesen Fällen müssen die Ascii-Codes der Zeichen verwendet werden. Bei einer Änderung sollten Sie natürlich auch einen anderen Namen der Tastenbelegung in den ersten Zeilen des Basic-Laders wählen.

Die Beispiel-Tastatur enthält im Zentralblock die deutsche Tastenbelegung und in der Shift- und Control-Ebene des Zehnerblockes Funktionstasten. Die genaue Belegung finden Sie am besten selbst heraus. Natürlich sollten Sie sich eine eigene Tastatur definieren. Eine große Hilfe ist der Aufruf von WORKFILE:

Während Sie ein neues Programm mit dem Namen <Programm> entwickeln, werden Sie den Namen <Programm> für die Aufrufe von Editor, Assembler und so weiter sehr oft benötigen. Durch 'WORKFILE <Programm>' belegen Sie einen Funktionsstring mit dem angegebenen Namen. Durch Druck von CTRL und der kleinen ENTER-Taste erhalten Sie danach immer wieder den Text <Programm> zurück. Das Listing von WORKFILE.COM sehen Sie in Bild 12-7.

```
0100  LXI  H,0080          0109  MVI  B,0C
0103  MOV  A,M             010B  INX  H
0104  SUI  01              010C  INX  H
0106  ACI  00              010D  CALL BE9B
0108  MOV  C,A             0110 0F BB ..
```

Bild 12-7: *Listing von WORKFILE.COM*

Das Assemblerlisting des Tastenbelegers

```
                ;Tastenbelegung auf allen Schneider Computern ändern
                ;               unter CP/M 2.2
                ;
0100                    org     0100h
1000 =          oldsp   equ     1000h   ;Zwei unbelegte Speicherzellen
5000 =          stktop  equ     5000h   ;Lokaler Stapel im zentralen RAM
                ;
                ;       Lokalen Stapel initialisieren
0100 210000     entry:  lxi     h,0
0103 39                 dad     sp
0104 220010             shld    oldsp
0107 310050             lxi     sp,stktop
010A CD00BB             call    0bb00h  ;Keyboard initialisieren
                ;       Expansionsstrings setzen
010D 215401             lxi     h,expan
0110 7E         morex:  mov     a,m
0111 23                 inx     h
0112 3C                 inr     a
0113 CA2601             jz      noexp
0116 3D                 dcr     a
0117 4E                 mov     c,m
0118 23                 inx     h
0119 E5                 push    h
011A 0600               mvi     b,0
011C 09                 dad     b
011D E3                 xthl
011E 47                 mov     b,a
011F CD0FBB             call    0bb0fh
0122 E1                 pop     h
0123 C31001             jmp     morex
                ;       Tastenbelegungen setzen
0126 7E         noexp:  mov     a,m
0127 23                 inx     h
0128 3C                 inr     a
0129 CA4901             jz      nokey
012C 3D                 dcr     a
012D 46                 mov     b,m
012E 23                 inx     h
012F F5                 push    psw
0130 E5                 push    h
0131 CD27BB             call    0bb27h
0134 E1                 pop     h
0135 F1                 pop     psw
0136 46                 mov     b,m
0137 23                 inx     h
0138 F5                 push    psw
0139 E5                 push    h
013A CD2DBB             call    0bb2dh
013D E1                 pop     h
013E F1                 pop     psw
013F 46                 mov     b,m
0140 23                 inx     h
0141 E5                 push    h
0142 CD33BB             call    0bb33h
0145 E1                 pop     h
0146 C32601             jmp     noexp
```

```
                ;       Programmende, Rücksprung in CCP
0149 EB         nokey:  xchg            ;hl enthaelt Adresse des Meldetextes
014A 0E09               mvi     c,9     ;Textausgabe
014C CD0500             call    5
014F 2A0010             lhld    oldsp
0152 F9                 sphl
0153 C9                 ret
                ;Expansionsstrings:
0154 0004444952expan:   db      0,4,'DIR',13
015A 0104444952         db      1,4,'DIR '
0160 020452454E         db      2,4,'REN '
0166 0305535441         db      3,5,'STAT',13
016D 0405535441         db      4,5,'STAT '
0174 0504455241         db      5,4,'ERA '
017A 0604504950         db      6,4,'PIP '
0180 070441534D         db      7,4,'ASM '
0186 08054C4F41         db      8,5,'LOAD '
018D 0905545950         db      9,5,'TYPE '
0194 0A02423A           db      10,2,'B:'
0198 0B09574F52         db      11,9,'WORKFILE '
01A3 0C00               db      12,0
01A5 FF                 db      255             ;Ende Expansionsstrings
                ;Tastenbelegung
01A6 421B1B1B           db      66,27,27,27
01AA 10080808           db      16,8,8,8
01AE 0A378080           db      10,'7',128,128
01B2 0B388181           db      11,'8',129,129
01B6 03398282           db      3,'9',130,130
01BA 14348383           db      20,'4',131,131
01BE 0C358484           db      12,'5',132,132
01C2 04368585           db      4,'6',133,133
01C6 0D318686           db      13,'1',134,134
01CA 0E328787           db      14,'2',135,135
01CE 05338888           db      5,'3',136,136
01D2 0F308989           db      15,'0',137,137
01D6 072E8A8A           db      7,'.',138,138
01DA 060D8B8C           db      6,13,139,140
                ;Deutsche Tasten
01DE 112A3BFF           db      17,42,59,255
01E2 132B3AFF           db      19,43,58,255
01E6 162D60FF           db      22,45,96,255
01EA 183D4000           db      24,61,64,0
01EE 197E5E1E           db      25,126,94,30
01F2 1A7D5D1D           db      26,125,93,29
01F6 1C7B5B1B           db      28,123,91,27
01FA 1D7C5C1C           db      29,124,92,28
01FE 2B7A5A1A           db      43,122,90,26
0202 413222FF           db      65,50,34,255
0206 47795919           db      71,121,89,25
020A FF                 db      255             ;Ende Tasten
                aus:
020B 04020F010Estring:  db      4,2,15,1,14,0,28,0,1,1,28,1,24,24
0219 0D0A1D0101         db      13,10,29,1,1,13,10
0220 4B6F6D6D61         db      'Kommando - Tastenbelegung aktiv'
023F 0D0A24             db      13,10,'$'
0242                    end
```

Der Basic-Lader zum Tastenbeleger

```
100 'Basic-Lader fuer die Datei 'CCPTASTE.COM'
110 '
120 OPENOUT"CCPTASTE.COM"
130 '
140 '1. Teil: Allgemeine Verarbeitung einer Belegungstabelle (mit Pruefsumme)
150 FOR zeile=10010 TO 10270 STEP 10
160 pruef=0
170 FOR i=0 TO 7
180 READ d$:d=VAL("&"+d$)
190 PRINT#9,CHR$(d);:pruef=pruef+d
200 NEXT
210 READ p
220 IF pruef<>p THEN PRINT"Pruefsummenfehler in Zeile"zeile:CLOSEOUT:END
230 NEXT
240 '
250 '2. Teil: Daten ueber die spezielle Tastenbelegung (ohne Pruefsumme)
260 READ d$: WHILE d$<>"Ende"
270 IF LEFT$(d$,1)="!" THEN p$=RIGHT$(d$,1) ELSE p$=CHR$(VAL("&"+d$))
280 PRINT#9,p$;
290 READ d$:WEND
300 '
310 CLOSEOUT:PRINT"Alles O.K.":END
320 '
330 '

10000 'Hauptprogramm
10010 DATA C3,86,01,00,08,08,08,54, 438
10020 DATA 61,73,74,65,6E,62,65,6C, 846
10030 DATA 65,67,75,6E,67,20,66,7D, 793
10040 DATA 72,20,53,63,68,6E,65,69, 748
10050 DATA 64,65,72,20,43,50,43,34, 613
10060 DATA 36,34,20,65,69,6E,73,74, 685
10070 DATA 65,6C,6C,65,6E,0D,0A,0D, 564
10080 DATA 0A,43,6F,70,79,72,69,67, 743
10090 DATA 68,74,20,28,63,29,20,31, 513
10100 DATA 31,2E,30,39,2E,31,39,38, 408
10110 DATA 35,0D,0A,48,65,6C,6D,75, 583
10120 DATA 74,20,54,69,73,63,68,65, 756
10130 DATA 72,0D,0A,41,73,74,65,72, 648
10140 DATA 6E,73,74,72,61,7E,65,20, 811
10150 DATA 34,30,2C,20,44,2D,38,30, 393
10160 DATA 35,32,20,4D,6F,6F,73,62, 647
10170 DATA 75,72,67,0D,0A,1A,21,00, 416
10180 DATA 00,39,22,00,10,31,00,50, 236
10190 DATA CD,00,BB,21,DA,01,7E,23, 805
10200 DATA 3C,CA,AC,01,3D,4E,23,E5, 838
10210 DATA 06,00,09,E3,47,CD,0F,BB, 720
10220 DATA E1,C3,96,01,7E,23,3C,CA, 994
10230 DATA CF,01,3D,46,23,F5,E5,CD,1053
10240 DATA 27,BB,E1,F1,46,23,F5,E5,1271
10250 DATA CD,2D,BB,E1,F1,46,23,E5,1237
10260 DATA CD,33,BB,E1,C3,AC,01,EB,1271
10270 DATA 0E,09,CD,05,00,2A,00,10, 291
10280 DATA F9,C9
10290 '
10300 '
```

```
10310 'Tabellenbereich des Programms:
10320 'darf vom Anwender angepasst werden (ohne Pruefsumme)
10330 'Entweder - Hexzahlen
10340 '    oder - '!', gefolgt von Ascii-Zeichen
10350 '
10360 '
10370 '***** Strings auf den Funktionstasten
10380 'jeweils Nummer, Laenge und Zeichenkette
10390 DATA 00,04,!D,!I,!R,0D
10400 DATA 01,04,!D,!I,!R,20
10410 DATA 02,04,!R,!E,!N,20
10420 DATA 03,05,!S,!T,!A,!T,0D
10430 DATA 04,05,!S,!T,!A,!T,20
10440 DATA 05,04,!E,!R,!A,20
10450 DATA 06,04,!P,!I,!P,20
10460 DATA 07,04,!A,!S,!M,20
10470 DATA 08,05,!L,!O,!A,!D,20
10480 DATA 09,05,!T,!Y,!P,!E,20
10490 DATA 0A,02,!B,3A
10500 DATA 0B,09,!W,!O,!R,!K,!F,!I,!L,!E,20
10510 DATA 0C,00
10520 'Code fuer Tabellenende
10530 DATA FF
10540 '
10550 '***** Tastenbelegung
10560 'Lage der Funktionstasten
10570 DATA 42,1B,1B,1B
10580 DATA 10,08,08,08
10590 DATA 0A,!7,80,80
10600 DATA 0B,!8,81,82
10610 DATA 03,!9,82,82
10620 DATA 14,!4,83,83
10630 DATA 0C,!5,84,84
10640 DATA 04,!6,85,85
10650 DATA 0D,!1,86,86
10660 DATA 0E,!2,87,87
10670 DATA 05,!3,88,88
10680 DATA 0F,!0,89,89
10690 DATA 07,!.,8A,8A
10700 DATA 06,0D,8B,8C
10710 'Deutsche Umlaute
10720 DATA 1A,!},!],1D
10730 DATA 1C,!{,![,1B
10740 DATA 1D,7C,!\,1C
10750 DATA 2B,!z,!Z,1A
10760 DATA 47,!y,!Y,19
10770 DATA 19,!ß,!^,1E
10780 'Sonderzeichen im Zentralblock
10790 DATA 11,!*,!;,FF
10800 DATA 13,!+,3A,FF
10810 DATA 16,!-,!',FF
10820 DATA 18,!=,!§,00
10830 DATA 41,!2,22,FF
10840 'Code Tabellenende
10850 DATA FF
10860 '
```

```
10870 '***** Meldetext:
10880 'Farbwahl+ "KOMMANDO -Tastenbelegung aktiv"
10890 DATA 04,02,0F,01,0E,00,1C,00
10900 DATA 01,01,1C,01,18,18,0D,0A
10910 DATA 1D,01,01,0D,0A,!K,!o,!m
10920 DATA !m,!a,!n,!d,!o,20,!-,20
10930 DATA !T,!a,!s,!t,!e,!n,!b,!e
10940 DATA !l,!e,!g,!u,!n,!g,20,!a
10950 DATA !k,!t,!i,!v,0D,0A
10960 'Abschlusszeichen
10970 DATA 24
10980 '
10990 'Programmende
11000 DATA Ende
```

Anhänge

Anhang A: Die Standard-CP/M-Befehle

Die CCP-Befehle

`DIR <Dateinamen>`
(Teil-)Disketteninhaltsverzeichnis ausgeben

`ERA <Dateinamen>`
Datei(en) löschen

`REN <Dateiname1>=<Dateiname2>`
Datei umbenennen

`TYPE <Dateiname>`
Datei auf Bildschirm ausgeben

`SAVE <Länge> <Dateiname>`
RAM-Inhalt in der Länge 256*<Länge> abspeichern

`USER <Nummer>`
Benutzernummer wechseln

`<Laufwerk>:`
Bezugslaufwerk wechseln, erlaubt ist 'A' bis 'P'

Die STAT.COM-Befehle

`STAT VAL:`
Ausgabe einer Tabelle über die mit STAT erlaubten Eingaben

`STAT DEV:`
Tabelle über die aktuelle Zuordnung reale/virtuelle Geräte

`STAT <Kanal>:=<Gerät>:`
Dem logischen Ein-/Ausgabekanal <Kanal> wird das reale Ein-/Ausgabegerät <Gerät> zugwiesen. Eine Liste der erlaubten Geräte und Kanäle zeigt 'STAT VAL:'.

`STAT USR:`
Übersicht über die belegten Userbereiche ausgeben

`STAT DSK:`
Merkmale der aktiven Diskettenlaufwerke feststellen

`STAT <Laufwerk>:DSK:`
Merkmale des angegebenen Laufwerkes feststellen

`STAT`
Angabe über Schreibschutz und Speicherplatz der Disketten

`STAT <Laufwerk>:`
Angabe über freien Speicherplatz der angegebenen Diskette

`STAT <Dateinamen>`
Dateimerkmale und Größe der angegebenen Dateien anzeigen

`STAT <Dateinamen> $<Dateimerkmal>`
Die angegebenen Dateien erhalten die Merkmale 'R/O', 'R/W' (schreibgeschützt oder nicht) , 'SYS' oder 'DIR' im (Inhaltsverzeichnis sichtbar oder nicht).

`STAT <Laufwerk>:=R/O`
Das angegebene Laufwerk wird bis zum nächsten Warmstart in den schreibgeschützten Zustand versetzt.

Die PIP.COM-Befehle

`PIP` (ohne Parameter)

Geht in den Interaktiv-Modus, dann können mehrere Befehlszeilen eingegeben werden.

`PIP <Befehlszeile>`

Führt das angegebene Kommando aus und bricht sofort ab.

Grundform einer PIP-Befehlszeile:

`<Zieldatei>=<Herkunftsliste><Optionenliste>`

Eine <Zieldatei> kann sein:

- Ein Laufwerksname `<Laufwerk>:`
- Ein einfacher Dateiname `<Dateiname>`
- Ein Ausgabekanal `CON: PUN: LST: PRN: OUT:`

(PRN: entspricht dabei einem LST:-Kanal mit gesetzten Optionen [NPT8], OUT: einem benutzerdefinierbaren Ausgabegerät)

Eine <Herkunftsliste> besteht:

- Aus mehreren <Quelldateien>, getrennt durch Kommas.
- Bei Quelldateien können auch Mehrfachnamen (mit '*' und '?') verwendet werden.

Eine <Quelldatei> kann sein:

- Ein Mehrfachname `<Dateinamen>`
- Ein Eingabekanal `CON: RDR: INP: NUL: EOF:`

(INP: entspricht dabei einem benutzerdefinierbaren Eingabegrerät, NUL: entspricht 40 Null-Bytes und EOF: einer Textendemarkierung ^Z)

Eine <Optionenliste> kann auch fehlen. Wenn vorhanden, besteht sie aus:

- Einer Liste von <Optionen>, eingeschlossen durch eckige Klammern.
- Die Optionen bestehen in der Regel nur aus einem einzigen Buchstaben.
- In der Optionenliste folgen diese Buchstaben in beliebiger Reihenfolge, ohne Abstand aufeinander.

Erlaubt sind die folgenden <Optionen>:

L	Beim Kopieren alle Großbuchstaben in Kleinbuchstaben wandeln.
U	Beim Kopieren alle Kleinbuchstaben in Großbuchstaben wandeln.
Z	Beim Kopieren das Bit 7 aller Bytes löschen.
N	Beim Kopieren Zeilennummern voransetzen, ohne führende Nullen, mit abschließendem Doppelpunkt.
N2	Beim Kopieren Zeilennummern voransetzen, mit führenden Nullen, abgeschlossen durch ^I (TAB-Zeichen).
Tn	Spaltenbreite für den Tabulator auf *n* setzen und Tabulatoren durch eine passende Zahl von Leerzeichen ersetzen.
Dn	Beim Kopieren alle Textteile hinter der Spalte n löschen.
F	Beim Kopieren alle Seitenvorschübe löschen.
Pn	Beim Kopieren alle n Zeilen Seitenvorschübe einfügen.
S<Text>^S	Übertragung in die Zieldatei erst bei erreichen der Textstelle <Text> beginnen.
Q<Text>^S	Kopieren bei erreichen der Textstelle <Text> beenden.
E	Alle gelesenen Zeichen werden zusätzlich auf dem Bildschirm dargestellt.
V	Die Aufzeichnung in der Zieldatei wird auf Übertragungsfehler geprüft (nur bei Übertragung von Diskettendateien).
O	Der Dateinhalt wird ohne Überprüfung auf ^Z übertragen. das ist nur beim Zusammenfassen von Nicht-COM-Files nötig.
W	Eine schon vorhandene schreibgeschützte Datei mit demselben Namen wie die Zieldatei wird ohne Warnung gelöscht.
R	Es werden auch Dateien gelesen, die im Inhaltsverzeichnis nicht sichtbar sind.
Gn	Die Quelldatei liegt im Benutzerbereich *n* der Diskette
B	Beim Empfangen eines Zeichens ^S wird das Einlesen angehalten.
H	Beim Lesen wird geprüft, ob die Datei den Normen einer Intel HEX-Datei entspricht.
I	Beim Zusammenfassen von HEX-Dateien werden Zeilen, die mit ':00' beginnen, ausgelassen.

Beispiel:

```
PIP
*A:NEU.TXT=B:ALT.TXT[N2UT8]
*
```

Die Datei ALT.TXT wird in die Datei NEU.TXT kopiert, wobei alle Tabulatoren durch Leerzeichen ersetzt werden, jeder Zeile eine Zeilennummer vorangesetzt wird und alle Klein- in Großbuchstaben umgewandelt werden.

Die ED.COM-Befehle

Der Texteditor wird durch 'ED <Dateiname>' gestartet. Danach kann der Text aus <Dateiname> mit den folgenden Befehlen bearbeitet werden:

nA	*n* Textzeilen ab der ersten noch nicht gelesenen Zeile aus der Datei <Dateiname> in den Textpuffer holen und am Ende anfügen.
nW	*n* Textzeilen vom Anfang des Textpuffers in eine Ergebnisdatei schreiben.
nX	*n* Textzeilen ab der aktuellen Textposition in eine Zwischendatei schreiben.
R	Zwischendatei an der aktuellen Textposition einfügen.
R<Datei>	Text aus der Datei <Datei>.LIB an der aktuellen Textposition einfügen.
+/-B	Aktuelle Textposition an Textanfang/Textende stellen.
+/-nC	Aktuelle Textposition um *n* Spalten verschieben.
+/-nL	Aktuelle Textposition um *n* Zeilen verschieben.
nF<Text>	Das *n*-te Auftreten des Textes <Text> im Textpuffer aufsuchen.
nN<Text>	Das *n*-te Auftreten des Textes <Text> im Textpuffer aufsuchen. Wenn die Position nicht gefunden wird, werden automatisch Zeilen aus dem Textuffer nachgeladen.
+/-n	aktuelle Textposition um *n* Zeilen verschieben und die erreichte Zeile auf dem Bildschirm ausgeben.
+/-nT	*n* Textzeilen ab der aktuellen Position auf dem Bildschirm ausgeben.
+/-nP	*n* Bildschirmseiten ab der aktuellen Position auf dem Bildschirm ausgeben und die aktuelle Position auf die erreichte Zeile stellen.
I	In Einfügemodus umschalten.
I<Text>^Z	Den Text <Text> an der aktuellen Position einfügen.
+/-nD	*n* Zeichen löschen.
+/-nK	*n* Zeilen an der aktuellen Position löschen.
nS<gelöscht>^Z<eingefügt>^Z	Der Text <gelöscht> wird im Textpuffer gesucht, gelöscht und durch den Text <eingefügt> ersetzt.
nJ<suchen>^Z<eingefügt>^Z<gelöscht<^Z	Der Text <suchen> wird im Textpuffer aufgesucht, danach der Text <eingefügt> eingefügt und der Text bis zur Stelle <gelöscht> gelöscht.
E	Bearbeitung eines Textes beenden. Dabei wird die alte Originaldatei in den Namen <Datei>.BAK umbenannt, und die Zieldatei in den Namen der Originaldatei umbenannt.

H	Die bisher gemachten Änderungen auf der Diskette sichern.
O	Alle seit dem letzten Sichern der Datei gemachten Änderungen löschen.
Q	Das Programm ED.COM beenden und alle seit dem letzten Aufruf gemachten Änderungen löschen.
nM<Befehle>	Die ED-Befehle in der Liste <Befehle> werden *n*-mal abgcarbeitet.
OV	Angeben wieviel Speicher belegt und noch frei ist.
+/-V	Zeilennummernangaben ein-/ausschalten
+/-U	Automatische Umwandlung in Großbuchstaben Ein-/Ausschalten.
nZ	*n*-mal eine Verzögerungsschleife durchlaufen (Wichtig zum Beispiel beim M-Befehl

Bei der Ausführung der Editor-Befehle kann ein Fehler mit der folgenden Meldung auftreten:

```
BREAK <Fehlercode> AT <fehlerhafter Befehl>
```

Der <Fehlercode> kann dabei die folgenden Werte annehmen:

?	Ein nichtexistenter Befehl wurde eingegeben, zum Beispiel 'Y'.
>	Entweder ist der Textpuffer voll, oder ein einzelner Befehl ist zu lang.
#	Der Befehl konnte nicht sooft ausgeführt werden, wie im N-Befehl verlangt.
O	Die bei einem R-Befehl angesprochene Datei ist nicht vorhanden.

Die ASM.COM-Befehle und Fehler

Aufruf: ASM <Dateiname>.<ASM-Datei><HEX-Datei><PRN-Datei>

dabei ist:

<ASM-Datei> Laufwerk, von der die Quell-Datei genommen wird
<HEX-Datei> Laufwerk, auf der die HEX-Datei abgelegt wird
(Z, wenn Datei nicht erzeugt werden soll)
<PRN-Datei> Laufwerk, auf der die PRN-Datei abgelegt wird
(Z, wenn Datei nicht erzeugt werden soll
X, wenn Datei auf dem Bildschirm ausgegeben wird)

Arithmetische/logische Operationen

```
a   +   b   Addition                              Vorrang: 4
a   -   b   Subtraktion                                   4
    +   b   Vorzeichen positiv                            4
    -   b   Vorzeichen negativ                            4
a   *   b   Vorzeichenlose Multiplikation                  5
a   /   b   Vorzeichenlose Division                        5
a MOD   b   Rest von a/b                                   5
  NOT   b   bitweise Negation von b                        3
a AND   b   bitweise Und-Verknüpfung                       2
a OR    b   bitweise Oder-Verknüpfung                      1
a XOR   b   bitweise Exclusiv-oder-Verknüpfung             1
a SHL   b   a nach links schieben, b mal                   5
a SHR   b   a nach rechts schieben, b mal                  5
```

Pseudobefehle

```
ORG   Adresse des Daten-/Programmbereichs festlegen
END   Programmende, eventuell Einsprungadresse
EQU   Wert an Konstante zuweisen
SET   Wert an variable zuweisen
IF    Start einer bedingten Assemblierung
ENDIF Ende einer bedingten Assemblierung
DB    Datenbytes einfügen
DW    Datenworte einfügen
DS    Speicherplatz reservieren
```

ASM-Fehlermeldungen

D Data Error: Falscher Datentyp im Datenbereich.
E Expression Error: Der Ausdurck ist nicht korrekt.
L Label Error: Marke darf an dieser Stelle nicht verwendet werden.
N Not implemented: In dieser Assemblerversion ist der verwendete Befehl nicht implementiert.
O Overflow: Einzu berechnender Ausdruck ist zu kompliziert.
P Phase error: Beim ersten und beim zweiten Assemblerdurchlauf hat eine Marke nicht den gleichen Wert.
R Register Error: Ein nicht erlaubtes Register wurde adressiert.
S Syntax error: Eine Befehlszeile ist nicht korrekt.
U Undefined Label: Die hier verwendete Marke wurde nicht definiert.
V Value Error: Ein Operand in einem Ausdruck ist nicht korrekt.

Die DDT.COM-Befehle

`DDT <Dateiname>`
: Beim Start wird die Datei <Dateiname> sofort zur Untersuchung in den Speicher geladen.

`DDT`
: Das DDT-Programm wird gestartet, ohne eine Datei zu laden.

`A<Start>`
: Erlaubt die Dateneingabe ab der Adresse <Start> in 8080-Assemblercode.

`D<Start>,<Ende>`
: Gibt einen Speicherbereich in Hex-code aus.

`F<Start>,<Ende>,<Füllbyte>`
: Füllt angegebenen Bereich mit einem konstantem Wert.

`G<Start>,<Breakpoint1>,<Breakpoint2>`
: Das Programm wird gestartet und läuft solange ohne Unterbrechung, bis es einen der beiden Breakpoints erreicht.

`H<Zahl1>,<Zahl2>`
: Gibt die hexadezimale Summe und Differenz der beiden angegebenen Zahlen aus.

`I<Dateiname>`
: Bereit das Einlesen einer Datei vor.

`L<Start>,<Ende>`
: Listet einen Speicherbereich in 8080-Assembler-Mnemonic.

`M<Start>,<Ende>,<Ziel>`
: Kopiert einen Speicherbereich an eine andere Stelle.

`R<Versatz>`
: Liest eine Datei von der Diskette ein.

`S<Start>`
: Ermöglicht hexadezimale Eingabe.

`T<Anzahl>`
: Führt eine Anzahl Programmschritte als Einzelschritt mit Bildschirmausgabe aus.

`U<Anzahl>`
: Führt eine Anzahl Programmschritte ohne Anzeige der Zwischenergebnisse aus.

`X<Register>`
: Zeigt oder ändert bestimmte CPU-Register.

Anhang B:

Speicherbelegungen und Aufbau wichtiger Tabellen

Form von Dateinamen

`<Dateiname>:  <Laufwerksname><Name>.<Typ>`

- Der <Laufwerksname> besteht aus einem Buchstaben A bis P, gefolgt von einem Doppelpunkt. Der Laufwerksname kann auch wegelassen werden, dann bezieht sich die Operation auf eine Datei im aktuellen Laufwerk.
- Bei der Eingabe besteht der <Name> aus 0 bis 8 Ascii-Zeichen. Bei der systeminternen Verwendung wird der Name immer mit Leerzeichen auf die Länge 8 erweitert.
- Der <Typ> besteht bei der Eingabe aus 0 bis 3 Ascii-Zeichen. Bei der systeminternen Verwendung wird der Typ immer mit Leerzeichen auf die Länge 3 aufgefüllt.
- Systemintern wird der Punkt zwischen <Name> und <Typ> nicht Verwendet. Auch in der Eingabe darf der Punkt fehlen, wenn der <Typ> die Länge Null hat.
- Ein <Dateiname>, der Fragezeichen '?' enthält, bezieht sich auf alle Dateien, die in ihrem Namen an der Position des Fragezeichens ein beliebiges Zeichen haben. Ein Sternchen '*' dient als Abkürzung für mehrere Fragezeichen bis zum Namen-/Typende.

In Dateinamen sind folgende Zeichen verboten:

- Steuerzeichen mit Codes von 0 bis 1Fh und Delete (Code 7Fh)
- Kleinbuchstaben von 'a' bis 'z', ohne Umlaute
- In einem Standard-CP/M-System verboten sind die Zeichen:

  < > . , ; : = _ [ ] Ä Ü

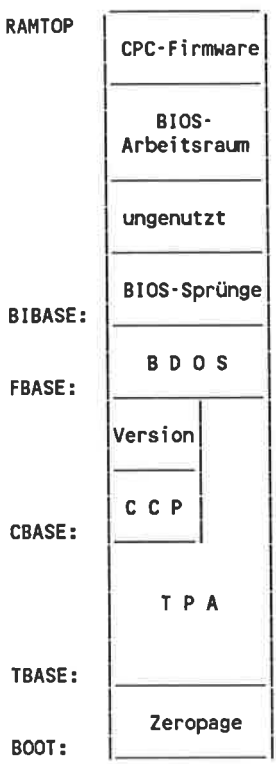
- In den CPC-Computern, aber nur bei der Verwendung in Basic, auch die Zeichen:

  ( ) % / | \ ö Ö

- Bei der Verwendung eines Vortex-Diskcontrollers in Basic auch die Zeichen:

  { } ä ü

Die Speicheraufteilung der CPC-Computer unter CP/M



	Lage relativ zum FDOS	normale Lage im CPC 464/664 (ohne Verschiebung)
RAMTOP	FFFFh	
CPC-Firmware	B100h	
BIOS-Arbeitsraum	B0FFh	
	AD33h	
ungenutzt	AD32h	
	FBASE+0E2Dh	AD33h
BIOS-Sprünge	FBASE+0E2Ch	AD32h
BIBASE:	FBASE+0DFAh	AD00h
B D O S	FBASE+0DF9h	ACFFh
FBASE:	FBASE	9F06h
Version	FBASE-0001h	9F05h
	FBASE-0005h	9F00h
C C P	FBASE-0006h	9EFFh
CBASE:	FBASE-0806h	9700h
T P A	FBASE-0807h	96FFh
TBASE:	0100h	0100h
Zeropage	00FFh	00FFh
BOOT:	0000h	0000h

Die Belegung der Zeropage

Adresse	Inhalt/Bedeutung
0000h-0002h	Warmstart-Einsprung
0003h	Beinhaltet das Input/Output-Byte
0004h	Bits 0 bis 3: Nummer des Bezugslaufwerks Bits 4 bis 7: aktuelle Usernummer.
0005h-0007h	BDOS-Einsprungadresse
0008h-002Fh	Standard-CP/M: nicht genutzt CPC-Computer: Bankswitchinglogik
0030h-0037h	Standard-CP/M: reserviert für spätere Verwendung durch neue CP/M-Versionen CPC-Computer: übernimmt Aufgabe des Bereiches 0038h bis 003Fh eines Standard-CP/M-Rechners
0038h-003Ah	Standard-CP/M: nur vom DDT-Programm benötigt CPC-Computer: Interrupt-Einsprung
003Bh-003Fh	Standard-CP/M: reserviert für spätere Verwendung durch neue CP/M-Versionen CPC-Computer: Einsprung externer Interrrupt
0040h-004Fh	kann vom BIOS verwendet werden, im CPC aber nicht genutzt
0050h-005Bh	reserviert für spätere Verwendung
005Ch-007Ch	Standard-Dateibeschreiber
007Dh-007Fh	erweiterter Dateibeschreiber
0080h-00FFh	Standard-Datenpuffer

Bedeutung des Input/Output-Bytes

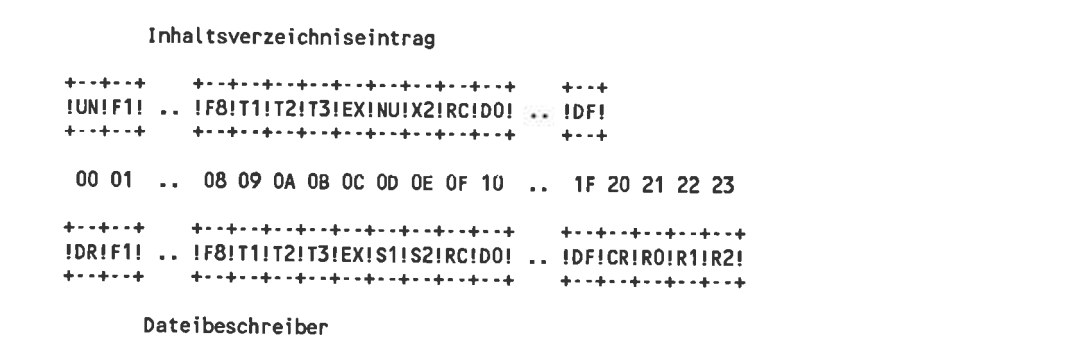
```
Bits 0/1:  Zuordnung eines realen Gerätes zum Konsolenkanal (CON:)
      00: (TTY:) Ein-/Ausgabe über SIO1-Gerät
      01: (CRT:) Ein-/Ausgabe über Bildschirm und Tastatur
      10: (BAT:) 3"-Floppy: Eingabe über den Lochstreifenleser
                            Ausgabe auf den Druckerkanal
                 Vortex:    nicht implementiert
      11: (UC1:) Ein-/Ausgabe über SIO2-Gerät

Bits 2/3:  Zuordnung eines realen Gerätes zum Lochstreifenleserkanal (RDR:)
      00: (TTY:) Eingabe über SIO1-Gerät
      01: (PTR:) nicht implementiert
      10: (UR1:) Eingabe über SIO2-Gerät
      11: (UR2:) Eingabe über die Tastatur

Bits 4/5:  Zuordnung eines realen Gerätes zum Lochstreifenstanzerkanal (PUN:)
      00: (TTY:) Ausgabe über SIO1-Gerät
      01: (PTP:) nicht implementiert
      10: (UP1:) Ausgabe über SIO2-Gerät
      11: (UP2:) Ausgabe auf den Bildschirm

Bits 6/7:  Zuordnung eines realen Gerätes zum Druckerkanal (LST:)
      00: (TTY:) Ausgabe über SIO1-Gerät
      01: (CRT:) 3"-Floppy: Ausgabe auf den Bildschirm
                 Vortex:    nicht implementiert
      10: (LPT:) Ausgabe auf den Centronics-Drucker
      11: (UL1:) Ausgabe über SIO2-Gerät
```

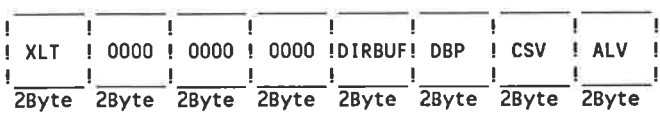
Aufbau eines Inhaltsverzeichniseintrages/ Dateibeschreibers



DR	Laufwerksnummer; 0=Bezugslaufwerk; 1..16: A bis P
UN	Benutzernummer, unter der die Datei zu finden ist
F1...F8	Bits 0 bis 6: Dateiname
	Bits 7:
F1'..F4'	können vom Anwender beliebig belegt werden
F5'..F8'	reserviert für spätere CP/M-Versionen
T1...T3	Bits 0 bis 6: Typ der Datei
	Bits 7:
T1'	gesetzt, wenn Datei schreibgeschützt ist
T2'	gesetzt, wenn Datei bei 'DIR' nicht gelistet wird
T3'	reserviert für spätere Betriebssystemversionen
EX	Bits 0 bis 4: niederwertige Bits der logischen Extentnummer des Eintrages
NU	Nicht genutzt, immer Null
S1	Für die interne Verwendung durch das BDOS reserviert.
X2	Bits 0 bis 3: höherwertige Bits der logischen Extentnummer des Eintrages.
S2	Für die interne Verwendung durch das BDOS reserviert
RC	Anzahl der Records in diesem Extent
D0...DF	Nummer der von diesem Extent belegten Blöcke
CR	gerade behandelter Record des logischen Extents
R0...R2	Recordnummer beim wahlfreien Dateizugriff

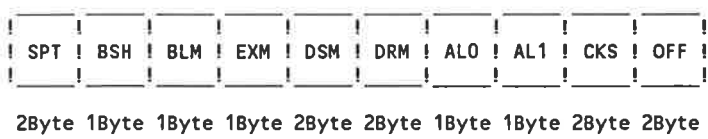
Die Diskettenbeschreiber

Der Disk-Parameter-Header



XLT	Adresse der Sektorübersetzungstabelle dieses Laufwerkes.
0000	Vom BDOS benötigte Zwischenspeicher.
DIRBUF	Adresse eines 128 Byte langen Zwischenspeichers für Inhaltsverzeichnisoperationen; wird von allen Laufwerken gemeinsam benutzt.
DBP	Adresse des Disk-Paramter-Blocks dieses Laufwerks.
CSV	Adresse eines Zwischenspeicher, der für jeden Verzeichnis-record dieses Laufwerks ein Byte aufnehmen kann.
ALV	Adresse eines Zwischenspeichers, der für jeden Block dieses Laufwerks ein Bit aufnehmen kann.

Der Disk-Parameter-Block



SPT	Gesamtzahl der Records pro Spur
BSH	Blockgröße: Ein Block enthält 128*2^(BSH) Bytes.
BLM	Hängt vom BSH-Wert ab: BLM=(2^BSH)-1
EXM	Anzahl der logischen Extents pro physischem Extent-1
DSM	Gesamtblockzahl der Diskette Minus 1
DRM	Anzahl der Inhaltsverzeichniseinträge Minus 1
AL0, AL1	Jedes 1-Bit reserviert einen Block für das Directory
CKS	Anzahl der überprüften Verzeichnis-Records
OFF	Anzahl der Systemspuren

Anhang C: Die Betriebssystem-Funktionen

Die BDOS-Funktionen

Nummer	Funktions-Bezeichnung	Aufruf	Ergebnis
0	Warmstart	----	----
1	Konsoleneingabe	----	a: Zeichen
2	Konsolenausgabe	e:Zeichen	----
3	Reader-Eingabe	----	a:Zeichen
4	Puncher-Ausgabe	e:Zeichen	----
5	Druckerausgabe	e:Zeichen	----
6	Direkte Konsolen-Ein-/Ausgabe	e:255	a:Zeichen
		e:Zeichen	----
7	I/O-Byte lesen	----	a:I/O-Byte
8	I/O-Byte setzen	e:I/O-Byte	----
9	Zeichenkette ausgeben	de:Stringadr.	----
10	Zeichenkette einlesen	de:Pufferadr.	----
11	Konsolenstatus abfragen	----	a:255=Zeich. 0=keines
12	CP/M-Versionsnummer abfragen		hl:Version
13	Diskettensystem zurücksetzen	----	----
14	Bezugslaufwerk wählen	e:Laufwerk	----
15	Datei eröffnen	de:FCB	a:0..3=Ok. 255=Fehler
16	Datei schließen	de:FCB	a:0..3=Ok. 255=Fehler
17	ersten Verzeichniseintrag lesen	de:FCB	a:0..3=Ok. 255=Fehler
18	weiteren Verzeichniseintrag		a:0..3=Ok. 255=Fehler
19	Datei löschen	de:FCB	a:0..3=Ok. 255=Fehler
20	Sequentielles Lesen	de:FCB	a:0..3=Ok. 255=Fehler
21	Sequentielles Schreiben	de:FCB	a:0..3=Ok. 255=Fehler
22	Datei erzeugen	de:FCB	a:0..3=Ok. 255=Fehler
23	Datei umbenennen	de:FCB	a:0..3=Ok. 255=Fehler
24	Aktive Laufwerke feststellen	----	hl:LW-Vektor
25	Bezugslaufwerk feststellen	----	a:Laufwerk
26	Pufferspeicheradresse festlegen	de:Pufferadr.	----
27	Adresse der Belegungstabelle	----	hl:ALV-Adr.
28	Bezugslaufwerk schreibschützen	----	----
29	Schreibschutz feststellen	----	hl:LW-Vektor
30	Dateiattribute ändern	de:FCB	a:0..3=Ok. 255=Fehler
31	Adresse Disk-Parameter-Block	----	hl:DPB-Adr.
32	Usernummer abfragen:	e:255	a:Usernummer
	setzen:	e:Usernummer	----

33	Wahlfreies Lesen	de:eFCB	a:0=Ok.
	(nicht geschriebene Daten lesen)		1=Fehler
	(Extent kann nicht geschlossen werden)		3=Fehler
	(nicht existierender Extent adressiert)		4=Fehler
	(Recordnummer größer 65535)		6=Fehler
34	Wahlfreies Schreiben	de:eFCB	a:0=Ok.
	(Extent kann nicht geschossen werden)		3=Fehler
	(Inhaltsverzeichnis voll)		5=Fehler
	(Diskette voll oder Recordnummer > 65535)		6=Fehler
35	Dateigröße berechnen	de:eFCB	....
36	Aktuelle Recordnummer berechnen	de:eFCB	....
37	Laufwerk zurücksetzen	de:LW-Vektor	....
40	Wahlfreies Schreiben mit Blockinitialisierung	de:eFCB	a:wie Funkt. Nr. 34

Die Standard-BIOS-Routinen

Nummer/Funktions-Bezeichnung	Aufruf	Ergebnis
0 Kaltstart	----	----
1 Warmstart	----	----
2 Konsolenstatus	----	a:255=Zeich. 0=keines
3 Konsoleneingabe	----	a:Zeichen
4 Konsolenausgabe	c:Zeichen	----
5 Druckerkanalausgabe	c:Zeichen	----
6 Puncherkanalausgabe	c:Zeichen	----
7 Readerkanaleingabe	----	a:Zeichen
8 Laufwerk auf Spur Null	----	----
9 Laufwerk auswählen (bei Auto-Formaterkennung:)	c:LW e:Bit0=0	hl:DPH des LW
10 Spurnummer wählen	bc:Nummer	----
11 Sektornummer wählen	bc:Nummer	----
12 Datenpuffer wählen	bc:DMA-adr.	----
13 gewählten Sektor lesen	----	a:0=o.k. 1=Fehler
14 gewählten Sektor schreiben (Schreibens ins Directory) (Schreiben in unbelegten Block)	c:0=normal 1=Directory 2=neuen Block	a:0=o.k. 1=Fehler
15 Druckerkanal-Status	----	a:0=warten 255=Ok.
16 Sektornummer Übersetzen	bc:logische de:XLT-Tabelle	hl:physische

Die zusätzlichen BIOS-Einsprünge der CPC-Computer

Adresse	Funktions-Bezeichnung	Aufruf	Ergebnis
BE80	BIOS-Meldungen sperren	a:0=erlaubt 255=gesperrt	----
BE83	Zeitkonstanten für Disks	hl:Zeittabelle	----
BE86	Diskettenparameter wählen	a:Formatnr.	----
BE89	physischen Sektor lesen	e:Laufwerk d:Spur c:Sektor hl:Puffer	cy:0=Fehler 1=Ok.
BE8C	physischen Sektor schreiben	e:Laufwerk d:Spur c:Sektor hl:Puffer	cy:0=Fehler 1=Ok.
BE8F	Spur formatieren	e:Laufwerk d:Spur hl:Parameter	cy:0=Fehler 1=Ok.
BE92	Kopf positionieren	e:Laufwerk d:Spur	cy:0=Fehler 1=Ok.
BE95	Laufwerkstatus feststellen	a:Laufwerk	a:Status cy:0=Fehler 1=Ok.
BE98	Wiederholungswert bei Fehler	a:Count	----
BE9B	Externer Zugang ins BIOS	----	----
BE9E	Schneider: Fast/Slow-Modus Vortex: Motor abschalten	a:0=Slow 255=Fast	----
BEA1	Schneider: SIO initialisieren Vortex: nicht genutzt	----	----
BEA4	Initial Console Buffer (Übergabe, ob durch Eingabe-abbruch möglich oder nicht)	a:0=Abbruch 255=kein hl:Zeichenkette	----

Die Kommunikationsschnittstelle der CPC-Computer

Übergabeparameter

Funktion	Erhält vom BIOS	geben ans BIOS
SIO-Eingabestatus	----	a:0=kein Zeichen da 255=Ok.
SIO-Zeichen holen	----	a:Zeichencode
SIO-Ausgabestatus	----	a:0=SIO nicht bereit 255=SIO bereit
SIO-Zeichen senden	c:Zeichencode	----

Lage der Einsprünge

Adresse	Bedeutung im Schneider-BIOS	Bedeutung im Vortex-BIOS
BEA7h	Eingabestatus SIO 1	Eingabestatus SIO 1
BEAAh	Zeichen holen SIO 1	Eingabestatus SIO 2
BEADh	Ausgabestatus SIO 1	Ausgabestatus SIO 1
BEB0h	Zeichen senden SIO 1	Ausgabestatus SIO 2
BEB3h	Eingabestatus SIO 2	Zeichen holen SIO 1
BEB6h	Zeichen holen SIO 2	Zeichen holen SIO 2
BEB9h	Ausgabestatus SIO 2	Zeichen senden SIO 1
BEBCh	Zeichen senden SIO 2	Zeichen senden SIO 2

Stichwortverzeichnis

Bücher zu Schneider CPCs

J. Hückstädt

CP/M 2.2 Anwenderhandbuch CPC 464/664/6128

Dezember 1985, 212 Seiten

Wenn Sie glücklicher Besitzer eines Schneider-Computers sind und mehr wissen wollen über das leistungsstarke Betriebssystem CP/M 2.2, dann ist dieses Buch genau das richtige für Sie! Es behandelt CP/M 2.2 nicht nur in seiner allgemeinen Form, wie Sie für sämtliche CP/M-Computer gültig ist, sondern bezieht auch die Hardware der CPC-Computer mit ein.

Best.-Nr. MT 859
ISBN 3-89090-204-9
DM 46,-/sFr. 42,30/öS 358,80

C. Straush

Schneider CPC Grafik-Programmierung

1. Quartal 1986, 225 Seiten

Dieses Buch wendet sich an die Schneider CPC-Besitzer, die alles über die Grafikfähigkeiten ihres Computers wissen wollen. Es bietet einen umfassenden Überblick über die verschiedenen Anwendungsbereiche der Grafikprogrammierung: zwei- und dreidimensionale Diagrammdarstellungen, Definition und Bewegung von Sprites, Entwurf von Titelgrafiken, Einsatz der Grafik bei der Unterstützung anderer Programme.

- Besonders interessant: ein Sprite-Generator, ein Malprogramm für hochauflösende Grafik, ein Programm zur Erstellung von Titelgrafiken sowie ein universelles Darstellungsprogramm.

Best.-Nr. MT 90182
ISBN 3-89090-182-4
DM 46,-/sFr. 42,30/öS 358,80

J. Hückstädt

Der Schneider CPC 6128

September 1985, 273 Seiten

Dieses Buch ist für jeden CPC 6128-Besitzer eine wertvolle Hilfe, die vielfachen Möglichkeiten dieses bisher einmaligen Computers kennenzulernen und anzuwenden. Der Computerneuling wird Schritt für Schritt in den Umgang mit dem Computer und die BASIC-Programmierung eingeführt, bis er alle notwendigen Kenntnisse besitzt, die mancher Profi bereits mitbringt. Aber an dieser Stelle wird das Programmieren mit dem CPC 6128 erst interessant, nämlich dann, wenn es darum geht, eine eigene Dateiverwaltung aufzubauen oder Grafik und Sound zu programmieren. Weiterhin erfahren Sie alles über CP/M Plus auf dem CPC 6128.

Best.-Nr. MT 849
ISBN 3-89090-192-1
DM 46,-/sFr. 42,30/öS 358,80

Markt & Technik-Fachbücher erhalten Sie bei Ihrem Buchhändler

Markt&Technik
BUCHVERLAG

Hans-Pinsel-Str. 2, 8013 Haar bei München

Spitzen-Software für Schneider-Computer

WordStar 3.0 mit MailMerge

Der Bestseller unter den Textverarbeitungsprogrammen für PCs bietet Ihnen bildschirmorientierte Formatierung, deutschen Zeichensatz und DIN-Tastatur sowie integrierte Hilfstexte. Mit MailMerge können Sie Serienbriefe mit persönlicher Anrede an eine beliebige Anzahl von Adressen schreiben und auch die Adreßaufkleber drucken.

WordStar/MailMerge für den Schneider CPC 464*, CPC 664*
Bestell-Nr. MS 101 (3"-Diskette)
Bestell-Nr. MS 102 (5¼"-Diskette im VORTEX-Format)
WordStar/MailMerge für den Schneider CPC 6128
Bestell-Nr. MS 104 (3"-Diskette)
WordStar/MailMerge für den Schneider Joyce PCW 8256
Best.-Nr. MS 105 (3"-Diskette)

Hardware-Anforderungen: Schneider CPC 464*, CPC 664*, CPC 6128 oder Joyce, beliebiger Drucker mit Centronics-Schnittstelle
* Der Standard-Speicherplatz beim CPC 464/664 erlaubt ohne Speichererweiterung Blockverschiebe-Operationen nur bedingt und Simultan-Drucken gar nicht.

Dieses Programm kostet
DM 199,– inkl. MwSt. Unverbindliche Preisempfehlung

Und dazu die weiterführende Literatur:

Mit diesem Buch haben Sie eine wertvolle Ergänzung zum WordStar-Handbuch: Anhand vieler Beispiele steigen Sie mühelos in die Praxis der Textverarbeitung mit **Wordstar** ein. Angefangen beim einfachen Brief bis hin zur umfangreichen Manuskripterstellung zeigt Ihnen dieses Buch auch, wie Sie mit Hilfe von MailMerge Serienbriefe an eine beliebige Anzahl von Adressen mit persönlicher Anrede senden können.

Best.-Nr. MT 779
ISBN 3-89090-180-8

DM 49,–

Erhältlich bei Ihrem Buchhändler.

Sie erhalten jedes **WordStar-, dBASE II-** und **MULTIPLAN-Programm** für Ihren Schneider-Computer fertig angepaßt (Bildschirmsteuerung und Druckerinstallation). **Jeweils Originalprodukte!** Jedes Programmpaket enthält außerdem ein ausführliches Handbuch mit kompakter Befehlsübersicht. Die VORTEX-Speichererweiterung für den Schneider CPC 464 erhalten Sie direkt bei der Firma VORTEX oder bei Ihrem Computerhändler.

Diese Markt&Technik-Softwareprodukte erhalten Sie in den Computer-Abteilungen der Kaufhäuser oder bei Ihrem Computerhändler.

Markt&Technik
Schneider CPC-Software

Hans-Pinsel-Straße 2, 8013 Haar bei München

Spitzen-Software für Schneider-Computer

dBASE II, Version 2.41

dBASE II, das meistverkaufte Programm unter den Datenbanksystemen, eröffnet Ihnen optimale Möglichkeiten der Daten- u. Dateihandhabung. Einfach u. schnell können Datenstrukturen definiert, benutzt und geändert werden. Der Datenzugriff erfolgt sequentiell oder nach frei wählbaren Kriterien, die integrierte Kommandosprache ermöglicht den Aufbau kompletter Anwendungen wie Finanzbuchhaltung, Lagerverwaltung, Betriebsabrechnung usw.

dBASE II für den Schneider CPC 464*, CPC 664*
Bestell-Nr. MS 301 (3"-Diskette)
Bestell-Nr. MS 302 (5¼"-Diskette im VORTEX-Format)
dBASE II für den Schneider CPC 6128
Bestell-Nr. MS 304 (3"-Diskette)
dBASE II für den Schneider Joyce PCW 8256
Best.-Nr. MS 305 (3"-Diskette)

Hardware-Anforderungen: Schneider CPC 464*, CPC 664*, CPC 6128 oder Joyce, beliebiger Drucker mit Centronics-Schnittstelle
* dBASE II für den Schneider CPC 464/664 ist lauffähig mit der VORTEX-Speichererweiterung auf 128 KByte. Diese erhalten Sie direkt bei der Firma VORTEX oder bei Ihrem Computerhändler.

Dieses Programm kostet
DM 199,– Inkl. MwSt. Unverbindliche Preisempfehlung

Markt&Technik
Schneider CPC-Software
dBASE II™
ASHTON-TATE
für den Schneider CPC 6128
3" Schneider-Format

Und dazu die weiterführende Literatur:

Zu einem Weltbestseller unter den Datenbanksystemen gehört auch ein klassisches Einführungs- und Nachschlagewerk! Dieses Buch von dem deutschen Erfolgsautor Dr. Peter Albrecht begleitet Sie mit nützlichen Hinweisen, die nur von einem Profi stammen können, bei Ihrer täglichen Arbeit mit **dBASE II.** Schon nach Beherrschung weniger Befehle ist der Einsteiger in der Lage, Dateien zu erstellen, mit Informationen zu laden und auszuwerten.

Best.-Nr. MT 779
ISBN 3-89090-180-8
DM 49,–

Erhältlich bei Ihrem Buchhändler.

Sie erhalten jedes **WordStar-, dBASE II-** und **MULTIPLAN-Programm** für Ihren Schneider-Computer fertig angepaßt (Bildschirmsteuerung und Druckerinstallation). **Jeweils Originalprodukte!** Jedes Programmpaket enthält außerdem ein ausführliches Handbuch mit kompakter Befehlsübersicht. Die VORTEX-Speichererweiterung für den Schneider CPC 464 erhalten Sie direkt bei der Firma VORTEX oder bei Ihrem Computerhändler.

Diese Markt&Technik-Softwareprodukte erhalten Sie in den Computer-Abteilungen der Kaufhäuser oder bei Ihrem Computerhändler.

Markt&Technik
Schneider CPC-Software

Hans-Pinsel-Straße 2, 8013 Haar bei München

JETZT AUF SCHNEIDER-COMPUTERN:

Turbo Lader

DIE PROGRAMM-BIBLIOTHEK FÜR TURBO PASCAL®

TURBO-Lader-Grundpaket

Das TURBO-Lader-Grundmodul ist eine umfangreiche Programm-Bibliothek für den TURBO-Pascal-Programmierer. Sie umfaßt zahlreiche ausführlich dokumentierte Prozeduren und Funktionen, die der Profi zur schnellen Lösung seiner Programmieraufgaben verwenden kann.

Das TURBO-Lader-Grundpaket erfordert den TURBO-Pascal-Compiler. Es ist lieferbar auf 3"- und 5 1/4"-Disketten und lauffähig auf dem Schneider CPC 464, CPC 664, CPC 6128 und Joyce.

3"-Disk. Best.-Nr. MS 413
5¼"-Disk. Best.-Nr. MS 415

DM 138,-*

TURBO-Lader Business

TURBO-Lader Business umfaßt einen komfortablen Bildschirm-Maskengenerator und eine professionelle Dateiverwaltung. Der Maskengenerator gibt dem Pascal-Programmierer ein Werkzeug zur einfachen Bearbeitung von Bildschirm-Masken in die Hand.

TURBO-Lader Business erfordert den TURBO-Pascal-Compiler und das TURBO-Lader-Grundpaket. Es ist lieferbar auf 3"- und 5 1/4"-Disketten und lauffähig auf dem Schneider CPC 464, CPC 664, CPC 6128 und Joyce.

3"-Disk. Best.-Nr. MS 423
5¼"-Disk. Best.-Nr. MS 425

DM 148,-*

TURBO-Lader Science

TURBO-Lader Science ist eine Sammlung technisch/wissenschaftlicher Funktionen und professioneller statistischer Verfahren für die Bereiche Medizin, Betriebs- und Volkswirtschaft, Technik und Naturwissenschaften.

TURBO-Lader Science erfordert den TURBO-Pascal-Compiler und das TURBO-Lader-Grundpaket. Es ist lieferbar auf 3"- und 5 1/4"-Disketten und lauffähig auf dem Schneider CPC 464, CPC 664, CPC 6128 und Joyce.

3"-Disk. Best.-Nr. MS 433
5¼"-Disk. Best.-Nr. MS 435

DM 189,-*

Übrigens können Sie auch folgende Turbo-Pascal-Produkte für Schneider CPC und Joyce bei Markt & Technik beziehen:
Turbo Pascal 3.0, Turbo Pascal 3.0 mit Grafikunterstützung,
Turbo Tutor (Deutsch), Turbo Tutor (Englisch), Turbo Graphik Toolbox, Turbo Toolbox.

*inklusive MwSt., unverbindliche Preisempfehlung

TURBO Pascal® ist ein Warenzeichen der Borland Inc., USA. TURBO-Lader, TURBO-Lader Business und TURBO-Lader Science sind Warenzeichen der Fa. Lauer & Wallnitz.

Markt&Technik
Schneider CPC-Software

Hans-Pinsel-Straße 2, 8013 Haar bei München

Markt & Technik-Softwareprodukte erhalten Sie in den Computer-Abteilungen der Kaufhäuser und im Computershop.

Programmentwicklung unter CP/M 2.2 auf dem CPC 464/664

Programme, die professionell aussehen sollen, kann man nur unter einem »richtigen« Betriebssystem entwickeln. Das Standardbetriebssystem in der 8-Bit-Welt, CP/M 2.2, wird den Floppy-Disks zu den Computern CPC 464/664/6128 gleich mitgeliefert.

Dieses Buch vermittelt alle Informationen, die zum selbständigen Entwickeln von CP/M 2.2-Programmen nötig sind. Besprochen werden sowohl die grundlegende Funktionsweise des CP/M-Betriebssystems als auch alle dem Anwender schon zur Verfügung stehenden Systemroutinen, die viel Arbeit ersparen. Zwei Kapitel beschäftigen sich ausschließlich mit den zusätzlichen Möglichkeiten, die nur die Computer CPC 464/664/6128 bieten (BDOS-Routinen).

Wenn Sie alle gelernten Kenntnisse in der Praxis anwenden wollen, sind Kenntnisse der 8080- oder Z80-Assemblersprache vorteilhaft. Für einfachere Experimente genügen aber auch Kenntnisse einer höheren Programmiersprache wie Turbo-Pascal.

Die enthaltenen Anwendungsprogramme sind auch in Form eines BASIC-Laders abgedruckt, so daß sie auch Leser, die die Maschinensprache nicht beherrschen, abtippen können. Durch aufmerksames Lesen können aber selbst diese durchaus die Funktionsweise der Programme verstehen.

Aus dem Inhalt:

- Beschreibung der 8080-Assemblersprache anhand eines Vergleichs zum Z80-Assembler
- Hilfsmittel zur Programmerstellung
- Konzept des CP/M 2.2-Betriebssystems
- Anwendung der Systemroutinen mit Beispielen in Assembler und Turbo-Pascal
- Arbeiten mit Dateien
- Abweichungen des CPC 464/664/6128-CP/M vom Standard-CP/M
- Zuschneiden des Betriebssystems auf eigene Bedürfnisse
- Nützliche CP/M-Programme zum Abtippen (Utilities wie deutscher Zeichensatz, Drucker-Spooler etc.)

Alle im Buch vorgestellten Programme sind unter dem Betriebssystem CP/M 2.2 auf dem Schneider CPC 6128, in der CPC 664-Version, lauffähig. Beachten Sie: Unter dem Betriebssystem CP/M-Plus – ebenfalls für den CPC 6128 verfügbar – können die Programme nicht verwendet werden!

ISB N 3-89090-209-X

4 001057 902091

Markt&Technik
Verlag Aktiengesellschaft

DM 52,-
sFr. 47,80
öS 405,60