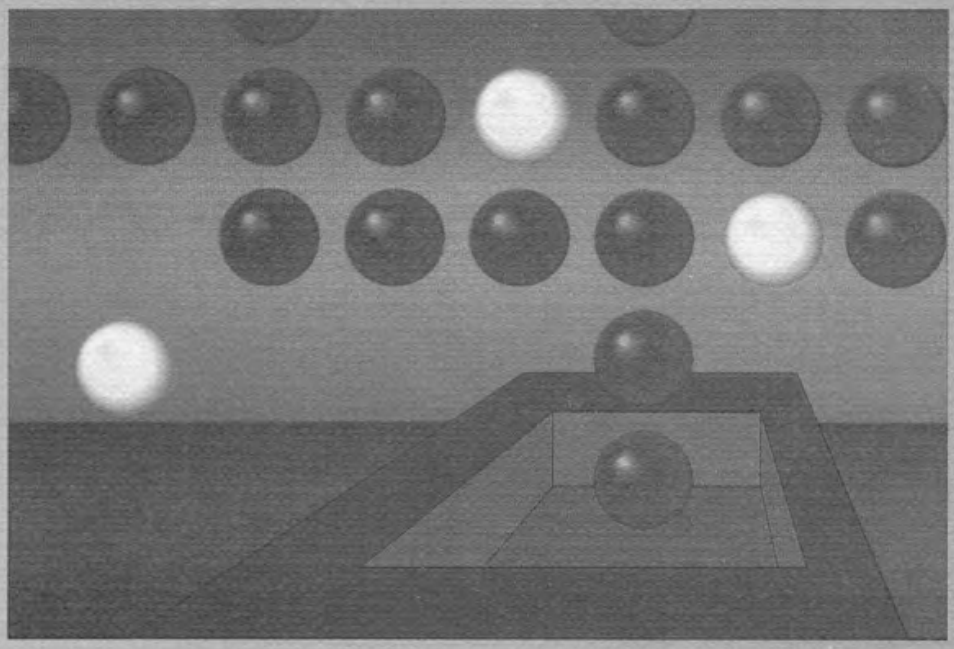
Franzis
Computer-Praxis

Plate
Anwenderhandbuch CP/M-68K

Einstieg in das Betriebssystem für die 68000-Familie



Eigenschaften und Dienste des CP/M-68K
Kommandostruktur
Dateizugriff
Eingebaute Kommandos
Transiente Kommandos
Texteditor ED
CP/M-68K Assembler und Hilfskommandos
Dynamic Debugging Tool DDT-68K
CP/M-68K Systemschnittstelle
Konventionen für den Systemaufruf
Geräte und IOBYTE
Das BIOS
BIOS-Aufrufe
Der C-Compiler
Parametertafeln für die Diskettenlaufwerke
BIOS-Listing
Fehlerbehandlung

Franzis'

Vorwort

Dieses Handbuch soll Ihnen die ersten Schritte beim Betrieb des Diskettenbetriebssystems CP/M 68 K erleichtern. Es kann das mit den Systemdisketten gelieferte Handbuch nicht ersetzen, insbesondere dann, wenn Sie am BIOS Änderungen vornehmen wollen. Für den "Nur-Benutzer" des Systems bietet es die wichtigsten Informationen und hilft beim Einstieg in CP/M 68 K.

Neben der kurz gefassten Beschreibung der Kommandos und ihrer Aufrufstruktur enthält es alle wichtigen Informationen über BIOS und BDOS und ermöglicht es Ihnen, Programme unter CP/M 68 K zu schreiben.

Zur Programmierung des Prozessors 68000 wurde in diesem Buch nichts gesagt, denn dafür gibt es spezielle, sehr umfangreiche Bücher, etwa "68000 Assembly Language Programming" von Kane, Hawkins und Leventhal, Verlag Osborne/McGraw-Hill oder "Die M68000-Familie", Band 1 und Band 2 von W.Hilf und A.Nausch, Verlag Te-Wi, MÜnchen.

Für das Erlernen der Sprache C ist das Buch "Programmieren in C" von Kernighan und Ritchie, Verlag Hanser, München zu empfehlen.

Da das CP/M 68 K stark auf dem CP/M 2.2 für den Prozessor Z-80 aufbaut (so sind zum Beispiel die Dateien und Disketten völlig identisch organisiert), ist als Hintergrundlektüre auch das Buch "Betriebssystem CP/M" von Jürgen Plate, Franzis-Verlag oder ein anderes Buch über CP/M 2.2 zu empfehlen.

Dipl. Inform. J. Plate

Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen und Verfahren werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden *).
Alle Schaltungen und technische Angaben in diesem Buch wurden vom Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. Der Verlag und der Autor sehen sich deshalb gezwungen, darauf hinzuweisen, daß sie weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen können. Für die Mitteilung eventueller Fehler sind Autor und Verlag jederzeit dankbar.

*) Bei gewerblicher Nutzung ist vorher die Genemigung des möglichen Lizenzinhabers einzuholen.

Inhalt

1 Eigenschaften und Dienste des CP/M 68K

1.1 Einführung

CP/M ist ein Betriebssystem für Mikrocomputersysteme, die Floppy-Disk oder Winchesterplatten als Massenspeicher verwenden. Durch ein umfassendes Dateiverwaltungskonzept ermöglicht CP/M einen schnellen Zugriff auf Programme und Daten. Das Dateisystem unterstützt benannte Dateien und erlaubt eine dynamische Zuweisung des Speicherplatzes auf der Diskette sowohl für sequentiellen als auch für wahlfreien Zugriff. Mit dem Dateisystem kann eine große Zahl von Programmen in Quell- und Objektform auf einer Diskette gespeichert werden.
CP/M 68 K besteht aus vier logischen Einheiten:

BIOS	E/A-Kern, hardwareabhängig (Basic I/O System)
BDOS	Diskettentreiber-Routinen (Basic Disc Operating System)
CCP	Kommandointerpreter (Console Command Processor)
TPA	Programmbereich für System- und Anwenderprogramme (Transient Program Area)

Das BIOS enthält die Primitivoperationen für den Diskettenzugriff und für die Schnittstelle zu den Standardperipheriegeräten (Konsole, d.h. Tastatur und Bildschirm oder Teletype, Lochstreifenleser, Lochstreifenstanzer, benutzerdefinierte Geräte). Mit geeigneten Änderungen des BIOS kann der Benutzer das CP/M an jede Hardwareumgebung anpassen.
Durch die Ansteuerung einer oder mehrerer Platten- (Disketten-) Laufwerke mit jeweils eigenen Dateiinhaltsverzeichnissen sorgt das BDOS für die Dateiverwaltung. Das BDOS verwendet eigene Zuordnungsstrategien bei der Diskettenverwaltung, so daß eine dynamische Dateikonstruktion möglich ist und die Bewegung des Schreib/Lesekopfes über die Diskette bei einem Zugriff minimiert wird. Das BDOS hat verschiedene Routinen, unter anderem die folgenden Primitivoperationen, die auch von Benutzerprogrammen aus zugreifbar sind:

SEARCH	Suche eine Datei mit einem bestimmten Namen
OPEN	Eröffne Datei für die folgenden Zugriffe
CLOSE	Schließe Datei nach der Bearbeitung
RENAME	Ändere den Namen einer bestimmten Datei
READ	Lies einen Datensatz aus einer bestimmten Datei

WRITE	Schreibe einen Datensatz auf eine bestimmte Datei
SELECT	Aktiviere ein bestimmtes Laufwerk für die weitere Bearbeitung

Der CCP bietet eine symbolische Schnittstelle zwischen Bediener (Konsole) und dem restlichen System. Der CCP liest und bearbeitet die Kommandos der Benutzerkonsole. Unter anderem gibt es Kommandos zum Auflisten eines Dateiinhaltes, zum Drucken des Inhaltes einer Datei und zum Steuern des Ablaufes von Programmen wie Assembler, Editor oder Debugger. Die Aufrufstruktur des CCP ist in Abschnitt 1.2.1 beschrieben.
Der letzte Teil des CP/M ist die TPA (Transient Program Area), was etwa bedeutet: der Bereich der wechselnden Programme. In diesen Bereich werden Programme durch einen Kommandobefehl an den CCP von Diskette geladen. Zum Beispiel sind in der TPA beim Programm-Edieren der Code des Editor-Programms und dessen Datenbereiche gespeichert. Unter CP/M erstellte und getestete Programme werden genauso in die TPA geladen und laufen dort ab.
Ein Teil der Komponenten des CP/M (gegebenenfalls auch alle Teile) können von Benutzerprogrammen überlagert werden (overlay). Sobald ein Programm in die TPA geladen worden ist, können dessen Datenbereiche den CCP, das BDOS und das BIOS überschreiben. Falls der Bereich des BIOS nicht verletzt wird, kann durch das Benutzerprogramm auf einen Kalt- oder Warmstartlader (Bootstrap-Loader) im BIOS zugegriffen werden, der das CP/M wieder von Diskette lädt. Der Benutzer muß dazu nur am Ende des Programmablaufs von seinem Programm in den Warmstartlader springen um das CP/M wieder neu von der Diskette zu holen.
Das CP/M einschließlich des BIOS besteht also aus eigenständigen Modulen. Das BIOS legt die Hardwareumgebung fest, in der das CP/M arbeitet. Das bedeutet, durch Änderung der Peripherietreiber kann das Standardsystem leicht so modifiziert werden, um in einer nicht standardmäßigen Umgebung eines Benutzersystems zu arbeiten.

1.2 Funktionelle Beschreibung

Der Benutzer kommuniziert mit dem CP/M hauptsächlich über den CCP. Dieser liest und interpretiert die Kommandos, die an der Benutzerkonsole eingegeben werden. Im Allgemeinen selektiert der CCP ein oder mehrere Laufwerke, die betriebsbereit sind. Das Standardsystem kann bis zu 16 Laufwerke steuern. Sie werden mit den Buchstaben A ... P benannt. Ein Laufwerk ist angemeldet (logged in), wenn der CCP das Laufwerk selektiert hat. Damit der Benutzer weiß, welches Laufwerk gerade aktiv ist, besteht die Kommandoanforderung aus der Angabe des Laufwerks, gefolgt von ">". Beim Kaltstart wird das Laufwerk A selektiert und der CCP meldet sich mit:

```
CP/M 68 K     Version m.n
```

wobei m.n die CP/M Versions- bzw Revisionsnummer ist.

Der Benutzer wird mit

```
A>
```

aufgefordert, ein Kommando einzugeben. Gleichzeitig wird der Bediener informiert, das Laufwerk A selektiert wurde. Die Kommandos sind auf zwei Ebenen implementiert: die eingebauten Kommandos (built in Commands), die direkter Bestandteil des CCP sind, und die Kommandos, die von Diskette geladen werden müssen (transient Commands).

1.2.1 Kommandostruktur

Die eingebauten Kommandos sind Teil des CCP-Programms, wogegen die "transient commands" von der Diskette in die TPA geladen und dann gestartet werden.
Die eingebauten Kommandos sind:

ERA	Lösche die angegebene(n) Datei(en)	(Erase)
DIR	Liste das Dateiinhaltsverzeichnis auf der Konsole	(Directory)
DIRS	Liste die Systemdateien auf der Konsole	
REN	Benenne die angegebene(n) Datei(en) um	(Rename)
TYPE	Protokolliere eine Datei auf einem E/A-Gerät	
SUBMIT	Übergeben der Kontrolle an eine Kommando-Datei für die Stapelverarbeitung (Batch).	

Die meisten Kommandos beziehen sich auf eine Datei oder eine Gruppe von Dateien. Wie eine Dateiangabe erfolgt, wird in 1.2.2 erklärt.

1.2.2 Dateizugriff

Eine Dateireferenz identifiziert eine oder eine Gruppe von Dateien auf einem bestimmten Laufwerk. Die Dateireferenzen sind also entweder eindeutig (unambigous) oder mehrdeutig (ambigous). Eine eindeutige Dateireferenz bezeichnet genau eine Datei auf der Diskette. Bei einer mehrdeutigen Dateireferenz werden mehrere Dateien ausgewählt. Eine Dateireferenz besteht aus zwei Teilen, dem Dateinamen und dem Dateityp. Obwohl die Angabe eines Dateityps freiwillig (optional) ist, klassifiziert er die Datei, wenn er doch angegeben wird. So charakterisiert der Dateityp "68K" die Assembler-Quelldateien. Der Dateiname unterscheidet dann die einzelnen Quellen. Dateiname und Dateityp werden durch einen Punkt getrennt; sie haben also die folgende Form:

```
Dateiname.Dateityp
```

Der Dateiname besteht aus maximal 8 Zeichen, der Dateityp aus

maximal 3 Zeichen. Wie schon erwähnt kann der Dateityp entfallen, was die Angabe zu:

```
Name
```

reduziert. Diese Referenz ist gleichbedeutend mit "Name. ". In Dateinamen sind Buchstaben, Ziffern und die Sonderzeichen "$", "!", "#", "&", "^", "+", "-" und "§" erlaubt.
Eine mehrdeutige Dateireferenz wird für die Suche einer Datei im Directory und für die Auswahl von Dateien nach einem bestimmten Muster (pattern matching) verwendet. Die Form ist dabei die gleiche wie bei der eindeutigen Dateireferenz, es tritt jedoch das Symbol "?" im Dateinamen oder im Dateityp auf. In einer Dateireferenz eines CP/M-Kommandos stimmt überall dort wo das Zeichen "?" steht, das Zeichen mit jedem anderen möglichen überein, das auf der gleichen Position steht. Beispielsweise stimmt die mehrdeutige Referenz

```
X?Z.6?K
```

überein mit den eindeutigen Namen

```
XYZ.68K
```

und

```
X3Z.68K
```

Die mehrdeutige Referenz

```
*.*
```

ist äquivalent der mehrdeutigen Referenz

```
????????.???
```

Dagegen sind

```
Datei.*        bzw.          *.Typ
```

Abkürzungen für

```
Datei.???      bzw.     ???????.Typ
```

Beispielsweise liefert die Kommandoeingabe

```
A> DIR *.*
```

die Liste aller Dateinamen auf der Diskette in Laufwerk A.
Dagegen bewirkt

```
A> DIR X.Y
```

die Suche nach der Datei X.Y . Das Kommando

```
A> DIR X?Y.6?K
```

bewirkt die Suche nach allen Dateinamen, die mit dem Muster der mehrdeutigen Referenz übereinstimmen. Mit

```
A> DIR *.S
```

erhält man eine Liste aller Assembler-Quelldateien.
Die folgenden Namen sind Beispiele für eindeutige Dateireferenzen:

```
X          XYZ        GAMMA
X.Y        XYZ.68K    GAMMA.1
```

Zusätzlich zu Dateiname und Dateityp kann der Benutzer noch den Namen eines Laufwerks angeben. In diesem Fall wird der Laufwerksname (Buchstaben von A bis P) gefolgt von einem Doppelpunkt vor den Dateinamen gesetzt. Bevor auf die Datei zugegriffen wird, wird das entsprechende Laufwerk selektiert. Es folgen einige Beispiele für Dateireferenzen mit Laufwerksangaben:

```
A:X.Y      B:XYZ      C:GAMMA
P:XYZ.68K  B:X.6?K    C:*.S
```

Bei allen Datei-, Typ- und Laufwerksnamen werden die Kleinbuchstaben zuerst in Großbuchstaben übersetzt, bevor sie vom CCP weiterverarbeitet werden. Die Namen "b:xyz.68k" und "B:XYZ.68K" sind also vollkommen gleichwertig.

1.3 Laufwerk umschalten

Ein Benutzer kann von dem augenblicklich angemeldeten Laufwerk auf ein anderes umschalten, indem er den Namen (A ... P) des gewünschten Laufwerks, gefolgt von einem Doppelpunkt eingibt, wenn der CCP ein Kommando anfordert. Es folgt ein Beispiel für eine Kommandosequenz nach der Begrüßungsmeldung des CP/M Betriebssystems und dem automatischen Selektieren des Laufwerks A:

```
A> DIR      Auflisten aller Dateinamen der Dateien auf der
            Diskette in Laufwerk A.

A: SAMPLE   68K  SAMPLE   PRN

A> B:       Umschalten auf Laufwerk B.
B> DIR *.68K

            Auflisten aller 68K-Dateien auf der Diskette
            in Laufwerk B.

B: DUMP     68K  FILES    68K

B> A:       Umschalten auf Laufwerk A.
```

1.4 Eingebaute Kommandos

Die im letzten Abschnitt besprochenen Datei- und Gerätereferenzen werden nun verwendet um die Struktur der eingebauten Kommandos (built in commands) zu beschreiben. Man beachte dazu die folgenden beiden Abkürzungen, die in den weiteren Beschreibungen häufig auftreten werden:

ufn	eindeutige Dateireferenz (unambigous file name)
afn	mehrdeutige Dateireferenz (ambigous file name)

Es wird nochmals darauf aufmerksam gemacht, daß der CCP Kleinbuchstaben immer in Großbuchstaben umwandelt, bevor das Kommando bearbeitet wird. Kleinbuchstaben in Kommandonamen werden also wie Großbuchstaben behandelt.

1.4.1 ERA afn

Das Kommando ERA (erase) löscht Dateien auf der augenblicklich angemeldeten Diskette (d.h. der Diskette, die in dem Laufwerk liegt, dessen Name bei der Kommandoanforderung ausgegeben wurde). Alle Dateien, die der mehrdeutigen Referenz afn entsprechen, werden gelöscht. Die folgenden Beispiele zeigen die Verwendung des Kommandos ERA:

ERA X.Y	Die Datei X.Y auf der Diskette im aktiven Laufwerk wird aus dem Direktory gelöscht und der belegte Platz freigegeben.
ERA X.*	Alle Dateien mit dem Namen X und beliebigem Typ werden auf der aktuellen Diskette gelöscht (s. auch ERA *.*).
ERA *.S	Alle Assembler-Quelldateien werden gelöscht.
ERA X?Y.6?K	Alle Dateien, auf die das Muster passt, werden gelöscht (z.B.: XZY.68K).
ERA *.*	Alle Dateien auf der angemeldeten Diskette werden gelöscht. Werden Wildcards angegeben, fordert der CCP nochmals eine Bestätigung an, indem er auf der Konsole ausgibt: CONFIRM (Y/N)? Will man die Dateien löschen, muß man mit Y (Yes = Ja) antworten, sonst mit N (No = Nein).
ERA B:*.PRN	Alle Dateien auf der Diskette in Laufwerk B, die der Referenz ????????.PRN entsprechen werden unabhängig vom gerade aktiven Laufwerk gelöscht.

1.4.2 DIR afn

Das Kommando DIR (Directory) hat zur Folge, daß alle Dateinamen, die der mehrdeutigen Referenz afn genügen, auf der Konsole gelistet werden. Als Sonderfall bewirkt das Kommando

```
DIR
```

das Auflisten aller Dateinamen auf der angemeldeten Diskette (DIR hat also dieselbe Bedeutung wie DIR *.*). Beispiele erlaubter DIR-Kommandos sind:

```
DIR X.Y
DIR X?Y.6?K
DIR *.68K
```

Wie bei den anderen CCP-Kommandos kann ein Laufwerksname vor der Dateiangabe stehen. In den folgenden Beispielen wird zuerst das gewünschte Laufwerk selektiert und danach erst nach den Dateien gesucht:

```
DIR B:
DIR B:X.Y
DIR B:*.68K
```

Falls keine einzige passende Datei gefunden wird, erscheint auf der Konsole die Meldung

```
NO FILE
```

DIRS listet alle Dateien mit Systemattribut (siehe STAT).

1.4.3 REN ufn1=ufn2

Das Kommando REN (Rename) gibt dem Benutzer die Möglichkeit, Dateien auf einer Diskette umzubenennen. Die Datei mit dem Namen ufn2 wird in ufn1 umbenannt. Es wird dabei angenommen, daß die Datei, die umbenannt werden soll (ufn2), auf der Diskette im aktuellen Laufwerk liegt. Anstelle des Gleichheitszeichen kann auch der nach links zeigende Pfeil verwendet werden, falls dieses Zeichen an der Konsole verfügbar ist. Beispiele:

```
REN X.Y=Q.R          Die Datei Q.R heißt nun X.Y .

REN XYZ.68K=XYZ.XXX  Die Datei XYZ.XXX wird zu XYZ.68K  umbe-
                     nannt.
```

Es ist möglich, daß vor jeder der Angaben ufn1 und ufn2 der Name eines Laufwerks steht. Falls nur vor ufn1 ein Laufwerkname steht, wird angenommen, daß die umzubenennende Datei ufn2 auf diesem Laufwerk zu finden ist. Ebenso wird angenommen daß ufn 1 auf dem gleichen Laufwerk steht, wenn nur bei ufn2 ein Laufwerkname

steht. Stehen die Laufwerkangaben bei beiden Namen, müssen sie gleich sein. Die folgenden Beispiele illustrieren den Sachverhalt:

```
REN A:X.68K=Y.68K    Auf Laufwerk A wird Y.68K in X.68K umbenannt.

REN Z.BAS=B:X.BAS    Auf Laufwerk B wird X.BAS in Z.BAS umbenannt.

REN B:A.68K=B:A.BAK  Auf Laufwerk B wird A.BAK in A.68K umbenannt.
```

Falls die Datei mit dem Namen ufn1 bereits vorhanden ist, gibt das REN-Kommando die Fehlermeldung

```
FILE EXISTS
```

(Datei existiert schon) aus und es findet keine Umbenennung statt. Gibt es keine Datei namens ufn2, erfolgt die Fehlermeldung

```
NO FILE
```

auf der Konsole.

1.4.4 TYPE ufn

Das Kommando TYPE listet den Inhalt der ASCII-Datei (eine Datei mit lesbarem Text) ufn auf der Konsole. Zulässige TYPE-Kommandos sind:

```
TYPE X.Y

TYPE X.PRN

TYPE XXX
```

Das Kommando TYPE expandiert Tabulatoren (Zeichen Control/I). Dabei wird alle 8 Spalten ein Tabulatorstop angenommen. Auch hier ist wieder ein Laufwerkname in der Dateiangabe möglich:

```
TYPE B:X.PRN         Die Datei X.PRN auf der Diskette in
                     Laufwerk B wird auf der Konsole gelistet
```

1.4.5 USER n

Das USER-Kommando gibt die Möglichkeit, getrennte Dateigruppen in einem Directory unterzubringen. Es hat die Gestalt

```
USER n
```

wobei n eine ganze Zahl zwischen 0 und 15 ist. Bei einem Kalt-

start wird dem Bediener automatisch der Benutzerbereich 0 zugewiesen. Der Bediener kann das USER-Kommando verwenden um in einen anderen logischen Bereich des Inhaltsverzeichnisses zu gelangen. Laufwerke, die unter einem Benutzerbereich selektiert wurden, bleiben auch beim Umschalten des Benutzerbereichs aktiviert. Die Benutzernummer ist ein Präfix, der den Zugriff auf eine bestimmte Gruppe von Einträgen im Directory erlaubt, Dateien anderer Benutzernummern sind nicht zugreifbar.
Der Benutzerbereich bleibt erhalten, bis er durch das entsprechende USER-Kommando umgeschaltet wird. Bei einem Kaltstart wird immer USER 0 voreingestellt. Wurde ein USER-Bereich ungleich 0 eingestellt, meldet sich der CCP mit dem erweiterten Prompt aus User-Nummer und Laufwerk:

```
4A>USER 3

3A>B:

3B>USER 0

B>
```

1.5 Edieren der Kommandozeile und Ausgabesteuerung

Der CCP erlaubt es, die Kommandozeile im begrenzten Umfang zu edieren, solange sie noch nicht mit der Return-Taste abgeschlossen wurde (Im Folgenden ist mit z.B. "Control/C" gemeint, daß die Control-Taste niedergedrückt gehalten wird, während die Taste hinter dem Schrägstrich ("C") gedrückt wird)

Control/C	Warmstart, falls es am Zeilenanfang eingegeben wird.
Control/E	Physikalisches Zeilenende. Es wird ein Return ausgegeben, aber die Zeile nicht abgeschlossen.
Control/H	Es wird ein Zeichen zurückgegangen. (Backspace)
Control/J	Abschicken der Zeile (=> Linefeed).
Control/M	Abschicken der Zeile (=> Return).
Control/R	Die letzte Zeile wird noch einmal bereinigt ausgegeben.
Control/U	Die gesamte Eingabezeile wird gelöscht.
Control/X	Die gesamte Eingabezeile wird gelöscht.
Control/Z	Ende der Konsoleingabe (wird von PIP und

ED verwendet).

RUBOUT/DEL — Löschen des zuletzt eingegebenen Zeichens.

Die Funktionen Control/S und Control/P steuern die Ausgabe:

Control/P — Die folgenden Ausgaben werden auf das aktuelle Ausgabegerät geleitet. Bis zum nächsten Control/P werden alle Ausgaben auf der Konsole und auf dem aktuellen Ausgabegerät (z.B. Drucker) ausgegeben.

Control/S — Die Ausgabe auf der Konsole wird angehalten. Der Programmablauf und die Ausgabe wird fortgesetzt, sobald eine Taste gedrückt wird (z.B. wieder Control/S). So kann die Ausgabe zum Ansehen angehalten werden.

Die CCP-Kommandos dürfen bis zu 255 Zeichen lang sein. Eine Kommandozeile wird erst dann bearbeitet, wenn sie mit Return oder Linefeed abgeschlossen wurde.

In einer Kommandozeile können mehrere Kommandos eingegeben werden, CP/M 68 K kann also eine ganze Kommandoliste auch ohne das SUBMIT-Kommando bearbeiten. Die einzelnen Kommandos werden dabei durch Ausrufezeichen voneinander getrennt. Zum Beispiel:

```
DIRS!STAT A:*.*!PIP B:=A:*.68K!STAT B:*.*!DIR B:
```

1.6 Transiente Kommandos

Die sogenannten transienten Kommandos sind Programme, die von der aktuellen Diskette in die TPA geladen und dann ausgeführt werden. Es folgt nun eine Liste der transienten Kommandos, die mit dem CP/M geliefert werden. Weitere Funktionen und Dienste lassen sich vom Benutzer sehr einfach hinzufügen.

STAT — Gibt die Anzahl der freien Bytes auf einer Diskette an und liefert statistische Informationen über die Dateien und Geräte. Weiter kann die Gerätezuordnung geändert werden.

DDT — Laden und Starten des CP/M-Debuggers.

PIP — Laden und Starten eines Programms zum Dateitransfer auf Diskette und zum Verkehr mit den Peripheriegeräten.

ED — Laden und Starten des CP/M-Texteditors.

COPY Kopieren der Diskette.

Die transienten Kommandos werden genauso spezifiziert, wie die eingebauten Kommandos. Zusätzliche Kommandos können vom Benutzer definiert werden. Vor dem Namen eines transienten Kommandos kann ein Laufwerkname stehen. Das transiente Kommando wird dann von der Diskette in dem angegebenen Laufwerk geladen und gestartet. Das Kommando

```
B:STAT
```

bewirkt, daß das Laufwerk B kurzzeitig angemeldet wird, um den Code des Programms STAT zu laden. Danach wird das vorher aktive Laufwerk wieder angemeldet und nun das Programm ausgeführt. Die transienten Kommandos werden nun im Detail beschrieben.

1.6.1 STAT

Das Kommando STAT liefert allgemeine Informationen über Laufwerkseigenschaften, Dateigröße und Gerätezuweisungen. Es wird durch die zwei folgenden Aufrufformen aktiviert:

```
STAT

STAT "Befehlszeile"
```

Mit der "Befehlszeile" können die augenblicklichen Gerätezuweisungen angesehen und geändert werden. Die verschiedenen zulässigen Befehlszeilen werden im Folgenden aufgeführt und erläutert:

STAT
Falls der Benutzer eine leere Befehlszeile eingibt, berechnet das STAT-Kommando den freien Speicherplatz auf den vorher aktivierten Laufwerken. Es liefert für jedes aktive Laufwerk eine der beiden Meldungen:

```
    d: R/W, SPACE: nnn K
oder
    d: R/O, SPACE: nnn K
```

R/W bedeutet, daß die Diskette gelesen und beschrieben werden kann (Read and Write), wogegen R/O anzeigt, daß die Diskette schreibgeschützt ist (Read Only). Eine Diskette bzw ein Laufwerk kann auf R/O gesetzt werden, was weiter unten beschrieben wird. Bei einem Diskettenwechsel wird das Laufwerk aus Sicherheitsgründen automatisch auf R/O gesetzt (dieser Zusand wird durch einen Warmstart aufgehoben). Der noch verfügbare Speicherplatz auf der Diskette wird als nnn Kilobyte angegeben.

STAT d:
Wenn ein Laufwerkname d: angegeben wird, wird das entsprechende Laufwerk selektiert und dessen freier Speicherplatz ermittelt. Es kann also das Kommando "STAT B:" abgeschickt werden, auch wenn

Laufwerk A angemeldet ist. Die resultierend Meldung ist dann:

```
FREE SPACE ON B: nnn K
```

STAT afn
Es kann in der Befehlszeile eine Datei oder eine Dateigruppe spezifiziert werden. Alle Dateien, die mit afn übereinstimmen, werden in alphabetischer Reihenfolge mit ihrem Speicherplatzbedarf und ihren Attributen unter der folgenden Überschrift aufgelistet:

```
RECS  BYTES FCBS ATTRIBUTES

rrrr   bbbK   ee aaa d:dateiname.dateityp
```

rrrr ist die Anzahl der 128-Byte Datensätze, die von der Datei belegt werden. bbb ist die Anzahl der belegten Kilobytes (bbb = rrrr * 128 / 1024). ee ist die Zahl der 16K-Extensionen im Belegungsverzeichnis (ee = bbb / 16). aaa gibt die Zugriffsberechtigung für die Datei an; R/W für Lesen und Schreiben und R/O für schreibgeschützte Dateien. d: ist die Laufwerksbezeichnung (A bis P), dateiname ist der bis zu 8 Zeichen lange Name und dateityp der Typ der Datei. Nach dem Auflisten der einzelnen Dateien wird dann noch der freie Speicherplatz angegeben.

STAT d:afn
Die gleiche Ausgabe wie für STAT afn, es wird jedoch vorher das Laufwerk d: selektiert.

STAT d:=R/O
Das Laufwerk d wird schreibgeschützt. Diese Sperre bleibt bis zum nächsten Kalt- oder Warmstart erhalten. Falls dennoch versucht wird, auf die schreibgeschützte Diskette zu schreiben, erfolgt eine Fehlermeldung.

Das CP/M wartet bis eine Taste gedrückt wird, worauf ein Warmstart ausgeführt wird (gleichzeitig wird der Schreibschutz aufgehoben).

Mit dem STAT-Kommando kann auch die Zuordnung von logischen und physikalischen Geräten gesteuert werden. Es gibt vier logische Peripherigeräte. Jedem logischen Gerät wird eines von mehreren physikalischen Peripheriegeräten zugewiesen. Es kann auch ein einziges physikalisches Gerät mehreren logischen Geräten zugewiesen werden.
Die logischen Geräte sind:

CON	die Systemkonsole (Console) (über sie führt die Kommunikation des CCP mit dem Bediener)
AXI	der Lochstreifenleser (Reader)
AXO	der Lochstreifenstanzer (Punch)

LST das Ausgabegerät für Listings, z.B. Drucker (List)

Die wirklich an das Computersystem angeschlossenen Geräte werden durch Unterprogramme des BIOS-Teils des CP/M gesteuert. Das bedeutet, daß das logische Gerät AXI: auch ein schneller optischer Lochstreifenleser, eine Teletype (Fernschreiber) oder ein Magnetband-Kassettengerät sein kann. Um die Gerätezuweisung zu vereinfachen, wurden für die verschiedenen physikalischen Geräte die folgenden Abkürzungen eingeführt:

TTY: Teletype (ASCII-Fernschreiber)

CRT: Datensichtgerät

BAT: Stapelbetrieb (batch processing)
Die Konsole wird durch die Geräte AXI: (anstelle der Tastatur) und LST: (anstelle z.B. des Bildschirms) ersetzt.

UC1: Benutzerdefiniertes Konsolgerät (user console)

PTR: Lochstreifenleser (paper tape reader)

UR1: Benutzerdefinierter Leser #1

UR2: Benutzerdefinierter Leser #2

PTP: Lochstreifenstanzer (paper tape punch)

UP1: Benutzerdefinierter Stanzer #1

UP2: Benutzerdefinierter Stanzer #2

LPT: Drucker (line printer)

UL1: Benutzerdefinierter Drucker #1

Es wird nochmals darauf hingewiesen, daß die Namen der physikalischen Geräte nicht immer den Geräten entsprechen, die angeschlossen sind. So kann das Gerät PTP: je nach Wunsch des Benutzers auch als ein Magnetband-Kassettengerät implementiert werden. Die Gerätezuordnung und die Unterprogramme zu deren Steuerung sind im BIOS des CP/M festgelegt. In der Standardversion des CP/M entsprechen die Geräte und deren Namen denen des MDS 800 Entwicklungssystems.
Mit dem Kommando

```
STAT VAL:
```

erhält man einen Überblick über die vorhandenen STAT-Befehle. Die Ausgabe nach STAT VAL: ist:

```
Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
Disk Status  : DSK: d:DSK:
```

```
User Status  : USR:
Iobyte Assign:
CON: = TTY: CRT: BAT: UC1:
AXI: = TTY: PTR: UR1: UR2:
AXO: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
```

Sie liefert eine knappe Übersicht aller zulässigen STAT-Befehle. In den letzten vier Zeilen werden auch die möglichen Zuweisungen von logischen zu physikalischen Geräten aufgelistet. Dem logischen Gerät links vom Gleichheitszeichen kann eines der vier physikalischen Geräte auf der rechten Seite zugewiesen werden. Die aktuelle Gerätezuweisung wird durch

```
STAT DEV:
```

auf der Konsole gelistet (DEV steht für Devices (=Geräte)). Es wird zum Beispiel ausgegeben:

```
CON: = CRT:
AXI: = UR1
AXO: = PTP:
LST: = TTY:
```

Dem links vom Gleichheitszeichen stehenden logischen Gerät ist jeweils das rechts stehende physikalische Gerät zugeordnet. Mit dem Kommando:

```
STAT lg1=pg1,lg2=pg2, ... ,lgn=pgn
```

kann die Zuordnung von logischen (lg1 ... lgn) zu physikalischen (pg1 ... pgn) Geräten geändert werden. Beispiele für die Gerätezuweisung:

```
STAT CON:=CRT:

STAT AXO:=TTY:,LST:=LPT:,AXI:=TTY:
```

Mit dem STAT-Kommando können auch Dateiattribute (file indicators) geändert werden. Kommandos der Art

```
STAT d:name.typ $S
```

liefert für die Datei, die durch Laufwerksname "d", Dateiname "name" und Dateityp "typ" bezeichnet wird, die folgende Information (der Laufwerksname kann wie immer entfallen, Dateiname und -typ dürfen auch mehrdeutig sein):

```
SIZE  RECS   BYTES  FCBS ATTRIBUTES

ssss   rrrr   bbbK   ee aaa d:dateiname.dateityp
```

Man erhält durch den Parameter $S also zusätzliche Informationen in der Rubrik "Size". Unter "Size" wird die virtuelle Dateigröße

in Blöcken angegeben. Unter "Recs" steht die Summe der virtuellen Records in der Datei. Bei sequentiellen Dateien stimmen die Werte ssss und rrrr überein. bbb gibt an, wieviele Bytes tatsächlich belegt werden. Die kleinste belegbare Speicherblock-Größe wird bei der Konfiguration festgelegt. Für sequentielle Dateien entspricht die Zahl der belegten Bytes der Anzahl der Blöcke mal der Blockgröße, wobei der letzte Block nicht immer vollständig belegt ist. Dateien mit wahlfreiem Zugriff wird beim Schreiben Speicherplatz zugewiesen. bbb entspricht also den belegten Blöcken; ssss gibt die Nummer des höchsen belegten Blocks an, also das logische Dateiende; rrrr gibt die Zahl der logischen Records in den Extensionen an, wobei hier durchaus "Löcher" vorkommen können, die aber mitgezählt werden.
ee gibt die Zahl der Extensionen im Inhaltsverzeichnis an. Abhängig von der Konfiguration können in einer Extension bis zu 128 KByte (8 logische Extensionen) adressiert werden (in Sonderfällen sogar bis zu 256 KByte).
aaa liefert Angaben über die Zugriffsart , R/O für schreibgeschützte Dateien, R/W für Dateien, auf die geschrieben werden darf. Durch eine Klammer um den Dateinamen wird angegeben, daß das Attribut SYS gesetzt wurde und die Datei beim Kommando DIR nicht aufgelistet wird (z.B. bei PIP.68K unten). Die Attribute lassen sich mit dem Kommando STAT setzen:

```
STAT d:filename.type $RO
STAT d:filename.type $RW
STAT d:filename.type $SYS
STAT d:filename.type $DIR
```

Mit den oben angegebenen Befehlszeilen werden die Dateiattribute für die durch "d:filename.type" spezifizierte Datei die Attribute R/O und SYS gesetzt oder gelöscht.
Mit R/O wird der Schreibschutz gesetzt und er bleibt erhalten, bis er durch STAT mit dem Befehl R/W wieder gelöscht wird. Die Datei wird im Inhaltsverzeichnis mit dem entsprechenden Indikator versehen, so daß der Schreibschutz auch nach einem Kaltstart noch vorhanden ist.
Mit dem SYS-Attribut wird im Directory ein SYS-Indikator gesetzt, der zur Folge hat, daß die Datei beim DIR-Kommando nicht aufgeführt wird. Dieser Indikator wird durch das Attribut DIR wieder aufgehoben. Die Dateispezifikation kann eindeutig oder mehrdeutig sein. Ist die Dateispezifikation mehrdeutig, werden alle Dateien nach der Änderung auf dem Bildschirm aufgelistet. Der Laufwerksname ist optional.
Wird versucht, eine schreibgeschützte Datei zu löschen oder auf sie zu schreiben, erscheint eine Fehlermeldung auf der Konsole. Durch eine beliebige Eingabe auf diese Meldung hin wird ein Warmstart ausgelöst (z.B.: Return).
Die Eigenschaften eines Laufwerks "d:" lassen sich ermitteln, indem das Kommando

```
STAT d:DSK:
```

gegeben wird (d: ist ein Laufwerkname; A ... P). Die Eigenschaften des Laufwerks werden in folgender Form aufgelistet:

```
    d:   Drive Characteristics
65536:   128 Byte Record Capacity
 8192:   Kilobyte Drive Capacity
  128:   32 Byte Directory Entries
    0:   Checked Directory Entries
 1024:   Records/Extend
  128:   Records/Block
   58:   Sectors/Track
    2:   Reserved Tracks
```

Nach der Laufwerksangabe (d:) folgt die Gesamt-Block-Kapazität (bei 65536 handelt es sich um ein 8-MByte-Laufwerk) und danach die Kapazität in Kilobyte. Es folgen die Größe des Directories und die Anzahl der überprüften (checked) Eintragungen. Der Prüfmechanismus gestattet es, festzustellen, ob das Speichermedium (z.B.: Diskette) während einer Operation gewechselt wurde, ohne daß ein Warmstart erfolgte. Bei herausnehmbaren Medien ist dieser Wert meist gleich der Größe des Directory, bei fest eingebauten Speichern ist der Wert meist 0, da hier das Speichermedium ohne Warm- oder Kaltstart nicht gewechselt werden kann.
Die Adressierungskapazität des Directories wird aus der Zahl der Blöcke pro Extension berechnet (im Beispiel 128 Rec/Block * 128 Bytes/Rec oder 16KByte/Block). Es folgt dann die Angabe der Sektoren pro Spur und die Zahl der Spuren, die z.B. für das System reserviert wurden. Dieser Mechanismus erlaubt es aber auch, auf einem physikalischen Laufwerk durch Reservieren der Spuren mit niedrigeren Nummern mehrere logische Laufwerke zu implementieren. Es werden die Bereiche übersprungen, die zu anderen logischen Laufwerken gehören.
Mit dem Kommando

```
STAT DSK:
```

erhält man die Laufwerkscharakteristiken aller aktiven Laufwerke. Das letzte STAT-Kommando hat die Form

```
STAT USR:
```

Es wird aufgelistet, welche Benutzernummer eingestellt ist und für welche Benutzernummern Dateien auf der Diskette im aktuellen Laufwerk existieren. Zum Beispiel:

```
Active User  : 0
Active Files : 0 1 3
```

Die erste Zeile gibt die aktuelle Benutzernummer an (sie wurde mit dem letzten USER-Kommando angegeben). Danach folgt dann eine Liste der Benutzernummern, für die Dateien im Directory existieren. Im Beispiel oben ist die aktuelle Benutzernummer 0 (wie nach einem Kaltstart) und es gibt Dateien für drei Benutzer auf der Diskette. Wenn der Bediener mit USER 1 bzw. USER 3 auf die anderen Benutzerbereiche umschaltet, kann er mit dem DIR-Kommando das Inhaltsverzeichnis der jeweiligen Benutzernummern ansehen.

1.6.2 PIP

PIP (Peripheral Interchange Program) ist ein Programm für den Informationsaustausch mit Peripherieeinheiten. Es stellt die grundlegenden Konversionsroutinen zum Laden, Stanzen, Drucken Kopieren und Kombinieren von Dateien zur Verfügung. Das PIP-Kommando wird durch eine der beiden folgenden Eingaben gestartet:

```
PIP
```

oder

```
PIP Befehlszeile
```

In beiden Fällen wird PIP in die TPA geladen und gestartet. Bei der ersten Kommandoform fordert PIP die Befehlszeilen fortlaufend von der Konsole an. PIP meldet sich solange mit "*" in der ersten Spalte, bis eine leere Befehlszeile abgeschickt wird (z.B. Return). Jede Befehlszeile bewirkt eine der unten erläuterten Konversionen.
Die zweite Form des PIP-Kommandos ist eine einzige Aktion, die sofort ausgeführt wird. Danach wird zum CCP zurückgesprungen ohne daß PIP weitere Eingaben anfordert.
Die Befehlszeilen haben die folgende Gestalt:

```
Ziel=Quelle#1,Quelle#2,Quelle#3, ... ,Quelle#n
```

"Ziel" ist die Datei oder das Peripheriegerät, das die Daten empfängt. "Quelle#1 ... Quelle#n" ist eine Reihe von Dateien oder Peripheriegeräten, von denen Daten auf das Ziel kopiert werden. Dabei wird die Reihe von links nach rechts bearbeitet (die einfachste Befehlszeile hat also die Form "Ziel=Quelle").
Falls mehrere Quelldateien in der Kommandozeile angegeben werden, wird vorausgesetzt, daß die Dateien ASCII-Zeichen enthalten und mit dem Dateiendezeichen (end-of-file) SUB (Control/Z) abgeschlossen sind (der Parameter O hebt diese Voraussetzung auf). Um mit den Datei- und Gerätenamenskonventionen zu bleiben, werden vom PIP die Kleinbuchstaben in Großbuchstaben umgewandelt. Die Befehlszeile kann maximal 255 Zeichen lang sein (Falls die Eingabezeile länger als eine Konsolzeile ist, kann mit Control/E ein Zeilenvorschub erzeugt werden).
Ziel und die Quellen müssen Gerätebezeichnungen oder eindeutige Dateinamen (mit oder ohne Laufwerksangabe) sein. Das bedeutet, daß sowohl Ziel als auch Quellen auf den verschiedensten Laufwerken stehen können. Falls kein Laufwerk angegeben ist, wird das angemeldete Laufwerk verwendet. Die Ziel-Datei kann auch ein- oder mehrmals auf der rechten Seite als Quelle vorkommen. In diesem Fall wird die Zieldatei erst konvertiert, wenn alle Konversionen und Konkatenationen auf der rechten Seite vollzogen wurden. Ist die Kommandozeile fehlerfrei und ist die Zieldatei bereits vorhanden, wird die Zieldatei überschrieben (sie wird nicht verändert, wenn sich ein Fehler in der Befehlszeile befindet). Es folgen einige Beispiele für korrekte Befehlszeilen mit Erläuterungen:

```
PIP X=Y
```
Kopiere die Datei Y auf die Datei X. X und Y sind eindeutige Dateinamen. Die Datei Y bleibt unverändert.

```
PIP X=Y,Z
```
Konkateniere die Dateien Y und Z und kopiere das Ergebnis auf X. Y und Z werden nicht verändert. Konkatenation ist das Hintereinanderhängen von Dateien.

```
PIP X.ASM=Y.ASM,Z.ASM,FIN.ASM
```
Die Datei X setzt sich aus den Dateien Y+Z+FIN zusammen. Alle Dateien sind vom Typ ASM.

```
PIP NEW.ZOT=B:OLD.ZAP
```
Kopiere die Datei OLD.ZAP vom Laufwerk B auf die Datei NEW.ZOT auf dem aktuellen Laufwerk.

```
PIP B:A.U=B:B.V,A:C.W,D.X
```
Es werden Dateien von den verschiedensten Laufwerken (B, A und aktuelles Laufwerk) in einer Datei A.U auf Laufwerk B zusammengefaßt.

Die PIP-Kommandos zum Übertragen von Dateien von einem Laufwerk auf das andere können auch abgekürzt werden. Die Kurzformen lauten:

```
PIP d:=afn

PIP d1:=d2:afn

PIP ufn=d:

PIP d1:ufn=d2:
```

Das erste Kommando kopiert alle Dateien des aktuellen Laufwerks, die der mehrdeutigen Dateireferenz ufn genügen, auf das Laufwerk d (d = A ... P). Sie haben dort die gleichen Namen.
Das zweite Kommando hat die gleiche Wirkung wie das erste, es wird jedoch das Quell-Laufwerk explizit angegeben.
Das dritte Kommando ist eine Abkürzung für "PIP d1:ufn=d2:ufn" wobei d1 das aktuelle Laufwerk ist.
Das vierte Kommando kopiert die Datei ufn auf dem Laufwerk d2 auf eine Datei gleichen Namens auf dem Laufwerk d1.
In allen Fällen muß das Laufwerk, auf dem die Quelldateien liegen, verschieden sein von dem Laufwerk, auf dem die Zieldateien liegen. Falls eine mehrdeutige Dateireferenz angegeben wurde, dann listet PIP alle Dateien, die kopiert werden, auf der Konsole. Falls auf der Zieldiskette eine Datei mit demselben Namen existiert, wird sie überschrieben - vorausgesetzt, die Operation verläuft fehlerfrei.
Die folgenden PIP-Kommandos sind Beispiele für zulässige Disk-zu-Disk-Übertragungen:

```
PIP B:=*.68K
        Kopiert alle Kommandodateien (mit dem Typ 68K) vom
        aktuellen Laufwerk auf das Laufwerk B.

PIP A:=B:ZAP.*
        Kopiert alle Dateien mit dem (primären) Namen "ZAP" von
        Laufwerk B auf A.

PIP ZAP.ASM=B:
        Gleichwertig zu PIP ZAP.ASM=B:ZAP.ASM .

PIP B:ZOT.68K=A:
        Gleichwertig zu PIP B:ZOT.68K=A:ZOT.68K .

PIP B:=GAMMA.BAS
        Gleichwertig zu PIP B:GAMMA.BAS=GAMMA.BAS

PIP B:=A:GAMMA.BAS
        Gleichwertig zu PIP B:GAMMA.BAS=A:GAMMA.BAS
```

Mit PIP kann aber auch auf logische und physikalische Geräte, die an den CP/M-Computer angeschlossen sind, zugegriffen werden. Die Gerätenamen sind die gleichen wie beim STAT-Kommando. Zusätzlich gibt es noch die Namen einiger Spezialgeräte, die weiter unten erklärt werden. Die logischen Geräte sind (siehe STAT):

CON:	Konsole	(console)
AXI:	Externes Eingabegerät	(auxiliary input)
AXO:	Externes Ausgabegerät	(auxiliary output)
LST:	Drucker	(list)

Die physikalischen Geräte sind:

TTY:	Sichtgerät, Teletype (Fernschreiber mit LS-Leser und LS-Stanzer), Drucker mit Tastatur, ...
CRT:	Sichtgerät, auch mit Drucker
UC1:	vom Benutzer definiertes Konsolgerät
PTR:	Lochstreifenleser
UR1: UR2:	vom Benutzer definierter LS-Leser
PTP:	Lochstreifenstanzer
UP1: UP2:	vom Benutzer definierter LS-Stanzer
LPT:	Drucker

UL1: vom Benutzer definierter Drucker

Das physikalische Gerät BAT: ist nicht dabei, weil diese Zuweisung nur angibt, daß die Konsole durch AXI: und LST: ersetzt wird. Die Gerätetreiber für CON, AXI, AXO und LST werden alle im BIOS-Teil des CP/M definiert. Sie können leicht an die jeweilige Hardwareumgebung angepasst werden. Die Zuweisung der aktuellen physikalischen Geräte erfolgt über das IOBYTE (siehe Kapitel 6). Das Zielgerät muß in der Lage sein, Daten zu empfangen (z.B. Drucker oder Stanzer). Desgleichen müssen die Quellgeräte in der Lage sein Daten zu senden oder zu generieren (so kann z.B. das LST:-Gerät keine Daten erzeugen).
Zusätzlich gibt es einige Sonder- oder Pseudogeräte im PIP-Kommando:

NUL: Schicke 40 Nullbytes (ASCII NUL) an das Zielgerät (z.B.: am Ende eines Stanz-Auftrags).

EOF: Schicke ein CP/M-Dateiende (end-of-file) an das Zielgerät (ASCII: SUB; Control/Z). Nach Beendigung der Übertragung einer ASCII-Datei wird es automatisch vom PIP geschickt.

PRN: Hat die gleiche Funktion wie LST:. Es werden jedoch die Tabulatorstops auf jeder achten Zeichenposition angenommen. Nach jeweils 60 Zeilen wird ein Seitenvorschub eingefügt (hat die gleiche Funktion wie der PIP-Parameter t8np).

überall in der PIP-Befehlszeile können Datei- und Gerätenamen stehen. In jedem Fall wird die angegebene Datei bis zum Ende (end-of-file) gelesen. Bei ASCII-Dateien ist dies das Zeichen SUB (Control/Z), bei den anderen end-of-data. Bei der Konkatenation von Quellen wird die Befehlszeile von links nach rechts abgearbeitet. Das Zielgerät oder die Zieldatei werden mit den Daten der Quelle(n) fortlaufend beschrieben. Bei ASCII-Dateien wird zum Schluß ein end-of-file (SUB = Control/Z) angehängt. Ist das Ziel eine Diskettendatei, wird eine Arbeitsdatei mit "$$$" als Dateityp kreiert . Nach fehlerfreier Bearbeitung wird die Zieldatei gelöscht, falls sie schon existierte und die $$$-Datei auf den Namen der Zieldatei umbenannt. Dateien mit dem Typ 68K werden immer als Nicht-ASCII-Dateien interpretiert.
Eine Kopieroperation kann jederzeit durch das Drücken einer beliebigen Taste abgebrochen werden (z.B. Return). PIP gibt dann die Meldung "ABORTED" aus, um anzuzeigen, daß die Operation nicht korrekt beendet wurde. Wird im PIP abgebrochen oder tritt ein Fehler auf, werden vom PIP auch alle offenen (nicht ausgeführten) Kommandos während einer SUBMIT-Operation gelöscht und so das SUBMIT abgebrochen.
Ist die Quelle ein Peripheriegerät (z.B. Lochstreifenleser) und das Ziel eine Diskettendatei vom Typ HEX (Datei im Intel-Hexformat), dann wird eine Sonderfunktion des PIP aktiviert. In einem solchen Fall prüft PIP die Quelle, ob es sich um eine korrekte HEX-Datei mit ausschließlich Hexadezimalziffern und einwandfreien Block-Prüfsummen handelt. Wird ein Fehler gefunden,

gibt PIP eine Fehlermeldung aus und wartet auf eine Korrektureingabe. Meist genügt es, den Lochstreifen etwa 50 cm zurückzuspulen und nochmals einzulesen. Wenn der Steifen zum nochmaligen Lesen bereit ist, genügt das Drücken der Return-Taste um den PIP wieder anlaufen zu lassen. Läßt sich die Eingabe auf diese Weise nicht korrigieren, kann PIP mit Return zum Weitermachen aufgefordert werden und die fehlerhafte Stelle in der Datei nachträglich mit dem ED korrigiert werden.
Es ist weiter möglich, das Dateiende von der Tastatur aus einzugeben. Wenn es sich bei der Quelle um das Gerät AXI: handelt, überwacht PIP während des Lesens die Tastatur. Wenn Control/Z eingegeben wird, beendet PIP den Lesevorgang.
Beispiele für PIP-Kommandos:

PIP LST:=X.PRN
Ausgabe der Datei X.PRN auf dem Drucker. Anschließend wird zum CCP zurückgekehrt.

PIP
Start des PIP für eine Befehlssequenz (PIP meldet sich mit "*").

* CON:=X.ASM,Y.ASM,Z.ASM
Als erstes werden die Dateien X.ASM,Y.ASM und Z.ASM hintereinander auf der Konsole ausgegeben.

* X.HEX=CON:,Y.HEX,PTR:
Kreiere eine HEX-Datei, indem zuerst von der Konsole gelesen wird, bis ein Control/Z eingetippt wird, danach wird die Datei Y.HEX gelesen und schließlich Daten vom LS-Leser übernommen (bis Control/Z gelesen wird).

<Return-Taste>
Return beendet den PIP.

PIP AXO:=NUL:,X.ASM,EOF:,NUL:
Stanzen einer Datei X.ASM mit je 40 Nullen als Vor- und Nachspann. Die Datei wird mit Control/Z abgeschlossen.

Weiter kann der Benutzer dem PIP etliche Parameter mitgeben, die in eckige Klammern eingeschlossen werden. Sie können durch Leerzeichen getrennt werden. Jeder Parameter beeinflußt die Kopieroperation, weshalb die Parameterliste den Namen der betroffenen Dateien oder Geräten folgen muß. Allgemein kann der Parameter von einer dezimalen Zahl gefolgt werden (Ausnahme: S und Q). Die PIP-Parameter sind:

B
Übertragung im Blockmodus: die Daten werden vom PIP gepuffert, bis ein ASCII x-off (DC3) vom Quellgerät empfangen wird. Das ermöglicht es, Daten für eine Diskettendatei von einem Gerät zu empfangen, das einen kontinuierlichen Datenstrom liefert (z.B.: Kassettengerät). Nach dem Empfang des DC3 werden die Daten wei-

tergegeben, der Puffer gelöscht und PIP wartet auf neue Daten. Die Puffergröße hängt vom verfügbaren Speicher ab, PIP gibt beim Pufferüberlauf eine Fehlermeldung aus.

Dn
Löscht beim Kopieren alle Zeichen in einer Zeile, die hinter der n-ten Spalte stehen. Dieser Parameter wird häufig verwendet, Dateien mit langen Zeilen auf einen schmalen Drucker oder Bildschirm zu kopieren.

E
(Echo) Alle Übertragungsoperationen werden auf der Konsole protokolliert.

F
(Filter)Entfernen aller Seitenvorschübe beim Kopieren. Gleichzeitig kann P verwendet werden, um neue Seitenvorschübe zu erzeugen.

Gn
(Get) Holt eine Datei aus einem anderen Benutzerbereich mit der Nummer n (n = 0 ... 15).

H
Datenübertragung von HEX-Dateien. Die Daten werden auf Korrektheit (Prüfsummen) getestet, überflüssige Zeichen entfernt. Tritt ein Fehler auf, wird eine Korrektureingabe von der Konsole angefordert.

I
Ignoriere die ":00"=Sätze in einer Intel HEX-Datei (I impliziert den Parameter H).

L
ändere Großbuchstaben in Kleinbuchstaben.

N
Numeriere die Zeilen. Die Numerierung beginnt bei 1 und die Zeilennummern werden um 1 erhöht. Führende Nullen werden unterdrückt und die Nummer wird von einem Doppelpunkt gefolgt. Wird N2 spezifiziert, werden führende Nullen gedruckt und ein Tab-Zeichen nach der Nummer eingefügt. Der Tabulator wird expandiert, wenn der Parameter T spezifiziert wird.

O
Objektdatei-Übertragung (kein ASCII), das CP/M end-of-file wird ignoriert.

Pn
Füge nach n Zeilen einen Seitenvorschub ein (mit Seitenvorschub am Anfang). Falls n=1 ist oder fehlt, wird n=60 gesetzt. Wird der Parameter F auch angegeben, werden die in der Datei vorhandenen Seitenvorschübe entfernt, bevor die neuen eingefügt werden.

Qs^z

Beende das Kopieren, wenn die Zeichenkette s angetroffen wird. Die Zeichenkette wird im Kommando mit Control/Z abgeschlossen.

R
Lies eine Systemdatei.

Ss^z
Starte das Kopieren erst dann, wenn in der Eingabe der String s angetroffen wird. Die Parameter S und Q lassen sich verwenden den bestimmten Teil einer Quelldatei zu kopieren. Die Start- und Stopstrings werden mit kopiert.

Verwendet der Benutzer die Form (1) des PIP, dann werden die Strings bei Q und S vom CCP in Großbuchstaben umgewandelt. Bei der Form (2) bleiben Kleinbuchstaben erhalten.

(1) PIP "Befehlszeile"
(2) PIP

Tn
Expandiere Tabulatorzeichen (Control/I) beim Kopieren. Tabulatorstops in jeder n-ten Spalte .

U
Konvertiere beim Kopieren Kleinbuchstaben zu Großbuchstaben

V
Verifiziere die Daten bei der Übertragung. Ziel muß eine Diskettendatei sein. Nach dem Schreiben wird der entsprechende Block nocheinmal gelesen und mit den Ausgangsdaten verglichen.

W
Ignoriere einen möglichen Schreibschutz.

Z
Setze das Paritätsbit der gelesenen Zeichen auf 0.

Zulässige PIP-Kommandos mit Parametern:

PIP X.ASM=B:[V]
Kopiere X.ASM vom Laufwerk B auf das aktuelle mit verifizieren der Daten.

PIP LPT:=X.ASM[NT8U]
Kopiere X.ASM auf den Drucker, nummeriere die Zeilen expandiere die Tabulatoren (Tabulatorstops jede achte Spalte) und konvertiere Klein- zu Großbuchstaben.

PIP AXO:=X.HEX[I],Y.ZOT[H]
Kopiere X.HEX auf den Stanzer und ignoriere dabe ":00"-Sätze in der Datei. Dannach lies die HEX-Datei Y.ZOT (hier werden die ":00"-Sätze kopiert).

```
PIP X.LIB=Y.ASM[sSUBR1:^z qJMP L3^z]
```

Aus der Datei Y.ASM wird ein Stück beginnend mit "SUBR1" und endend mit "JMP L3" auf die Datei X.LIB kopiert.

```
PIP PRN:=X.ASM [P50]
```

Die Datei X.ASM wird auf den Drucker geschickt. Die Zeilen werden nummeriert, Tabulatoren expandiert (Stops alle 8 Spalten) und alle 50 Zeilen ein Seitenvorschub erzeugt. Für PRN: ist die Parameterfolge "NT8P60" voreingestellt. Der Parameter P50 überschreibt die Voreinstellung.

Normalerweise kann PIP keine schreibgeschützte Datei überschreiben. Wird versucht eine geschützte Datei zu beschreiben, gibt PIP auf der Konsole aus:

```
DESTINATION FILE IS R/O, DELETE (Y/N) ?
```

Antwortet der Benutzer mit Y (Yes = Ja), wird die Datei überschrieben. Im anderen Fall gibt PIP die Antwort

```
** NOT DELETED **
```

und der Datentransfer findet nicht statt. PIP fährt dann mit der nächsten Anweisung fort. Die Anfrage von PIP bei schreibgeschützten Dateien kann verhindert werden, indem man den Parameter W angibt. Dann wird der Schreibschutz ignoriert. Das Kommando

```
PIP A:=B:*.68K[W]
```

kopiert alle 68K-Dateien, die nicht SYS-Dateien sind von Laufwerk B nach Laufwerk A. Schreibgeschützte Dateien werden überschrieben. Werden Dateien konkateniert, braucht der W-Parameter nur bei der letzten Datei in der Liste angegeben werden. Wird zusätzlich der Parameter R angegeben, lassen sich auch Dateien mit System-Attribut kopieren. Wird R nicht angegeben, erkennt PIP die SYS-Dateien nicht. Das Kommando

```
PIP ED.68K=B:ED.68K[R]
```

kopiert den ED von Laufwerk B auf das aktuelle ohne Rücksicht auf Schreibschutz oder Systemattribut. Wenn vorhanden werden die Systemattribute mit kopiert. Zu früheren Versionen von CP/M kann die Abwärts-Kompatibilität nur unter folgenden Bedingungen gewahrt werden:

- Die Datei ist nicht größer als ein Megabyte.
- Es existieren keine Dateiattribute.
- Die Datei wurde unter User 0 kreiert.

Soll die Kopie zu früheren CP/M-Versionen kompatibel sein, die nicht der Standardkonfiguration entsprechen (z.B.:doppelte Schreibdichte) muß u.U. das BIOS entsprechend angepasst werden Um Dateien in einen anderen Benutzerbereich zu Kopieren, kann die G-Option bei Quell- und Zieldatei angegeben werden, und zwar hinter dem Dateinamen. Will man zum Beispiel die Datei TEST im

Benutzerbereich 0 auf die Datei ZIEL im Benutzerbereich 3 kopieren, lautet das Kommando:

```
PIP ZIEL[G3]=TEST
```

1.6.3 ED ufn

ED ist der Editor des CP/M. Er ermöglicht die Erstellung und Änderung von ASCII-Dateien auf einem CP/M-Computer. Die Einzelheiten des ED sind im folgenden Kapitel beschrieben. Mit ED kann der Benutzer Quelldateien kreieren und bearbeiten, die aus Zeilen von ASCII-Zeichen bestehen. Jede Zeile wird durch das Symbol "end-of-line" (= Zeilenende) abgeschlossen, das aus den beiden Zeichen Return (0DH) und Linefeed (0AH) besteht. Es gibt praktisch keine Beschränkung der Zeilenlänge (keine Zeile kann jedoch die Länge des Arbeitsspeichers überschreiten). Die Länge einer Zeile wird definiert als die Zahl der eingetippten Zeichen zwischen zwei Return-Zeichen. Das ED-Programm kennt Kommandos zum Suchen, Ersetzen, Löschen und Einfügen von Strings (= Zeichenketten). Diese Kommandos eignen sich zum Erstellen und Ändern von Programm- und Textdateien unter CP/M. Obwohl der Arbeitsspeicher des CP/M-Computers begrenzt ist gibt es für die erstellte Datei keine Größenbeschränkung, da die Datei "seitenweise" in den Arbeitsspeicher gebracht wird.
Falls eine Quelldatei noch nicht existiert, wird sie von ED kreiert und für den Zugriff eröffnet. Ist die Quelldatei schon vorhanden (siehe A-Kommando), kann der Benutzer Daten an die Datei anhängen. Die neue eingegebenen Zeilen können dann aufgelistet, geändert und zurück auf die Diskette geschrieben werden (W-Kdo.). Bestimmte Programmteile lassen sich automatisch im Kontext suchen (N-Kdo.). Es kann sehr einfach auf bestimmte Teile einer großen Datei zugegriffen werden. Gibt der Bediener

```
ED X.ASM
```

ein, dann legt ED eine Arbeitsdatei namens

```
X.$$$
```

an. Während eines ED-Laufs, enthält diese Datei die edierten Daten. Nach Beendigung des ED wird die Originaldatei X.ASM in X.BAK (BAK => back-up) umbenannt und die edierte Datei X.$$$ erhält den ursprünglichen Namen X.ASM. X.BAK enthält also die Quelle, wie sie vor dem Edieren war, X.ASM die neue, geänderte Datei. Ein Benutzer kann also jederzeit auf die vorhergehende Version einer Datei zurückgreifen. Er muß dazu nur die neueste Version löschen und die BAK-Version umbenennen. Wurden beim Edieren von X.ASM Fehler gemacht, kann man mit der folgenden Kommandosequenz den früheren Zustand wieder herstellen:

```
DIR X.*                 Feststellen, ob X.BAK da ist.
ERA X.ASM               Neueste, falsche Version löschen.
REN X.ASM=X.BAK         Umtaufen der BAK-Datei.
```

Der ED kann jederzeit abgebrochen werden (Stromausfall, Control/C, Q-Kommando), ohne daß die Originaldatei zerstört wird. In diesem Fall wird keine BAK-Datei kreiert und die Originaldatei bleibt unversehrt erhalten.
Es ist auch möglich, die Originaldatei und die BAK-Datei auf verschiedenen Laufwerken zu halten. Das ED-Kommando hat dann das Format:

```
ED ufn d:
```

ufn ist der Name der zu edierenden Datei auf dem aktuellen Laufwerk und d: ist der Name eines anderen Laufwerks. ED liest und bearbeitet die Datei ufn; die neue Datei ufn wird dann auf das Laufwerk d geschrieben; nach der Beendigung des ED verbleibt auf dem aktuellen Laufwerk die Datei ufn.BAK. Ist A das aktuelle Laufwerk, dann ist das folgende Kommando zulässig:

```
ED X.ASM B:
```

Es wird die Datei A:X.ASM ediert, indem auf Laufwerk B eine Datei X.$$$ angelegt wird. Nach dem erfolgreichen Edieren wird A:X.ASM in A:X.BAK umbenannt und B:X.$$$ wird zu B:X.ASM umbenannt. Aus Gründen der Bequemlichkeit wird B dann auch als aktuelles Laufwerk angemeldet. Existiert vor dem Ediervorgang auf Laufwerk B bereits eine Datei B:X.ASM, gibt ED die Meldung

```
FILE EXISTS
```

(Datei existiert) aus. Es soll so verhindert werden, daß eine vorhandene Quelldatei zerstört wird. Die vorhandene Datei B:X.ASM muß gelöscht oder umbenannt werden und dann der ED neu gestartet werden.
Wie bei allen anderen Kommandos, muß der Ediervorgang nicht auf dem angemeldeten Laufwerk stattfinden, indem dem Dateinamen der Laufwerksname vorangestellt wird.

```
ED A:X.ASM
```
Edieren der Datei X.ASM auf dem Laufwerk A. Die neue Datei X.ASM und ebenso X.BAK befinden sich auf Laufwerk A.

```
ED B:X.ASM A:
```
Edieren der Datei X.ASM auf Laufwerk B. Nach dem ED-Lauf befinden sich X.ASM auf Laufwerk A und X.BAK auf Laufwerk B.

1.6.4 SUBMIT ufn param#1 ... param#n

Mit dem SUBMIT-Kommando lassen sich Kommandofolgen als Stapelauftrag (batch-job) abarbeiten. ufn ist der (eindeutige) Name einer Datei auf dem angemeldeten Laufwerk, die vom Typ SUB sein muß. Diese Datei enthält Prototypen von CP/M-Kommandos, die durch die Einsetzung der SUBMIT-Parameter param#1 ... param#n zu korrekten CP/M-Kommandos werden. Falls keine Fehler auftreten, werden die Zeilen in der SUB-Datei sequentiell abgearbeitet. Die Datei wird mit einem Texteditor, z.B. dem ED, erstellt. Es können formale Parameter der Form "$Zahl" im Text vorkommen:

```
$1 $2 $3 ... $n
```

Sie sind Platzhalter für die aktuellen Parameter, die beim SUBMIT-Aufruf angegeben werden. Bei der Bearbeitung der SUB-Datei durch SUBMIT werden die $-Parameter in der Datei durch die entsprechenden aktuellen Parameter ersetzt (param#1 für $1, param#2 für $2, usw). Stimmt die Zahl der aktuellen Parameter (in der Aufrufzeile) nicht mit der Zahl der formalen Parameter überein, wird SUBMIT abgebrochen und eine Fehlermeldung an die Konsole geschickt. Das SUBMIT-Kommando kreiert eine Datei mit dem substituierten Kommandos namens

```
$$$.SUB
```

auf dem aktuellen Laufwerk. Nach dem Warmstart (der nach Ende des SUBMIT-Kommandos stattfindet) wird diese Kommandodatei anstelle der Konsole vom CCP als Eingabe verwendet. Wird das SUBMIT-Kommando mit einer Datei ausgeführt, die nicht auf Laufwerk A ist, können die Kommandos erst ausgeführt werden, wenn die entsprechende Diskette in Laufwerk A steckt und ein Systemwarmstart erfolgte. Der Benutzer kann die Berarbeitung der Stapel-Kommandos jederzeit abbrechen, indem er die Taste RUBOUT (DEL, DELETE) drückt, nachdem ein Kommando auf der Konsole protokolliert wurde. In diesem Fall wird die Datei $$$.SUB gelöscht und die weiteren Kommandos von der Konsole erwartet. Die Berarbeitung der Kommandos in $$$.SUB wird auch abgebrochen, wenn in irgendeinem Kommando ein Fehler auftritt. Die Bearbeitung der $$$.SUB-Datei kann auch abgebrochen werden, indem ein laufendes Programm die Datei $$$.SUB löscht.
Um Dollarzeichen in eine Submit-Datei einzutragen muß man "$$" eingeben. Innerhalb der Datei reduziert SUBMIT dann "$$" zu "$". Die Zeichenfolge "^X" wird auf ein einziges Zeichen, Control/X (ASCII CAN), reduziert.
Das letzte Kommando einer Submit-Datei kann wieder ein SUBMIT-Kommando sein. Es sind so verkettete Stapelaufträge möglich. Angenommen, die Datei ASMBL.SUB existiert auf der Diskette und sie beinhaltet die folgenden Kommandoprototypen:

```
ASM $1

DIR $1.*

ERA *.BAK
```

```
PIP $2:=$1.PRN

ERA $1.PRN
```

Wird nun das Kommando

```
SUBMIT ASMBL X PRN
```

gegeben, dann ersetzt SUBMIT beim Erzeugen der $$$.SUB-Datei $1 durch "X" und $2 durch "PRN". Die Datei $$$.SUB sieht dann so aus:

```
ASM X

DIR X.*

ERA *.BAK

PIP PRN:=X.PRN

ERA X.PRN
```

Sie wird dann sequentiell vom CCP bearbeitet.
SUBMIT kann auch auf SUB-Dateien zugreifen, die nicht auf Laufwerk A stehen; es muß dann der Laufwerkname vor dem Dateinamen angegeben werden. Die erzeugte $$$.SUB-Datei wird aber nur in Laufwerk A bearbeitet. So ist es möglich, die Datei zu erzeugen, sie aber erst später automatisch ablaufen zu lassen, indem die Diskette in Laufwerk A gebracht wird.

1.6.5 INIT d:

Mit diesem Kommando wird das Directory initialisiert. INIT löscht das gesamte Directory, ohne die Diskette zu formatieren, hat also denselben Effekt wie ERA *.* .

1.6.6 COPY <Liste von Optionen>

COPY kopiert eine ganze Diskette auf eine andere, indem Spur für Spur übertragen wird. Wenn beim Aufruf von COPY keine Optionen angegeben werden, fragt das Kommando im Dialog danach.

Optionen:

MODE	ALL	Kopiere die ganze Diskette
	BOOT	Kopiere nur die Boot-Spuren
	FILES	Kopiere nur die Dateien
	END	Beende das Programm

Danach fragt das Programm nach dem Quell-Laufwerk, also dem Laufwerk, in dem die zu kopierende Diskette sitzt:

```
Enter SOURCE drive: _
```

und anschließend nach dem Ziel-Laufwerk, in dem sich eine leere, formatierte Diskette befindet:

```
Enter DESTINATION drive: _
```

Jetzt haben Sie im Falle, daß Sie sich geirrt haben, noch die Möglichkeit, mit CONTROL/C das Programm abzubrechen.

Sie Können alle Angabe aber schon beim Kommandoaufruf machen, wie die folgenden Beispiele zeigen. Die Reihenfolge ist Modus, Quell-Laufwerk, Ziel-Laufwerk, Zusatzoptionen:

```
COPY ALL
COPY ALL A
COPY ALL A B
COPY ALL A B [A]
COPY ALL A B [V]
COPY ALL A B [AV]
```

Die Zusatzoptionen A und V haben folgende Bedeutung (wobei das A vor dem V stehen muß):

Bei der Option A wird das Kopieren sofort gestartet, ohne, daß das Programm nocheinmal eine Bestätigung durch RETURN abfordert. Bei der Option V wird eine Prüfroutine beim Kopieren eingeschaltet, die eine korrekte Übertragung der Daten sicherstellt (ähnlich wie bei PIP).

1.7 Dateitypen

Einige Dateitypen haben eine spezielle Bedeutung unter CP/M 68 K. Etliche Kommandos benötigen keine Eingabe des Dateityps, setzen aber einen bestimmten Typ voraus. So haben z. B. ausführbare Programme (transiente Kommandos) den Typ '.68K'.

.S	Assemblerquelle
.BAK	Backupdatei des Editors
.C	Quelle eines C-Programms
.68K	ausführbares Programm
.H	Header-Datei für C
.HEX	Datei im Intel-Hexformat
.O	Objektcode-Datei
.PRN	Druckdatei
.REL	relokatives Programm, vom Linker erzeugt
.SUB	Submit-Auftrag
.SYS	CP/M 68 K - Systemdatei (nicht mit dem SYS-Attribut verwechseln!)
.$$$	Zwischendatei

1.8 Fehlerbehandlung

Wenn ein Programm eine Funktion des BDOS aufruft, kann von diesem ein Fehler gemeldet werden. Der CCP meldet folgende Fehler auf der Konsole:

CP/M DISK READ ERROR ON DRIVE x:	Lesefehler
CP/M DISK WRITE ERROR ON DRIVE x:	Fehler beim Schreiben
CP/M DISK SELECT ERROR ON DRIVE x:	Laufwerk nicht bereit
CP/M DISK CHANGE ERROR ON DRIVE x:	Die Diskette wurde gewechselt
CP/M DISK FILE ERROR: ffffffff.ttt IS READ-ONLY	Die Datei ist schreibgeschützt

Danach fragt das System:

Do You want to Abort (A), Retry (R), or Continue with bad data (C)

Wenn Sie A eingeben, wird das Kommando abgebrochen.
Wenn Sie R eingeben, wird versucht, die Operation noch einmal auszuführen.
Wenn Sie C eingeben, wird der Fehler ignoriert und im Programm fortgefahren.

Wenn Sie ein Laufwerk anwählen, das nicht bereit ist oder garnicht existiert, meldet sich der CCP mit der Frage:

Do you want to Abort (A) or Retry (R)?

Eine Eingabe von A bricht das Programm ab. Sie haben aber auch die Möglichkeit, das Laufwerk, zum Beispiel durch Einstecken einer Diskette, bereit zu machen und das Programm dann mit R fortzusetzen.
Bei der letzten Fehlermeldung (Datei schreibgeschützt) fragt der CCP:

Do you want to: Change it to read/write (C), or Abort (A)?

Man kann nun durch die Eingabe von A abbrechen oder durch C den Schreibschutz aufheben.

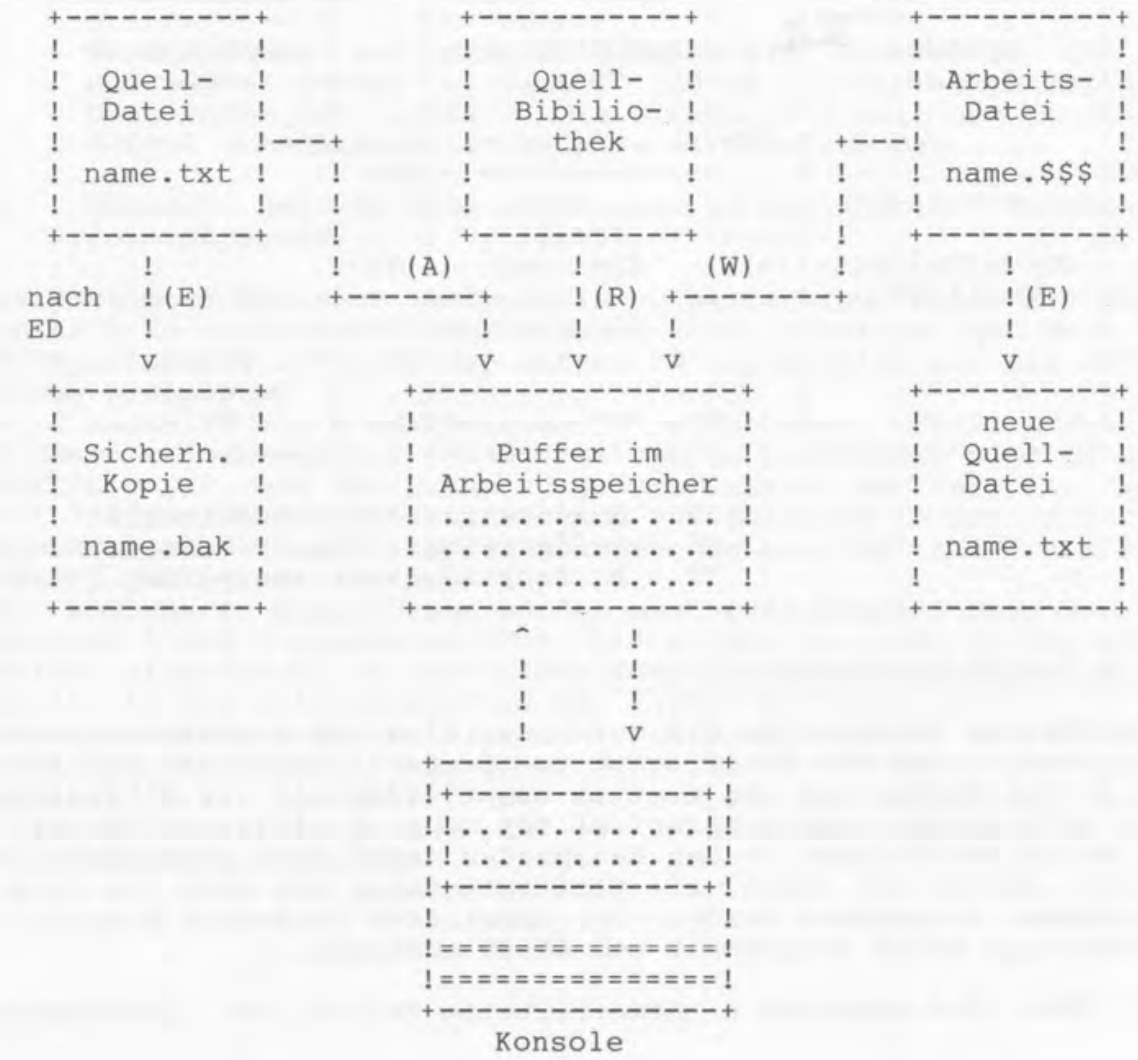
2 Texteditor ED

2.1 Einführung

ED ist der Texteditor von CP/M. Er wird verwendet, um Quelldateien zu erzeugen und zu ändern. Der ED wird in CP/M mit

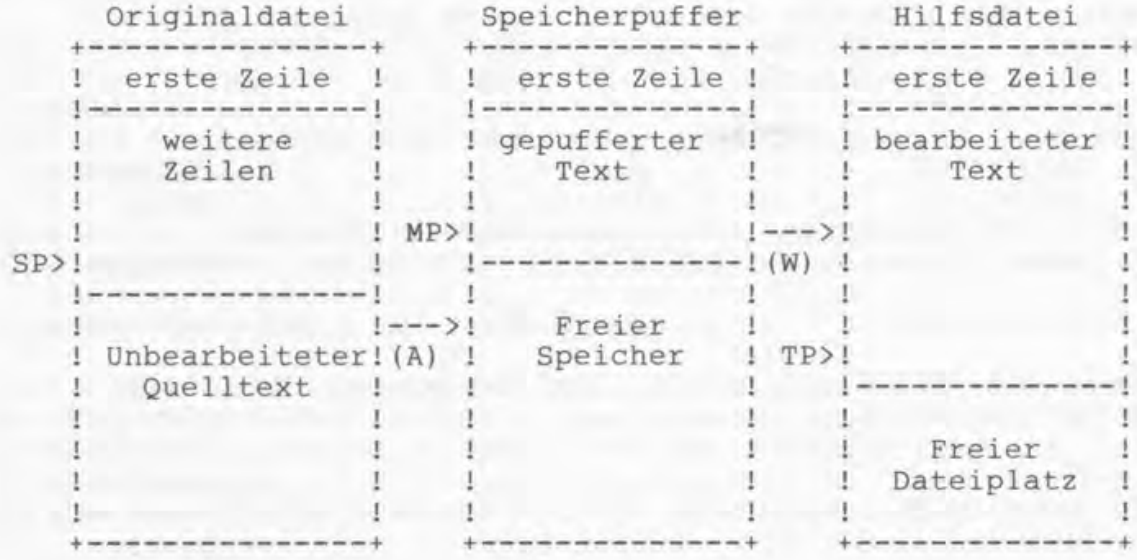
```
     ED dateiname
oder
     ED dateiname.typ
```

gestartet. Es muß beim Aufruf immer ein Dateiname angegeben werden. Der ED liest einen Abschnitt der angegebenen Quelldatei in den Hauptspeicher, wo die Datei vom Benutzer bearbeitet wird. Die modifizierte Quelle wird dann zurück auf Diskette geschrieben. Existiert die Quelldatei noch nicht, wird sie vom ED kreiert. Den prinzipiellen Ablauf von ED zeigt das folgende Diagramm.



2.1.1 Betrieb des ED

ED bearbeitet eine Quelldatei (im Diagramm "name.txt") im Speicher-Puffer. Dort kann der gesamte Text angesehen und verändert werden. Die Zahl der Zeilen, die gepuffert werden hängt von der Zeilenlänge ab. Durch Anweisungen des Benutzers werden die edierten Daten dann auf eine temporäre Arbeitsdatei geschrieben. Bei Beendigung des ED wird der Puffer auf Datei geschrieben und daran der restliche Text aus der Ausgangsdatei angefügt. Der Name der Originaldatei wird auf den Typ 'BAK' geändert (BAK für 'back-up'). So kann gegebenenfalls immer auf die letzte Version der Datei zurückgegriffen werden (siehe auch die CP/M-Kommandos ERA und REN). Die Arbeitsdatei erhält dann den Typ der Originaldatei und sie stellt das Resultat des Ediervorgangs dar. Das folgende Diagramm zeigt die Struktur von Originaldatei, Puffer und Arbeitsdatei.



SP = Quellzeiger (Source Pointer)
MP = Pufferzeiger (Memory Pointer)
TP = Hilfsdateizeiger (Temporary Pointer)

2.1.2 Textübertagungsfunktionen

Verschiedene ED-Befehle, die aus einem einzigen Buchstaben bestehen, veranlassen die Übertragung von Textzeilen aus der Quelldatei in den Puffer und von dort in die Hilfsdatei. Die Hilfsdatei wird später dann (möglicherweise) zur neuen Quelldatei. Obwohl die Befehlsbuchstaben in den Beispielen immer groß geschrieben werden, können sie sowohl als Kleinbuchstaben wie auch als Großbuchstaben eingegeben werden. Der numerische Parameter n kann ganzzahlige Werte zwischen 0 und 65535 annehmen.

nA Fügt die nächsten n unbearbeiteten Zeilen der Quelldatei,

die auf den SP (siehe 2. Diagramm) folgen, hinter dem MP im Puffer an. SP und MP werden entsprechend erhöht. Falls die Umwandlung in Großbuchstaben gewünscht ist (siehe U-Befehl), und der A-Befehl als Großbuchstabe eingegeben wurde, erfolgt beim Einlesen die Konvertierung in Großbuchstaben.

nW Schreibt die ersten n Zeilen aus dem Textpuffer auf die Hilfsdatei. Die restlichen Zeilen im Puffer (ab n+1 bis MP) werden im Puffer ganz nach oben geschoben. Die Zeiger TP und MP werden aktualisiert.

E Beendet den Ediervorgang. Kopiert den Text aus dem Puffer und danach die unbearbeiteten Zeilen der Quelldatei auf die Hilfsdatei. Danach werden die Quelldatei und die Hilfsdatei wie oben geschildert umbenannt.

H Beginne die neue Datei von vorne. Die neue Datei wird zur Quelldatei, der Puffer gelöscht und eine leere Hilfsdatei wird eröffnet. Der H-Befehl hat die gleiche Wirkung wie ein E-Befehl, gefolgt von einem neuen ED-Kommando mit der gleichen Datei.

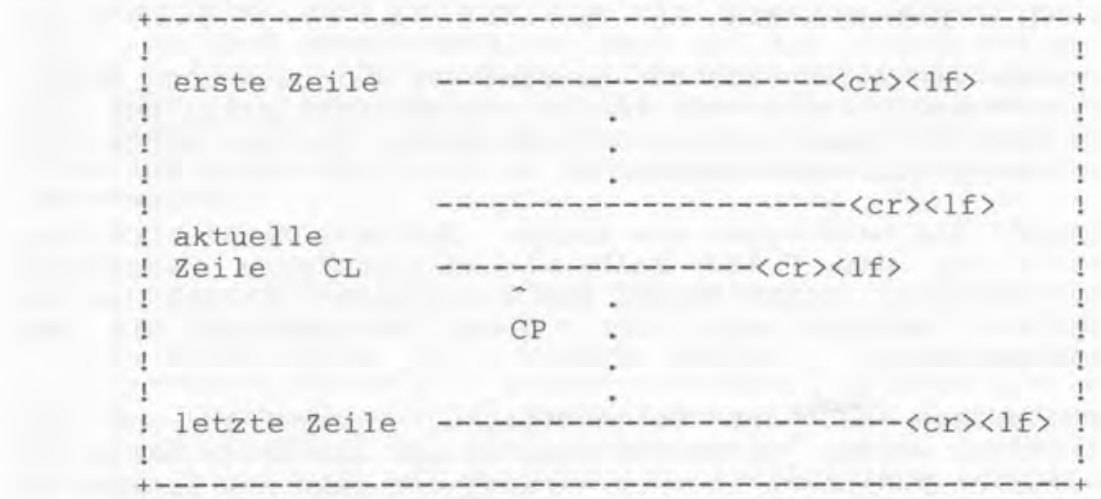
O Zurücksetzen auf die Originaldatei. Der Puffer und die Hilfsdatei werden gelöscht, und SP auf die erste Zeile der Quelldatei positioniert. Die Wirkung der seit dem letzten H-Befehl gemachten Edier-Befehle wird aufgehoben.

Q Beenden des ED ohne Änderungen an der Datei. Zurück zur Kommandoebene.

Es gibt noch ein paar Sonderfälle zu betrachten. Falls der Parameter n in einem Befehl weggelassen wird, erhält er den Wert 1. Beispielsweise schreibt der Befehl "W" eine Zeile auf die Hilfsdatei.
Wird anstelle von n das Zeichen "#" eingegeben, erhält n den größten erlaubten Wert 65535. Falls eine Quelldatei ganz in den Puffer passt, was bei nicht zu großen Dateien der Fall ist, kann mit "#A" die ganze Datei zu Beginn des Edierens in den Puffer gebracht werden und nach der Bearbeitung mit "#W" auf die Hilfsdatei geschrieben werden.
Zur bequemeren Bearbeitung werden noch zwei Sonderformen der Befehle A und W angeboten. Mit "0A" werden so viele Zeilen in den Puffer eingelesen, bis er halbvoll ist. "0W" schreibt soviele Zeilen in die Hilfsdatei, bis der Puffer halbleer ist.
Ist der Puffer voll, erfolgt eine Fehlermeldung des ED. Es kann dann jeder ED-Befehl gegeben werden, der den Puffer nicht vergrößert (z.B.: W). Falls eine Zeile wegen Pufferüberlaufs nicht ganz in den Puffer passt, wird die Zeile beim nächsten A-Befehl vollständig eingelesen.

2.1.3 Speicher- und Pufferorganisation

Der Puffer kann als eine Sequenz von Textzeilen (die mit A in den SPeicher gebracht wurden) betrachtet werden. Zum Puffer gehört ein (imaginärer) Textzeiger CP (character pointer). Dieser wandert abhängig von den Benutzerbefehlen durch den Puffer. Der Puffer läßt sich durch das folgende Diagramm veranschaulichen:



Die Striche veranschaulichen Zeilen unbestimmter Länge, die mit Wagenrücklauf (carriage return) "<cr>" und Zeilenvorschub (line feed) "<lf>" abgeschlossen werden. Das Zeichen "^" repräsentiert die Position des imaginären Zeigers CP. CP steht entweder vor dem ersten Zeichen der ersten Zeile, hinter dem letzten Zeichen der letzten Zeile oder zwischen zwei Zeichen! Die aktuelle Zeile CL ist diejenige Zeile, in der sich der CP befindet.

2.1.4 Zeilennummern und Beginn des Edierens

ED erzeugt absolute Zeilennummern, die sich verwenden lassen, um beim Edieren einzelne Zeilen oder Zeilengruppen auszuwählen. Ist ED im Einfügemodus (siehe I-Befehl), werden den Zeilen die Zeilennummern vorangestellt. Sie haben die Form

```
nnnnn:
```

wobei nnnnn eine absolute Zeilennummer im Bereich 1 ... 65535 ist. Ist der Puffer leer oder ist die aktuelle Zeile am Pufferende, erscheint die Nummer als Folge von 5 Leerzeichen.
Der Benutzer kann sich auf eine absolute Zeilennummer beziehen, indem er dem Befehl die Zeilennummer, gefolgt von einem Doppelpunkt, voranstellt. Die Nummer muß das gleiche Format wie die angezeigten Nummern besitzen. Ist das der Fall und existiert die Zeile im Puffer, verschiebt der ED den Zeiger auf diese Zeile.

Der Befehl

```
345:T
```

wird interpretiert als: "verschiebe den Zeiger auf die Zeile 345 und gib sie auf der Konsole aus". Die Zeilennummern werden nur vom Editor erzeugt und nicht in den Dateien gespeichert. Nach dem Löschen oder Hinzufügen eines Textstücks ändern sich die Zeilennummern. Es ist auch möglich, eine absolute Zeilennummer durch Angabe einer positiven oder negativen Distanz von der aktuellen Zeile zu adressierten. Dazu muß der Doppelpunkt vor der Parameterzahl stehen. Der Befehl

```
:400T
```

hat die Wirkung, daß von der aktuellen Zeile alle Zeilen bis einschließlich der Zeile 400 ausgegeben werden. Kombination der beiden Referenzen ergibt

```
345::400T
```

was bewirkt, daß zuerst auf die Zeile 345 positioniert wird und dann die Zeilen 345 bis einschließlich 400 ausgegeben werden.
Die geschilderte Art der Zeilenreferenz kann vor jedem ED-Befehl stehen. Die Zeilennumerierung wird durch den V-Befehl geteuert. Sie kann mit "-V" abgeschaltet werden.
Existiert die im ED-Kommando angegebenen Datei noch nicht, meldet der ED die Kreation einer neuen Datei mit

```
NEW FILE
```

Der Benutzer muß den I-Befehl nehmen, wenn er Text in den Puffer eintragen möchte. Jede der Eingabezeilen muß mit Return (<cr>) beendet werden. Die Eingabe von Control/Z als einziges Zeichen läßt den ED in den Kommandomodus zurückkehren.

2.1.5 Pufferfunktionen

Zu Beginn ist der Puffer leer. Der Benutzer kann entweder Zeilen der Quelldatei mit dem A-Befehl in den Puffer einlesen oder direkt von der Konsole mit dem Befehl

```
I
```

eingeben. Es lassen sich beliebig viele Zeilen eintippen. Jede Zeile wird mit <cr> (Return) beendet, das <lf> wird automatisch hinzugefügt. Die Konsoleingabe wird durch Eintippen des Zeichens Control/Z beendet (dargestellt durch ^Z). Der Zeiger CP steht hinter dem letzten eingegebenen Zeichen. Die Eingabesequenz

```
I<cr>
NOW IS THE<cr>
TIME  FOR<cr>
```

```
ALL GOOD MEN<cr>
^Z
```

hinterläßt den Puffer als

```
NOW IS THE<cr><lf>
TIME FOR<cr><lf>
ALL GOOD MEN<cr><lf>
```

Im allgemeinen akzeptiert der ED die Befehlsbuchstaben in Klein- oder Großschreibung. Ist der Befehl ein Großbuchstabe, werden alle Eingaben, die in Zusammenhang mit dem Befehl stehen zu Großbuchstaben konvertiert. Besonders beim I-Befehl werden alle Eingabezeilen konvertiert. Deshalb wird meist "I" verwendet, um Groß- und Kleinbuchstaben verwenden zu können.
Es gibt eine Zahl von Befehlen, die den Pufferzeiger CP verschieben oder den Text in seiner Umgebung auf der Konsole ausgeben. Bei den folgenden Befehlen, kann für n eine vorzeichenlose Zahl wahlweise angegeben werden. Steht vor dem Befehl -, so kann die Funktion mit n Zeichen oder Zeilen vor dem Zeiger ausgeführt werden. Wie schon beschrieben, wird ein fehlendes N durch den Wert 1 und das Zeichen "#" durch den Wert 65535 ersetzt. Bei fehlendem Parameter "+/-" wird "+" angenommen.

B/-B — stellt den CP an den Pufferanfang, falls "B" angegeben wurde und an das Pufferende, falls "-B angegeben wurde.

nC/-nC — Verschiebt den CP um n Zeichen nach vorne (C)oder rückwärts (-C). <cr><lf> sind zwei Zeichen!

nD/-nD — (delete) Löscht n Zeichen vor oder hinter dem CP.

nK/-nK — (kill) Löscht n Zeilen des Quelltextes vor oder hinter dem CP. Steht der CP nicht am Zeilenanfang, wird nur der entsprechende Teil der aktuellen Zeile gelöscht.

nL/-nL — Der CP wird an den Anfang der aktuellen Zeile verschoben. Ist n=0, passiert nichts weiter. Sonst wird der CP n Zeilen vorwärts oder rückwärts bewegt und bleibt dort am Zeilenanfang stehen. Ist n zu groß bleibt der CP am Anfag oder Ende des Puffers stehen.

nT/-nT — Für n=0 wird die aktuelle Zeile vom Anfang bis zum CP ausgegeben. Für n=1 wird die aktuelle Zeile ab dem CP bis zum Ende ausgegeben. Ist n>1 werden n Zeilen vorher bzw. die nachfolgenden n Zeilen ausgegeben. Um lange Ausgaben abzubrechen, muß nur eine beliebige Taste gedrückt werden.

n/-n — Die gleiche Wirkung wie nLT/-nLT. Es wird verschoben und eine Zeile angezeigt.

2.1.6 Befehlsfolgen

Es kann jede beliebige Zahl von ED-Befehlen hintereinander fortlaufend eingetippt werden (bis zum Fassungsvermögen des Konsolpuffers). Sie werden erst ausgeführt, wenn die Return-Taste gedrückt wurde. Der Bediener kann also die Zeilenkorrekturfunktionen des CP/M (siehe auch 1.5) verwenden:

Control/C	Warmstart, falls es am Zeilenanfang eingegeben wird.
Control/E	Physikalisches Zeilenende. Es wird ein Return ausgegeben, aber die Zeile nicht abgeschlossen.
Control/H	Es wird ein Zeichen zurückgegangen. (Backspace)
Control/J	Abschicken der Zeile (=> Linefeed).
Control/M	Abschicken der Zeile (=> Return).
Control/R	Die Zeile wird nocheinmal bereinigt ausgegeben (nach Korrekturen auf einer druckenden Konsole).
Control/U	Die gesamte Eingabezeile wird gelöscht.
Control/X	Die gesamte Eingabezeile wird gelöscht.
Control/Z	Ende der Konsoleingabe (wird von PIP und ED verwendet).
RUBOUT/DEL	Löschen des zuletzt eingegebenen Zeichens.

Angenommen, im Puffer stehen die Zeilen des vorhergehenden Abschnitts und der CP steht hinter dem letzten Zeichen im Puffer. Die folgenden Befehlssequenzen produzieren dann das rechts erläuterte Ergebnis. Verwenden Sie klein geschriebene Befehlsnamen um die automatische Konvertierung zu Großbuchstaben zu verhindern ("^" stellt den Pufferzeiger dar).

Befehlsfolge	Wirkung	Resultat im Puffer
1. B2T	Cp wird auf den Pufferanfang gesetzt und die nächsten 2 Zeilen ausgegeben: 'NOW IS THE TIME FOR'	NOW IS THE<cr><lf> ^ THE TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>

2. 5C0T	CP wird 5 Zeichen verschoben und die Zeile ab CP ausgegeben: 'NOW I'	NOW I S THE<cr><lf> THE TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
3. 2L-T	CP wird um 2 Zeilen nach unten geschoben und die vorhergehende Zeile ausgegeben: 'TIME FOR'	NOW IS THE<cr><lf> THE TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
4. -L#K	Cp wird eine Zeile hochgeschoben und die folgenden 65535 gelöscht.	NOW IS THE<cr><lf>
5. I TIME TO INSERT ^Z	Text einfügen. Es werden automatisch Großbuchstaben verwendet.	NOW IS THE<cr><lf> TIME TO<cr><lf> INSERT<cr><lf>
6. -2L#T	CP wird 2 Zeilen hochgeschoben. Die vorherigen 65535 Zeilen ausgegeben: 'NOW IS THE'	NOW IS THE<cr><lf> TIME TO<cr><lf> INSERT<cr><lf>
7. <cr>	Eine Zeile runter und eine Zeile ausgeben: 'INSERT'	NOW IS THE<cr><lf> TIME TO<cr><lf> INSERT<cr><lf>

2.1.7 Suchen und Ändern im Text

ED besitzt einen Befehl um Strings (Zeichenketten) im Puffer zu finden. Der Befehl hat die Form

nFs <cr>

oder

nFs ^Z

wobei s der gesuchte String ist. Er wird von <cr> oder Control/Z (^Z) abgeschlossen. ED frägt an der aktuellen Position des CP an im Text nach dem String s zu suchen. Es wird dabei nach dem n-ten Auftreten von s gesucht. Ist die Suche erfolgreich, wird CP genau hinter den gefundenen, mit s übereinstimmenden, String gesetzt.

Tritt der Suchstring nicht n-mal auf, bleibt CP auf seiner ursprünglichen Position. Im Suchstring kann das Zeichen Control/L als Repräsentant für das Zeilenende <cr><lf> gesetzt werden. Ein Beispiel für den F-Befehl:

Befehlsfolge	Wirkung	Resultierender Puffer
1. B#T	CP auf Pufferanfang und Ausgabe des gesamten Puffers.	NOW IS THE<cr><lf> ^ TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
2. FS T	Suche den String 'S T'.	NOW IS T HE<cr><lf> ^ TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
3. FI^Z0TT	Suche das nächste "I" und gib bis CP aus: 'TI' Dann gib den Rest der Zeile aus: 'ME FOR'	NOW IS THE<cr><lf> TI ME FOR<cr><lf> ^ ALL GOOD MEN<cr><lf>

Um Textänderungen in Zusammenhang mit den F-Befehl durchzuführen ist eine Kurzform des I-Befehls erlaubt. Diese Form ist

```
I s ^Z
```

oder

```
I s <cr>
```

wobei s der einzufügende String ist. Wird der String mit Control/Z abgeschlossen, dann wird er direkt hinter CP eingefügt und CP danach hinter den String gesetzt. Wird der String mit <cr> abgeschlossen, wird zusätzlich ein Zeilenende hinter dem String eingefügt. Beispiele:

Befehlsfolge	Wirkung	Resultierender Puffer
1. BITHIS IS^Z	Einfügen von 'THIS IS' an Pufferanfang	THIS IS NOW THE <cr><lf> ^ TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
2. FTIME^Z-4D IPLACE^Z	Suche den String 'TIME', lösche 4 Zeichen vor CP, füge 'PLACE' ein.	THIS IS NOW THE<cr><lf> SPACE FOR<cr><lf> ^ ALL GOOD MEN<cr><lf>

3. 3FO^Z-3D5D ICHANGES^Z	Finde das dritte Auftreten von 'O' (das 2. in GOOD) lösche 3 Zeichen davor und 5 danach und füge 'CHANGES' ein.	THIS IS NOW THE<cr><lf> PLACE FOR<cr><lf> ALL CHANGES <cr><lf> ^
4. -8CISOURCE<cr>	Gehe 8 Zeichen zurück und füge die Zeile 'SOURCE' ein	THIS IS NOW THE<cr><lf> PLACE FOR<cr><lf> ALL SOURCE<cr><lf> CHANGES<cr><lf> ^

Für einfache Stringänderung stellt ED auch einen Befehl zur Verfügung, der den F- und den I-Befehl kombiniert. Er hat die Gestalt

```
nS s1 ^Z s2 <cr>
```

oder

```
nS s1 ^Z s2 ^Z
```

was dieselbe Wirkung hat wie die n-malige Ausführung von

```
F s1 ^Z -kDIs2 <cr>
```

oder

```
F s1 ^Z -kDIs2 ^Z
```

k ist dabei die Länge des Strings s1. ED beginnt ab der CP-Position nach s1 zu suchen und ersetzt ihn dann durch s2. Diese Substitution wird n-mal ausgeführt (oder bis das Pufferende erreicht ist).
ED stellt einen weiteren, dem F-Befehl ähnlichen, Befehl zur Verfügung der beim Suchen Zeichen oder Zeilen einfügt. Er lautet

```
n N s <cr>
```

oder

```
n N s ^Z
```

Es wird die gesamte Quelldatei nach dem n-ten Auftreten des Strings s durchsucht (der F-Befehl findet s ja nur, wenn sich s im Puffer befindet). Befindet sich s im Puffer, ist die Wirkung dieselbe wie beim F-Kommando. Befindet sich s nicht im Puffer, wird vom N-Befehl der gesamte Pufferinhalt in die Hilfsdatei geschrieben (automatisches #W). Danach werden Eingabezeilen gelesen, bis der Puffer mindestens halbvoll ist oder die Quelldatei vollständig gelesen wurde. Die Suche wird auf diese Weise fortgesetzt, bis das n-te Auftreten von s gefunden wurde oder die Quelldatei vollständig in die Hilfsdatei übertragen wurde.
Schließlich gibt es noch den "Suchen-und-Ersetzen"-Befehl (juxtaposition command), der folgende Form besitzt:

```
nJ s1 ^Z s2 ^Z s3 <cr>
```

oder

```
nJ s1 ^Z s2 ^Z s3 ^Z
```

Bei diesem Befehl wird die folgende Operation n-mal durchgeführt: Suche ab dem CP den String s1. Falls er gefunden wurde, füge den String s2 ein und setze CP dahinter. Dann lösche vom CP bis zu (aber nicht einschließlich) String s3 alle Zeichen. Wird s3 nicht gefunden, erfolgt keine Löschung. Beispiel:

```
Ausgangszeile:    NOW IS THE TIME<cr><lf>

Befehl:           JW ^ZWHAT^Z^L<cr>          (^L ==> <cr><lf>)

Ergebnis:         NOW WHAT <cr><lf>
```

Vergessen Sie nicht, daß mit Control/L (^l) die Zeichenfolge <cr><lf> in Such- und Ersetzungsstrings eingegeben wird.
Bei den Befehlen F, S, N und J können maximal 100 Zeichen angegeben werden.

2.1.8 Quellbibliotheken

Mit dem R-Befehl können Quellbibliotheken in den Ediervorgang mit einbezogen werden. Die Form des Befehls ist

```
R <cr>
```

ED liest die Datei X$$$$$$$.LIB ein und speichert ihren Inhalt ab CP im Puffer. Der R-Befehl ähnelt also in gewisser Weise dem I-Befehl.

Mit dem Befehl X (Xfer = Transfer) kann der Text auch blockweise abgespeichert werden. Der Befehl

```
nX
```

speichert die nächsten n Zeilen auf einer Hilfdatei namens

```
X$$$$$$.LIB
```

Diese Datei existiert nur während des Ediervorgangs. Der Benutzer kann den CP beliebig im Puffer positionieren und Zeilen auf die Hilfsdatei übertragen, wobei die Zeilen in ihr akkumuliert werden. Die Hilfsdatei läßt sich mit dem Befehl

```
R
```

wieder einlesen. Die ganze transferierte Zeilenmenge wird so wieder in den Puffer geholt. Zu bemerken ist, daß mit X die transferierten Zeilen nicht gelöscht werden, sondern nur eine Kopie auf Datei angelegt wird (obwohl man nach dem X-Befehl die Zeilen natürlich mit K löschen kann). Ebensowenig wird nach

Ausführung des R-Befehls die Hilfsdatei gelöscht. Die mit X gespeicherten Zeilen können also beliebig oft in den Puffer geholt werden. Erst das Kommando

```
0X
```

löscht die Transferdatei X$$$$$$.LIB. Diese Datei wird auch bei normaler Beendigung des ED mit den Befehlen E oder Q gelöscht. Bricht man den ED mit Control/C ab, ist sie noch vorhanden, aber meist leer. Ein Aufruf von ED löscht die Datei auf jeden Fall.

2.1.9 Wiederholung von Befehlen

Mit dem Makro-Befehl M kann der Benutzer ED-Befehle zusammenfassen, sodaß sie wiederholt ausgeführt werden können. Der M-Befehl hat die Form

```
nM cs <cr>
```

oder

```
nM cs ^Z
```

wobei cs eine Folge von ED-Befehlen darstellt. In cs darf allerdings kein anderer M-Befehl vorkommen. Die Befehle werden n-mal abgearbeitet, wenn n>1 ist. Ist n=1 oder n=0 werden die Befehle solange abgearbeitet bis eine Fehlersituation auftritt (z.B. das Erreichen des Pufferendes bei F). Als Beispiel für den M-Befehl ändert das folgende Makro alle autretenden Strings 'GAMMA' nach 'DELTA'. Jede geänderte Zeile wird auf der Konsole ausgegeben:

```
MFGAMMA^z-5DIDELTA^z0TT<cr>
```

oder auch:

```
MSGAMMA^zDELTA^z0TT<cr>
```

2.2 ED Fehlermeldungen

Bei Fehlern meldet sich der ED mit "BREAK x AT C", wobei x einer der folgenden Fehlerindikatoren ist:

- `?` ED kennt den Befehl nicht
- `>` Der Puffer ist voll (die Befehle D, K, N, S oder W entfernen Zeichen aus dem Puffer) oder der String bei den Befehlen F, N oder S ist zu lang.
- `#` Das Kommando konnte nicht so oft wie verlangt ausgeführt werden (z.B. bei F).
- `O` Die angegebene LIB-Datei im R-Kommando kann nicht eröffnet werden.

Tritt ein Diskettenfehler auf, meldet sich ED mit einer

Fehlermeldung.
Der Bediener kann die Fehlermeldung ignorieren, indem er Return drückt (in diesem Fall sollten die Daten im Puffer auf Richtigkeit überprüft werden) oder er kann das System durch Drücken von Control/C zurücksetzen. In diesem Fall muß die BAK-Datei wiederverwendet werden. Zuerst wird mit

```
TYPE name.BAK
```

die Datei angesehen, ob sie auch den gewünschten Text enthält. Dann wird die fehlerhafte neue Datei gelöscht:

```
ERA name.typ
```

Schließlich bekommt die BAK-Datei den richtigen Namen:

```
REN name.typ=name.BAK
```

Es ist nun die vorhergehende Version der Datei wieder vorhanden und das Edieren kann wieder aufgenommen werden.
ED beachtet die Datei-Attribute. Will der Benutzer eine schreibgeschützte Datei edieren, erhält er die Meldung

```
** FILE IS READ ONLY **
```

Die Datei kann zwar geladen und angesehen, aber nicht geändert werden. Normalerweise wird der Benutzer den Ediervorgang beenden, mit dem STAT-Kommando das Attribut auf R/W ändern und dann nochmals den ED starten. Hat die Datei das SYS-Attribut, meldet der ED

```
'SYSTEM' FILE NOT ACCESSIBLE
```

(Systemdatei nicht zugreifbar) und bricht ab. Auch hier muß das Attribut geändert werden.

Wegen möglicherweise verheerender Tippfehler können einzelne Befehle nicht in Befehlsfolgen auftreten:

```
E, H, O, Q
```

Sie müssen einzeln getippt werden.
Die Befehle I, J, M, N, R und S sollten als i, j, m, n, r und s getippt werden, falls Groß- und Kleinbuchstaben im Text gemischt werden sollen. Sonst werden alle Buchstaben zu Großbuchstaben konvertiert.
Wird ein Befehl in Großbuchstaben gegeben, wird auch der Text zu Großbuchstaben und umgekehrt.

3 CP/M 68K Assembler und Hilfskommandos

3.1 Assembler AS68

Der Assembler erzeugt aus einer Datei mit einem Assembler-Quellprogramm eine Datei mit einem verschieblichen (relokativen) Objektprogramm und auf Wunsch ein Listing. Der Assembler verarbeitet sämtliche Mnemonics wie sie in den Datenblättern von Motorola angegeben sind. Zu empfehlen ist auch das Buch "Die 68000 Prozessorfamilie" vom Tewi-Verlag.

Der Assembler wird durch das Kommando

```
AS68 <Optionen> quelldatei
```

aufgerufen. Will man das Listing nicht auf die Konsole sondern in eine Datei schreiben, muß noch der Zusatz '>listingdatei' angegeben werden. Beispiele:

```
AS68 -I TEST

AS68 TEST>LISTING
```

Wenn sich auf Ihrer CP/M-Diskette noch keine Datei namens AS68SYM.DAT befindet, müssen Sie diese durch das Kommando

```
AS68 -I AS68INIT
```

einmal erzeugen. Die Optionen im AS68-Kommando werden je nach Bedarf zwischen Kommando und Quelldatei gesetzt. Es gibt folgende Optionen:

```
AS68 -F d: -P -S d: -U -L -N -I -O objektdatei QUELLE>LISTING
```

Ihre Bedeutung ist:

- -F Angabe des Laufwerks für temporäre Dateien
- -I Initialisierung des Assemblers (siehe oben)
- -P Erzeuge ein Listing
- -S Angabe des Laufwerks für die Quelldatei
- -U Veranlasst, daß undefinierte Symbole als global angesehen werden
- -L Sorgt dafür, daß alle Adressen als Langwort generiert werden.
- -N Unterdrückt die Optimierung von Vorwärts-Sprüngen

Pseudo-Operationen:

comm marke,ausdruck
Definiert Common-Bereich, der von mehreren Programm-Modulen gemeinsam benutzt werden kann. Der Linker setzt alle Common-Bereiche mit gleichem Namen auf dieselbe Adresse. Der

Ausdruck legt die Länge dieses Bereichs fest.

data
Dieser Befehl schaltet von Code-Segment auf Daten-Segment um.

bss
Dieser Befehl instruiert den Assembler, auf block storage segment (bss), umzuschalten. Es können jedoch Marken im bss definiert und Speicher belegt (ds) werden.

dc operand, operand, ...
Definieren von konstanten Datenbereichen (Zahlen, Ausdrücke, Bytes). z.B:
```
DC.B 'Guten Tag',$0D,$0A,'$'
DC.W 1234,5678
DC.L 12345678, 87654321
```

ds operand
Reservieren von Speicher. Der Operand gibt an, wieviele Einheiten zu reservieren sind. DS.B reserviert Bytes, DS.W Worte und DS.L Langworte.

end
Programmende

endc
Ende des Codes bei bedingter Assemblierung (conditionally assembly).

equ ausdruck
Gleichsetzen des Wertes einer Marke mit einem Ausdruck.

even
erzwingt die Fortsetzung der Codeerzeugung auf einer geraden Adresse.

globl marke, marke,...
xdef marke, marke,...
xref marke, marke,...
External-Deklaration von Marken. Diese Marken können von anderen Programmmodulen aus zugegriffen werden. Der Linker setzt dann den richtigen Wert ein.

ifeq ausdruck
ifne ausdruck
ifle ausdruck
iflt ausdruck
ifge ausdruck
ifgt ausdruck
Bedingte Assemblierung. Wenn die Auswertung des Ausdrucks den Wert WAHR ergibt, werden die folgenden Assemblerzeilen bis zum nächsten ENDC ausgeführt, sonst nicht.

```
     ifeq      = 0          iflt      < 0
     ifne      <> 0         ifge      >= 0
     ifle      <= 0         ifgt      > 0
ifc 'string1','string2'
ifnc 'string1','string2'
     Bedingte  Assemblierung.  Es wird verglichen,ob  die  beiden
     Strings gleich bzw ungleich sind.

offset ausdruck
     Erzeugt einen Pseudo-Speicherbereich durch die definition
     einer Offset-Tabelle. Die Tabelle beginnt bei der durch den
     Ausdruck spezifizierten Adresse. Sie wird jedoch nicht an
     den Linker weitergegeben.  Es sind nur EQU- und  REG-Befehle
     möglich. Offset wird durch BSS, DATA, END, SECTION oder TEXT
     beendet.

org ausdruck
     Festlegen der Startadresse.

page
     Seitenvorschub im Listing.

reg registerliste
     Legt eine  Registermaske für den MOVEM-Befehl. Die
     Registerliste entspricht jener des MOVEM-Befehls.

section number
     Es wird ein 'Base-segment' mit der entsprechenden Nummer
     definiert. Es sind Nummern von 0 bis 15 möglich, wobei 14
     den Datenbereich und 15 den bss-Bereich markiert. Alle
     anderen Bereiche definieren Textbereiche.

text
     Wechsel zum Text-Segment.


Die Assembler-Befehle (Mnemonics) entsprechen dem Motorola-Stan-
dard:

abcd      add decimal with extend
add       add
and       logicla and
asl       arithmetic shift left
asr       arithmetic shift right

bcc       branch conditionally , cc = condition
bchg      bit test and change
bclr      bit test and clear
bra       branch
bset      branch test and set
bsr       branch to subroutine
btst      bit test

chk       check register at bounds
clr       clear operand
```

```
cmp       compare

dbcc      test condition, decrement and branch, cc = condition
divs      signed divide
divu      unsigned divide

eor       exclusive or
exg       exchange registers
ext       sign extend

jmp       jump
jsr       jump subroutine

lea       load effective address
link      link stack
lsl       logical shift left
lsr       logical shift right

move      move
movem     move multiple registers
movep     move peripheral data
muls      signed multiply
mulu      unsigned multiply

nbcd      negate decimal with extend
neg       negate
nop       no operation
no        ones complement

or        logical or

pea       push effective address

reset     reset external devices
rol       rotate left without extend
ror       rotate right without extend
roxl      rotate left with extend
roxr      rotate right with extend
rte       return from exception
rtr       return and restore
rts       return from subroutine

sbcd      subtract decimal with extend
scc       set on condition, cc = condition
stop      stop
sub       subtract
swap      swap data register halves

tas       test and set operand
trap      trap
trapv     trap on overflow
tst       test bit

unlk      unlink stack
```

3.2 Linker

Der Linker kombiniert mehrere relokative Objektprogramme (auf Dateien mit dem Namenszusatz .REL) zu einen ablauffähigen Programm (.68K). es werden alle externreferenze abgesättigt und der Code entsprechend zusammengebunden. Das Kommandoformat ist:

```
LO68 <Optionen> objektdatei <objektdatei>
```

Wie beim Assembler können die Meldungen des Linkers auf eine Datei umgeleitet werden (z.B.: LO68 TEST>LISTING). Es können mehrere Objektdateien zusammengebunden werden. Die Optionen sind:

```
LO68 -F d: -R -S -I -Uname -O datei -X -Zadresse -Dadresse -Badresse
```

Die Optionen haben die Bedeutung:

-F	Laufwerk für Zwischendateien.
-R	Das erzeugte Programm wird relozierbar.
-S	Speicher sparen, Die Ausgabe wird gekürzt und die relocation-bits entfernt.
-I	Unterdrückt Fehlermeldungen bei 16-Bit-Adressüberlauf.
-U	Einbinden des angegebenen Bibliotheksmoduls.
-O	Definieren einer Objektdatei (Voreinst: C.OUT).
-X	Symboltabelle enthält alle Symbole, die nicht mit L beginnen. Sonst werden nur globale Symbole aufgenommen.
-T	Startadresse des Programm-Textsegments.
-Z	wie -T
-D	Startaddresse für Datensegment. (T,Z und D sind wichtig für Programme, die im EPROM gespeichert werden sollen).
-B	Startadressse für bss.

3.3 Archiv

Das Archiv-Programm verwaltet eine Programm-Bibliothek. Es können Programmmodule hinzugefügt, gelöscht oder herausgeholt werden und der Inhalt einer Bibliothek aufgelistet werden. Das Kommando hat die Form:

```
AR68 <Optionen><AV><Fd:><OPMOD>ARCHIVE<Liste von Objektdateien>
```

Auch hier kann wie bei Assembler und Linker die Ausgabe auf eine Datei umgeleitet werden (>Listing). Die Kommandoteile in spitzen Klammern werden nur bei Bedarf angegeben. Ihre Bedeutung ist:

AV	Zusatz zu Optionen
F	Laufwerk für Zwischendateien
OPMOD	Spezifiziert einen Objektmodul in der Datei, ab dem neue Module eingefügt werden sollen
ARCHIVE	Archivdatei (Bibliotheksdatei)

Optionen:
Die Optionen müssen in der Reihenfolge DRTWX angegeben werden.

D		Lösche die angegebenen Module
	V	Liste auf, welche Module gelöscht wurden und welche noch in der Bibliothek bleiben.
R		Kreiere eine neue Bibliothek.
	A	Liste auf, in welcher Reihenfolge die Module eingetragen werden.
	V	Liste auf, welche Module ersetzt wurden bzw. neu hinzukommen.
T		Drucke ein Inhaltsverzeichnis der Bibliothek.
	V	Zusätzliche Ausgabe der Modulgrößen.
W		Kopiere einen Objektmodul in die durch '>Datei' spezifizierte Ausgabedatei.
X		Kopiere ein oder mehrere Module auf das voreingestellte Laufwerk.
	V	Liste die Module, die kopiert wurden.

3.4 DUMP

Das Programm gibt den Dump einer Datei hexadezimal und in ASCII-Zeichen aus. Das Kommandoformat ist:

```
DUMP Sxxxx DATEI >LISTING
```

Es wird die Datei ab der Adresse ausgegeben, die durch S spezifiziert wurde (die S-Angabe kann entfallen, dann wird die ganze Datei bearbeitet). Die Ausgabe erfolgt wahlweise auf der Konsole oder in die angegebene LISTING-Datei.

3.5 Relokator

Der Relokator macht aus einer relokativen .REL-Datei eine absolut adressierte .68K-Datei. Das Kommandoformat ist:

```
RELOC -Badresse eingabedatei ausgabedatei
```

Mit der Option -B, die auch entfallen darf, wird die Basisadresse des Programms festgelegt. Voreinstellung ist die TPA-Basisadresse + 100H.
Beispiel: RELOC RELOC.REL RELOC.68K

3.6 SIZE68

Mit diesem Kommando kann die Größe von Programmsegmenten in .68K- oder .REL-Dateien ermittelt werden. Das Kommandoformat ist:

```
SIZE68 datei1 datei2 datei3 ... >LISTING
```

3.7 SEND68

Dieses Kommando erzeugt aus einer Programmdatei (.68K) eine Datei im Motorola-S-Format, zum Beispiel für die Übertragung an einen anderen Rechner. Das Kommandoformat ist:

```
SEND68 - eingabedatei ausgabedatei
```

Der Bindestrich ist optional. Ist er vorhanden, wird das bss-Segment nicht beachtet.

Beispiel:

```
* Datei-Kopierprogramm nur mit Laufwerk A
* Version 1.0 Rev.A / Schön / 31.10.85
*
          BRA.B    start
fcbpos:   DC.L     $0              * zeigt auf FCB-Anfang
aktram:   DC.L     $0              * akt. Blockbeginn
freeram:  DC.L     $0              * Beginn freies RAM
 .even
start:    LEA      head,A0         * Startmeldung
          BSR      text
          LEA      fcbpos,A2       * bleibt in A2
          LEA      fcb,A0
          MOVE.L   A0,(A2)
          LEA      ramstart,A0
          MOVE.L   A0,$8(A2)       * freeram setzen
mainloop: MOVE.L   $8(A2),$4(A2) * aktram setzen
srcdisk:  LEA      disk1,A0        * Sourcedisk
          BSR      text
infile:   BSR      clrfcb          * FCB setzen
          LEA      filnam,A0
          BSR      text
getnam:   LEA      filbuf,A0       * Filenamen holen
          MOVEQ.L  #2,D2
get1:     CLR.L    (A0)+           * FCB löschen
          DBRA     D2,get1
          LEA      filbuf,A0
          MOVEQ.L  #$A,D2          * max. Länge -1 -1
get2:     BSR      getchr          * Zeichen von Konsole
          CMPI.B   #$D,D0
          BEQ.B    get2
          CMPI.B   #$8,D0
          BEQ.B    get2
          BSR      cmpchr
          BNE.B    get2
          MOVE.B   D0,(A0)+
          BSR      outchr
get3:     BSR      getchr
          CMPI.B   #$D,D0
          BEQ.B    getend
          CMPI.B   #$8,D0
          BEQ.B    backsp
          CMPI.B   #$2E,D0
          BEQ.B    point
          BSR      cmpchr
          BNE.B    get3
point:    MOVE.B   D0,(A0)+
          BSR      outchr
          DBRA     D2,get3
          BRA.B    transfil
backsp:   CMPI.B #$A,D2
          BEQ.B get3
          SUBQ.L   #1,A0
          BSR      outchr
```

```
            ADDQ.L  #1,D2
            BRA.B    get3
getend:     MOVE.B  #0,(A0)
transfil:   LEA      filbuf,A0
            LEA      fcb,A1
            MOVE.B  #1,(A1)+
            MOVEQ.L #7,D2          * max. 8 Z. für Namen
trans1:     CMPI.B  #0,(A0)
            BEQ.B    namend
            CMPI.B  #$2E,(A0)
            BEQ.B    extens
            MOVE.B   (A0)+,(A1)+
            DBRA     D2,trans1
            CMPI.B  #0,(A0)        *  $00 nach 8 Zeichen?
            BNE.B    checkext      * nein, Ext. prüfen
            MOVEQ.L #2,D2          * ja, daher...
            BRA.B    fillrest      * keine Extension
checkext:   CMPI.B  #$2E,(A0)
            BNE      infile        * Fehleingabe
            BRA.B    extset        * Extension vorhanden
extens:     MOVE.B  #$20,(A1)+
            DBRA     D2,extens     * mit Spaces füllen
extset:     ADDQ.L  #1,A0          * Punkt überspringen
            MOVEQ.L #2,D2          * max. 3 Z. für Ext.
extloop:    CMPI.B  #0,(A0)
            BEQ.B    fillrest
            MOVE.B   (A0)+,(A1)+
            DBRA     D2,extloop
            BRA.B    loadfile
namend:     ADDQ.L  #3,D2          * Rest +3 f. Ext.
fillrest:   MOVE.B  #$20,(A1)+
            DBRA     D2,fillrest
loadfile:   BSR      clear
            BSR      open
            CMPI.B  #$FF,D0        * gefunden?
            BNE.B    okload        * ja
error1:     LEA      err1,A0
            BSR      text
            BSR      quit
            BRA      infile        * neu
okload:     BSR      dma           * DMA-Startadresse ...
            CLR.L    D3            * Blockzähler
load1:      BSR      read
            CMPI.W  #0,D0          * Ende?
            BNE.B    endread       * ja
            BSR      dmaplus       * neue DMA-Adresse
            ADDQ.W  #1,D3          * Zähler +1
            BRA.B    load1         * und weiter
endread:    SUBQ.L  #1,D3
            BSR      close
outfile:    BSR      clear
            LEA      disk2,A0
            BSR      text
            BSR      quit
fcbclr:     MOVEQ.L #5,D2          * hinteren Teil
```

```
            LEA      fcb+$C,A1     * des FCB
            BSR      clrloop       * löschen
exfile:     BSR      open
            CMPI.B   #$FF,D0       * File vorhanden?
            BEQ.B    save          * nein
erafile:    LEA      err3,A0
            BSR      text          * Hinweis
            LEA      eratxt,A0
            BSR      text          * Löschen?
            BSR      getchr
            BSR      cmpchr
            CMPI.B   #$4A,D0
            BNE.B    outfile       * nein
            MOVEQ.L  #$13,D0
            MOVE.L   (A2),D1
            TRAP     #2
            BRA.B    fcbclr        * weiter nach löschen
save:       MOVE.L   $8(A2),$4(A2) * DMA-Startadresse
savefile:   BSR      clear
            BSR      newfile
            CMPI.B   #$FF,D0
            BEQ.B    error2
oksave:     BSR      dma
save1:      BSR      write
            CMPI.B   #0,D0
            BEQ.B    writnext
error2:     LEA      error2,A0
            BSR      text
            BSR      quit
            BRA.B    outfile
writnext:   BSR      dmaplus
            DBRA     D3,save1
endwrite:   BSR      close
            BSR      clear
morecopy:   LEA      moretxt,A0
            BSR      text
            BSR      getchr
            BSR      cmpchr
            CMPI.B   #$4A,D0
            BEQ      mainloop      * nächstes File
endend:     MOVEQ.L  #0,D0         * hier Programmende
            TRAP #2
* Textausgabe
text:       MOVE.L   A0,D1
            MOVEQ.L  #$9,D0
            TRAP     #2
            RTS
* Quittieren nach Textausgabe (Return)
quit:       LEA      quittxt,A0
            BSR      text
quit1:      BSR      getchr
            CMPI.B   #$D,D0
            BNE.B    quit1
            LEA      quitera,A0
            BSR      text
```

```
            RTS
*
clrfcb:     MOVEQ.L #8,D2
            LEA     fcb,A1
clrloop:    CLR.L   (A1)+
            DBRA    D2,clrloop
            RTS
*
getchr:     MOVEQ.L #6,D0
            MOVE.W  #$FF,D1
            TRAP    #2
            ANDI.B  #$7F,D0
            RTS
*
outchr:     MOVE.B  D0,D1
            MOVEQ.L #2,D0
            TRAP    #2
            RTS
*
cmpchr:     CMPI.B  #$30,D0
            BLT     notequ
            CMPI.B  #$61,D0
            BLT     equal
            ANDI.B  #%11011111,D0
equal:      CMP.B   D0,D0
notequ:     RTS
*
clear:      MOVEQ.L #$D,D0
            TRAP    #2
            RTS
*
open:       MOVEQ.L #$F,D0
            MOVE.L  (A2),D1
            TRAP    #2
            RTS
*
close:      MOVEQ.L #$10,D0
            MOVE.L  (A2),D1
            TRAP    #2
            RTS
*
dmaplus     ADDI.L  #$80,$4(A2)
dma         MOVEQ.L #$1A,D0
            MOVE.L  $4(A2),D1
            TRAP    #2
            RTS
*
read:       MOVEQ.L #$14,D0
            MOVE.L  (A2),D1
            TRAP    #2
            RTS
*
write:      MOVEQ.L #$15,D0
            MOVE.L  (A2),D1
            TRAP    #2
```

```
            RTS
*
newfile:    MOVEQ.L #$16,D0
            MOVE.L   (A2),D1
            TRAP    #2
            RTS
*
fcb:        DS.B     $24
filbuf:     DS.B     $C
*
head:       DC.B     $0C,$1B,$59,$2E,$21
            DC.B     'mc-68000-Kopïerprogramm V1.0 Rev.A / sn / 31.10.85
            DC.B     $1B,$59,$32,$23
            DC.B     'Dateien kopieren nur mit Laufwerk A$'
*
disk1:      DC.B     $1B,$59,$25,$28,$1B,$4A
            DC.B     'Quellendiskette einlegen!$'
filnam:     DC.B     $1B,$59,$25,$2A,$1B,$4A
            DC.B     'Dateinamen eingeben: $'
disk2:      DC.B     $1B,$59,$25,$2E,$1B,$4A
            DC.B     'Zieldiskette einlegen!$'
quittxt:    DC.B     $1B,$59,$44,$38
            DC.B     '<Return>$'
quitera:    DC.B     $1B,$59,$20,$38,$1B,$4A,$24
err1:       DC.B     $1B,$59,$25,$30,$1B,$34
            DC.B     'Datei nicht gefunden!'
            DC.B     $1B,$35,$24
err2:       DC.B     $1B,$59,$25,$30,$1B,$34
            DC.B     'Disk oder Directory voll!'
            DC.B     $1B,$35,$24
err3:       DC.B     $1B,$59,$25,$30,$1B,$34
            DC.B     'Datei existiert bereits!'
            DC.B     $1B,$35,$24
moretxt:    DC.B     $1B,$59,$25,$30,$1B,$4A
            DC.B     'Noch eine Datei kopieren? (j/n) $'
eratxt:     DC.B     $1B,$59,$25,$32,$1B,$4A
            DC.B     'Datei löschen? (j/n) $'
 .even
ramstart:   data
end
```

4 Dynamic Debugging Tool (DDT-68K)

4.1 Einführung

Das Programm DDT-68K ermöglicht ein dynamisches, interaktives Testen und "Debugging" (wörtlich: Entwanzen; sinngemäß: Fehler beseitigen) der unter CP/M 68 K erzeugten Programme. Es gibt zwei Aufruf-Formen:

```
DDT-68K

DDT-68K datei.68K
```

wobei "datei" der eindeutige Name einer Datei mit dem zu testenden Programm ist. Die BDOS-Startadresse in 0005H wird geändert und reflektiert die reduzierte Größe der TPA.
Bei der zweiten Aufrufform wird gleich die spezifizierte 68K-Datei in den Speicher geladen. Die Wirkung des Aufrufs ist also dieselbe wie die DDT-68K-Befehlssequenz:

```
DDT-68K
Idatei.68K
R
```

Durch I wird das zu lesende Programm spezifiziert und mit R wird es dann eingelesen. Nach dem Aufruf meldet sich DDT-68K auf der Konsole mit

```
DDT-68K
Copyright 1982, Digital Research
```

Nach dieser Meldung fordert DDT-68K mit "-" den Benutzer auf, Befehle zu geben. Es können verschiedene, aus einem Zeichen bestehende, Befehle eingetippt werden. Jede Befehlszeile wird mit Return abgeschlossen. Die Eingabezeile kann mit den CP/M-Steuerzeichen korrigiert werden.

RUBOUT/DEL	löscht das letzte eingegebene Zeichen.
Control/U	löscht die gesamte Eingabezeile (eine neue Zeile kann nun eingegeben werden).
Control/C	Warmstart des CP/M.

Jede Befehlszeile kann bis zu 32 Zeichen lang sein (es wird als 33. Zeichen automatisch ein Return eingeschoben). Das erste Zeichen definiert den Typ des Befehls:

E	Programm zum Ausführen laden
D	Auflisten des Speichers hexadezimal und als ASCII-Zeichen.

F Füllen des Speichers mit konstantem Wert.

G Starten des Programms mit optionalen Unterbrechungspunkten.

H Hexadezimal rechnen.

I Vorgeben eines Standard-FCB.

L Auflisten des Speichers als Assemblermnemonics.

M Speicherbereich verschieben.

R Einlesen eines Programms zur Bearbeitung.

S Speicher ansehen und ändern.

T (Trace) Verfolgen des Programmablaufs.

U wie T, jedoch ohne Auflisten der Registerinhalte.

V Speicherbelegung des Programms anzeigen

W Speicherinhalt auf Diskette schreiben

X ansehen und ändern der CPU-Register.

In einigen Fällen stehen hinter dem Befehlsbuchstaben 0 bis 3 Parameter, die durch Leerzeichen oder Komma getrennt werden.Alle numerischen Eingaben für den DDT-68K erfolgen hexadezimal. Nach dem Drücken der Return-Taste werden die Befehle dann ausgeführt. Der Benutzer kann jederzeit den DDT-68K mit Control/C verlassen.

4.2 DDT-68K-Befehle

Die einzelnen Befehle werden unten genau beschrieben. Der DDT-68K fordert Befehle mit dem Prompt "-" in Spalte 1 an. In der Beschreibung der Befehle weden die numerischen Parameter durch einzelne Kleinbuchstaben dargestellt, die durch Kommata getrennt sind. Es sind als Parameter nur Hexadezimalzahlen von höchstens vier Stellen Länge erlaubt (längere Zahlen werden rechts abgeschnitten).
Einige Befehle arbeiten mit dem "CPU Status", der ja mit dem getesteten Programm zusammenhängt. Der CPU-Status hält den Inhalt der Register der CPU fest.

4.2.1 D (Display)

Mit dem D-Befehl kann der Benutzer den Speicherinhalt hexadezimal und als ASCII-Zeichenfolge auflisten. Die Form des Befehls ist

D DW DL

oder

```
     Ds         DWs        DLs
oder
     Ds,f       DWs,f      DLs,f
```

Der Zusatz 'W' bei einigen Kommandos dient dem Zugriff auf Wortgrößen (16 Bit), der Zusatz 'L' gestattet den Zugriff auf Langworte (32 Bit).

Im ersten Fall wird der Speicher ab der aktuellen "Displayadresse" 16 weitere Zeilen ausgegeben. Jede angezeigte Zeile hat die Form

```
aaaaaaaa dd dd dd ..... dd dd dd dd dd dd dd dd ccccccc.....ccccccc
```

wobei aaaaaaaa die Speicheradresse des ersten angezeigten Byte ist. dd sind die 16 folgenden Speicherbytes, die mit cccc...ccc noch einmal als ASCII-Zeichen dargestellt werden. Nichtdruckbare Zeichen werden hier als Punkt (.) ausgegeben. Es werden Groß- und Kleinbuchstaben unterschieden. Kann das Konsolgerät nur Großbuchstaben ausgeben, erfolgt eine Konvertierung der Kleinbuchstaben. Bei der zweiten Form des Befehls wird die Startadresse s für die Ausgabe angegeben. Bei der dritten Form des Befehls werden Startadresse s und die Endadresse f des aufzulistenden Speicherbereichs angegeben. Eine kontinuierliche Ausgabe zu je 16 Zeilen kann erzielt werden, indem fortlaufend der Befehl D gegeben wird. Lange Ausgaben können durch Drücken der Return-Taste abgebrochen werden (z.B.:D0,FFFF).

Bei DW werden Worte ausgegeben:

```
aaaaaaaa wwww wwww wwww wwww wwww ..... wwww wwww cccccc....cccccc
```

Bei DL dann Langworte:

```
aaaaaaaa llllllll llllllll llllllll ..... llllllll cccccc...cccccc
```

4.2.2 E (Load for Execution)

Der Befehl hat die Form

```
     Edatei
```

wobei der Dateityp zu ".68K" ergänzt wird.
Der E-Befehl lädt ein Programm so in den Speicher, daß es nachher mit den Befehlen G, T und U gestartet und getestet werden kann. Der Befehl löscht ein eventuell vorher im Speicher befindliches Programm, es kann also immer nur ein Programm getestet werden. Nach dem Laden gibt der DDT-68K Start und Ende jedes Dateisegments an. Diese Information kann auch mit dem V-Befehl abgerufen werden.

4.2.3 F (Fill)

Der Befehl hat die Form

```
Fs,f,c          FWs,f,c          FLs,f,c
```

und bedeutet: Fülle den Speicherbereich von Startadresse s bis Endadresse f mit dem konstanten Wert c. DDT-68K speichert zuerst den Wert c auf Adresse s. Dann wird s erhöht und gegen f getestet. Ist s>f, terminiert der Befehl, sonst wird weitergemacht. Das Kommando wird verwendet, um einen Speicherblock konstant vorzubesetzen.

4.2.4 G (Go)

Das bearbeitete Programm wird mit G an einer beliebigen Stelle gestartet. Es können zwei optionale Breakpoints gesetzt werden. Seine Form ist

```
     G
oder
     Gs
oder
     Gs,b1,...,b10
oder
     G,b1,...,b10
```

Beim ersten Aufruf wird das Programm ab dem aktuellen Programmzählerstand mit dem aktuellen CPU-Status gestartet. Es sind keine Breakpoints gesetzt. Beim zweiten Aufruf wird die Startadresse der Programmausführung s spezifiziert; ansonsten verläuft alles wie beim ersten Aufruf.
Beim dritten Aufruf werden zusätzlich zur Startadresse s 1 bis 10 Breakpoints spezifiziert. Die Programmausführung hält an, wenn die Adresse eines Breakpoints erreicht wird. Die Kontrolle geht dann an den DDT-68K zurück Der Befehl auf der Breakpoint-Adresse wird nicht mehr ausgeführt.
Gibt man keine Startadresse s an, wird das Programm ab dem aktuellen Programmzähler gestartet.

4.2.5 H (Hex Math)

Der Befehl hat die Form

```
     Ha,b
```

wobei a und b Hexadezimalzahlen sind. Der DDT-68K gibt die Summe und die Differenz beider Werte als 32-Bit-Zahlen aus.

4.2.6 I (Input)

Der I-Befehl erlaubt dem Bediener, in den voreingestellten FCB bei Adresse 005CH einen Dateinamen einzutragen. Der FCB kann vom Testprogramm so benutzt werden, als ob das Programm die Dateiangaben vom CCP erhalten hätte. Der FCB wird aber auch vom DDT-68K verwendet, um weitere Dateien einzulesen. Die Form des Befehls ist

```
Idateiname
```

Mit dem E-Befehl kann die Datei mit dem Programm dann eingelesen werden.

4.2.7 L (List)

Mit diesem Befehl kann ein bestimmter Speicherbereich disassembliert werden (d.h. in Form von Assemblermnemonics aufgelistet werden). Seine Form ist

```
     L
oder
     Ls
oder
     Ls,f
```

Die erste Form listet 12 Zeilen ab der aktuellen List-Adresse. Die zweite Form gibt auch 12 Zeilen aus, die List-Adresse wird jedoch vorher auf s gesetzt. Die dritte Form disassembliert den Speicher von der Startadresse s bis zur Endadresse f. In allen Fällen wird die Listadresse auf den nächsten Befehl gesetzt. So kann durch aufeinanderfolgende L-Befehle der Speicher kontinuierlich disassembliert werden. Nach einem G- oder T-Befehl wird die Listadresse auf den Programmzählerstand gesetzt. Lange Ausgaben können durch Drücken der Return-Taste abgebrochen werden.

4.2.8 M (Move)

Der M-Befehl erlaubt das blockweise Verschieben von Speicherbereichen. Seine Form ist

```
Ms,f,d
```

Der Speicherbereich von der Startadresse s bis zur Endadresse f wird auf die Zieladresse d verschoben.
Dabei wird zuerst ein Byte von s nach d transportiert und danach s und d erhöht. Ist s>f, terminiert der Befehl, sonst wird weitergemacht.

4.2.9 R (Read)

Der R-Befehl wird in Zusammenhang mit dem I-Befehl verwendet, um Dateien von der Diskette einzulesen. Seine Form ist

```
Rdatei
```

Die Datei wird en bloc in den Speicher eingelesen und die Start- und End-Adressen der eingelesenen Programmblöcke werden angezeigt.

4.2.10 S (Set)

Mit dem S-Befehl können Speicherplätze angesehen und geändert werden. Seine Form ist

```
Ss        SWs       SLs
```

wobei s die Adresse ist, ab der der Speicher inspiziert werden soll. DDT-68K antwortet mit der Ausgabe der aktuell bearbeiteten Speicheradresse und dem aktuellen Inhalt der Speicherzelle. Antwortet der Benutzer durch Drücken der Return-Taste, bleibt der Inhalt der Speicherzelle unverändert. Gibt er einen Bytewert ein, wird die Speicherzelle mit diesem Wert gefüllt. In jedem Fall gibt DDT-68K die folgende Speicheradresse mit Inhalt zur Inspektion aus. Der S-Befehl endet, wenn entweder ein Punkt (.) eingegeben wird, oder eine Falscheingabe erfolgte.

4.2.11 T (Trace)

Mit dem T-Befehl kann der Ablauf selektierter Progammteile verfolgt werden (1 bis 65537 Programmschritte). Seine Form ist

```
     T
oder
     Tn
```

Im ersten Fall wird der CPU-Status angezeigt und der nächste Befehl ausgeführt, wonach das Programm anhält und die nächste zu bearbeitende Adresse ausgibt. (Im Prinzip, wie ein Breakpoint nach dem nächsten Befehl). Der CPU-Status kann mit X inspiziert werden.
Die zweite Form des T-Befehls legt fest, daß die nächsten n Instruktionen abgearbeitet werden sollen, bevor ein Break erfolgt. Nach der Ausführung jeder Instruktion wird der CPU-Status aufgelistet. Die Anzeige der CPU-Register entspricht dem Format des X-Befehls. Der Trace kann jederzeit mit einem Breakpoint abgebrochen werden, indem der Benutzer das Zeichen RUBOUT (DEL) eingibt. Die Ablauf-Verfolgung eines Programms wird unterbrochen, wenn ein CP/M-Systemaufruf erfolgt (siehe Kapitel 5) und nach Abarbeitung des Systemaufrufs wieder aufgenommen. So werden Lauf-

zeitprobleme bei E/A-Operationen und beim Diskettenzugriff vermieden, da die Programmausführung im Trace-Modus ca. 500 mal langsamer als normal ist. Denn im Trace-Modus übernimmt der DDT-68K nach jeder abgearbeiteten Instruktion die Kontrolle.

4.2.12 U (Untrace)

Der Befehl U entspricht dem T-Befehl, jedoch wird hier kein CPU-Status auf der Konsole ausgegeben. Seine Form ist

```
     U
oder
     Un
```

wobei n auch hier angibt, wieviele Programmschritte unter Kontrolle des DDT-68K ablaufen sollen. Er wird hauptsächlich verwendet, um die Kontrolle über das zu testende Programm in jedem Fall zu behalten. Alle Einschränkungen und Eigenschaften des T-Befehls treffen auch auf den U-Befehl zu.

4.2.13 V (Value)

Der Befehl hat die Form

```
     V
```

Es wird die Speicherbelegung des durch E oder R eingelesenen Programms angezeigt. Es wird die Startadresse und Länge jedes Programmsegments angezeigt.

Zum Beispiel:

```
Text Base=00000500  data base=00000B72  bss base=00003FDA
text length=00000672  data length=00003468 bss length=0000A1B0
base page address=00000400  initial stack pointer=000066D4
```

4.2.14 W (Write)

Der Befehl hat die Form

```
     Wdatei
```

oder

```
     Wdatei,s,f
```

Der Speicherinhalt wird auf die angegebene Datei geschrieben. Bei der ersten Form des Befehls nimmt der DDT-68K die Startadresse (s) und die Endadresse (f) des letzten R-Befehls. Ist die Datei schon vorhanden, wird sie vom DDT-68K überschrieben.

4.2.15 X (Examine)

Mit dem X-Befehl können die Register und der Status der CPU angesehen und gegebenenfalls geändert werden. Seine Form ist

```
    X
oder
    Xr
```

r ist dabei ein Registername der CPU 68000:

```
    D0 ... D7
    A0 ... A7
    PC
    USP
    SSP
```

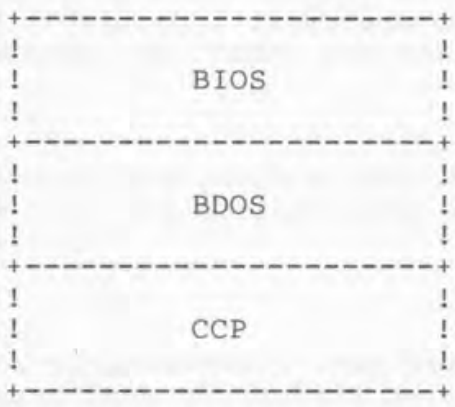
In der ersten Form des Kommandos (X alleine) werden der CPU-Status und die Registerinhalte ausgegeben.
Die zweite Form des Befehls erlaubt die gezielte Inspektion oder Änderung eines CPU-Registers oder einer Statusflagge. Der Inhalt des Registers wird auf der Konsole angezeigt. Antwortet der Benutzer mit dem Drücken der Return-Taste, bleibt das Register oder die Flagge unverändert. Andernfalls kann der Benutzer durch Eingabe eines Wertes den Inhalt des entsprechenden Registers ändern.

5 CP/M 68K Systemschnittstelle

5.1 Einführung

CP/M wird in drei logische Einheiten gegliedert: BIOS ("Basic I/O System" = Ein-/Ausgabekern), BDOS ("Basic Disk Operating System" = Diskettentreiberprogramm), CCP ("Console Command Interpreter" = Kommandointerpreter). Das BIOS ist ein Programmmodul, das der jeweiligen Hardware angepasst werden muß und alle Routinen für die Bedienung der E/A-Schnittstellen enthält. Obwohl ein Standard-BIOS von Digital Research vertrieben wird, werden besondere Anweisungen für das globale Ändern des BIOS mitgegeben um die Anpassung an jede beliebige Hardware zu ermöglichen (siehe Kapitel 6). Das BIOS und das BDOS werden zu einem logischen Modul vereinigt und mit gemeinsamen Eintrittspunkten als BDOS bezeichnet. Der CCP ist ein eigenständiges Programm, das eine Benutzerschnittstelle für die BDOS-Routinen und zu den Informationen im Hintergrundspeicher bereitstellt.

Die Speicherorganisation von CP/M sieht so aus:



CP/M 68 K kann an jeder Stelle oberhalb der Exception-Vektoren (0000H ... 03FFH) in den Speicher geladen werden. Auf der FSS-Diskette wird CP/M für zwei Ausbaustufen des Speichers angeboten, 128 KByte RAM und 256 KByte RAM. Der RAM-Bereich muß zusammenhängend sein und bei Adresse 0 beginnen.
Der Benutzer kommuniziert mit dem CCP durch Eingabe von Kommandozeilen nach jedem Prompt. Jede Kommandozeile hat eine der folgenden Erscheinungsweisen:

```
Kommando

Kommando Datei1

Kommando Datei1 Datei2
```

wobei "Kommando" entweder eine der eingebauten Funktionen wie DIR oder TYPE oder der Name eines Programms ist (DDT, ED, PIP, ...). Wenn das Kommando eine der eingebauten Funktionen von CP/M ist,

wird es sofort ausgeführt. Im anderen Fall durchsucht der CCP die augenblicklich angewählte Diskette nach einer Datei namens

Kommando.68K

Ist die Datei vorhanden, nimmt der CCP an, daß es sich um das Speicherabbild eines ausführbaren Programms handelt. Der CCP läd diese Datei von der Diskette in den Speicher.
Das Programm wird dann vom CCP angesprungen und beginnt mit der Ausführung, wobei es die E/A des BDOS verwendet. Das Programm wird vom CCP "aufgerufen", daher kann nach Ablauf einfach zum CCP zurückkehren, indem mit RTS zurückgesprungen wird oder das Programm die BDOS-Funktion 0 aufruft. Das aufgerufene Programm kann die E/A von CP/M einschliesslich des Diskettensystems verwenden um mit Benutzerperipherie oder Hintergrundspeichern Daten auszutauschen. Einen Sonderfall bilden bei CP/M-68K die Dateien vom Typ .REL, die ein Programm enthalten, das im Speicher frei verschieblich ist. Solche programme können auch als Kommando aufgerufen werden, wenn bei der Angabe des Namens der Dateizusatz .REL mit angegeben wird.

5.2 Konventionen für den Systemaufruf

CP/M- Funktionen, die für den Zugriff durch Benutzerprogramme bereitstehen, lassen sich in zwei Gruppen einteilen, einfache Geräte-E/A und Disketten-E/A. Die Gerätetreiber beinhalten:

Lesen Zeichen von Konsole
Schreiben Zeichen nach Konsole
Lesen Zeichen sequentiell von Band
Schreiben Zeichen sequentiell auf Band
Schreiben Zeichen nach List-Device
Holen oder Setzen E/A Status
Ausdrucken Konsolenpuffer
Lesen Konsolenpuffer
Abfragen, ob Konsole bereit

Die BDOS-Operationen, die Disketten-E/A betreiben, sind:

Rücksetzen Diskettensystem
Auswählen Laufwerk
Datei kreieren
Datei eröffnen
Datei schliessen
Durchsuchen Directory (Dateiverzeichnis)
Löschen Datei
Umbenennen Datei
Wahlfrei oder sequentiell lesen
Wahlfrei oder sequentiell schreiben
Abfragen zugreifbarer Diskettenlaufwerke
Abfragen des selektierten Laufwerks
Setzen der DMA-Adresse
Setzen/Rücksetzen der Dateiattribute
Alle BDOS-Funktionen werden aufgerufen, indem die D-Register mit

den passenden Parametern versorgt werden und dann das BDOS mit dem Befehl TRAP #2 angesprungen wird. In den Registern D1 und D0 werden die Ergebnisse zurückgegeben, wobei ein Wert von FFFFH in D0 auf einen Fehler hinweist.

Beispiel: Zeichen auf der Konsole ausgeben

```
CO:  MOVE.W    #2,D0      ; Funktionsnummer in D0
     MOVE.W    #'X',D1    ; Das auszugebende Zeichen in D1
     TRAP      #2         ; BDOS-Aufruf
                          ; Hier geht es dann weiter
```

CP/M implementiert auf jeder Diskette eine Struktur benannter Dateien, wobei die logische Dateiorganisation es ermöglicht, daß jede einzelne Datei eine beliebige Zahl von Blöcken enthalten kann: von einer leeren Datei bis zur vollen Diskettenkapazität. Die einzelnen Laufwerke werden logisch voneinander unterschieden, jedes hat sein eigenes Directory (Dateiverzeichnis) und seine eigenen Dateien. Der Name einer Diskettendatei besteht aus drei Teilen: dem Laufwerknamen (ein Buchstabe), dem Dateinamen, der aus bis zu 8 Zeichen ungleich Blank besteht und dem Dateityp (auch Extension genannt) der aus bis zu 3 Zeichen bestehen kann und auch fehlen darf. Der Dateityp sagt etwas über die Herkunft der Datei aus, wogegen der Dateiname zur Unterscheidung der einzelnen Dateien auf der Diskette dient.
Quelldateien werden als Folge von ASCII-Zeichen betrachtet, wobei jede Zeile von Wagenrücklauf (Carriage Return (CR=0DH)) und Zeilenvorschub (Line Feed (LF=0AH)) abgeschlossen wird. So kann ein CP/M Record von 128 Byte mehrere Zeilen Quelltext enthalten. Das Ende einer ASCII-Datei wird entweder durch das ASCII-Zeichen SUB (1AH = Control-Z) oder durch ein echtes end-of-file, das von der CP/M Leseoperation gemeldet wird, markiert. In Dateien, die Maschinencode enthalten wird das Zeichen SUB jedoch ignoriert, hier wird nur das end-of-file der Leseoperation verwendet.
In den Dateioperationen, die mit Funktionsnummer 15 beginnen, wird normalerweise ein Dateikontrollblock (FCB) adressiert. Programme verwenden oft den Speicherbereich bei 005CH der normalerweise von CP/M für die FCBs verwendet wird, für einfache Dateioperationen. Die Grundeinheit der Dateiinformation ist ein Block (Record) zu 128 Byte, der für alle Dateioperationen verwendet wird. Der FCB-Bereich besteht aus einer Folge von 33 Bytes für sequentiellen Zugriff und 36 Byte für wahlfreien Zugriff. Der FCB auf der voreingestellten Adresse BOOT+005CH kann auch für wahlfreien Zugriff verwendet werden, da beginnend bei BOOT+007DH drei Bytes frei sind. Das Format eines FCB enthält die folgenden Felder:

```
+--+--+--+---+--+--+--+--+--+--+--+--+---+--+--+--+--+--+---
!dr!f1!f2!...!f8!t1!t2!t3!ex!s1!s2!rc!d0!...!dn!cr!r0!r1!r2!
+--+--+--+---+--+--+--+--+--+--+--+--+---+--+--+--+--+--+---
 00 01 02 ... 08 09 10 11 12 13 14 15 16 ... 31 32 33 34 35
```

Die Bedeutung der Felder ist folgende:

dr	Laufwerknummer(0-16) 0 => verwende voreingestelltes Laufwerk 1 => Laufwerk A 2 => Laufwerk B . . 16 => Laufwerk P
f1...f8	Dateiname in Großbuchstaben (bit 8 = 0)
t1...t3	Dateityp in Großbuchstaben (bit 8 = 0) t1',t2' und t3' bezeichnen die höchstwertigen Bits dieser Zeichen mit der Bedeutung: t1' = 1 => Datei schreibgeschützt t2' = 1 => SYS-Datei, wird nicht bei DIR gelistet
ex	augenblickliche Extensionsnummer, wird vom Benutzer normalerweise auf 00 gesetzt, liegt während einer E/A-Operation zwischen 0 und 31.
s1	reserviert für System
s2	reserviert für System (0 für Funktionen 15,17,18,22)
rc	Blockzähler für Extension 'ex', Werte von 0 bis 127
d0...dn	Wird von CP/M gefüllt, reserviert für System
cr	Blocknummer des gerade gelesenen oder geschriebenen Blocks, wird vom Benutzer auf 0 gesetzt.
r0..r2	optionale Blocknummer im Bereich von 0-65535 für wahlfreie E/A mit Überlauf nach r2. r0 und r1 bilden einen 16-bit-Wert mit dem höchstwertigen Byte in r1.

Auf den folgenden Seiten werden nun die Funktionen im Einzelnen erklärt:

```
+-------------------------------------------+
!                                           !
! Funktion  0: System zurücksetzen          !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 00H                       !
!                                           !
+-------------------------------------------+
```

Die Funktion "System zurücksetzen" hat die Rückkehr zum CP/M auf Kommandoebene (CCP) zur Folge. Der CCP initialisiert das Diskettensystem.

```
+-------------------------------------------+
!                                           !
! Funktion  1: Konsoleingabe                !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 01H                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: ASCII-Zeichen             !
!                                           !
+-------------------------------------------+
```

Diese Funktion liest das nächste Zeichen von der Konsole in das Register D0 ein. Darstellbare Zeichen sowie Wagenrücklauf, Zeilenvorschub, und Rückwärtsschritt (Control/H) werden an die Konsole zurückgeschickt (Echo). Tabulatorzeichen (Control/I) bewegen den Cursor auf die nächste Tabulatorposition.Das Anhalten des Scolling (Control/S) wird abgeprüft ebenso das Ein- und Ausschalten des Protokolldrucks (Control/P). Das BDOS kehrt erst zum aufrufenden Programm zurück, wenn ein Zeichen eingetippt wurde. Der Programmablauf wird also angehalten, bis ein Zeichen zur Verfügung steht.

```
+-------------------------------------------+
!                                           !
! Funktion  2: Konsolausgabe                !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 02H                       !
!  Register D1.W: ASCII-Zeichen             !
!                                           !
+-------------------------------------------+
```

Das ASCII-Zeichen in Register D1 wird zur Konsole gesandt. Genau wie bei Funktion 1 werden Tabs expandiert sowie Control/P und Control/S abgefragt.

```
+-----------------------------------------------+
!                                               !
! Funktion  3: Reader Eingabe                   !
!                                               !
! Eintrittsparameter:                           !
!  Register D0.W: 03H                           !
!                                               !
! Resultat:                                     !
!  Register D0.W: ASCII-Zeichen                 !
!                                               !
+-----------------------------------------------+
```

Diese Funktion liest das nächste Zeichen vom logischen Gerät "Reader" (Lochstreifenleser) in das Register D0 (siehe auch Definition des IOBYTE in Kapitel 6). Die Kontrolle wird erst an das aufrufende Programm zurückgegeben, wenn ein Zeichen gelesen wurde.

```
+-----------------------------------------------+
!                                               !
! Funktion  4: Punch Ausgabe                    !
!                                               !
! Eintrittsparameter:                           !
!  Register D0.W: 04H                           !
!  Register D1.W: ASCII-Zeichen                 !
!                                               !
+-----------------------------------------------+
```

Diese Funktion sendet ein Zeichen in Register D1 zum logischen Gerät "Punch" (Lochstreifenstanzer).

```
+-----------------------------------------------+
!                                               !
! Funktion  5: List Ausgabe                     !
!                                               !
! Eintrittsparameter:                           !
!  Register D0.W: 05H                           !
!  Register D1.W: ASCII-Zeichen                 !
!                                               !
+-----------------------------------------------+
```

Diese Funktion sendet ein Zeichen in Register D1 zum logischen Protokoll-Ausgabegerät.

```
+-----------------------------------------+
!                                         !
! Funktion  6: Direkte Konsol-E/A         !
!                                         !
! Eintrittsparameter:                     !
!  Register D0.W: 06H                     !
!  Register D1.W: 0FFH (Eingabe) oder     !
!                 0FEH (Status)  oder     !
!                 ASCII-Zeichen (Ausgabe) !
!                                         !
! Resultat:                               !
!  Register D0.W: Zeichen oder Status     !
!                                         !
+-----------------------------------------+
```

Direkte Konsol-E/A wird von CP/M bereitgestellt um besondere Anwendungen zu unterstützen, die eine Konsol-E/A auf tieferem Niveau benötigen. Im allgemeinen sollte die Verwendung dieser Funktion vermieden werden, da sie keine der CP/M Kontrollfunktionen wie Control/P oder Control/S berücksichtigt. Jedoch sollten Programme, die unter früheren Versionen von CP/M direkte Konsol-E/A durch das BIOS ausführten so geändert werden, dass sie nun diese BDOS-Funktion verwenden. Beim Eintritt in Funktion 6 muss Register D1 entweder 0FFH sein, wodurch eine Eingabeanforderung signalisiert wird, oder 0FEH, wodurch der Status abgefragt wird, oder ein ASCII-Zeichen für die Ausgabe enthalten. Die Funktion 6 darf nicht zusammen mit den anderen Konsolfunktionen verwendet werden.

```
+-----------------------------------------+
!                                         !
! Funktion  7: Holen I/O Byte             !
!                                         !
! Eintrittsparameter:                     !
!  Register D0.W: 07H                     !
!                                         !
! Resultat:                               !
!  Register D0.W: Wert des I/O Byte       !
!                                         !
+-----------------------------------------+
```

Die Funktion 7 liefert den Wert des IOBYTE in Register D0.W.

```
+-----------------------------------------------+
!                                               !
! Funktion  8: Setzen I/O Byte                  !
!                                               !
! Eintrittsparameter:                           !
!  Register D0.W: 08H                           !
!  Register D1.W: Wert de I/O Byte              !
!                                               !
+-----------------------------------------------+
```

Diese Funktion ändert den Wert des IOBYTE auf jenen, der in Register D1 gegeben wurde.

```
+-----------------------------------------------+
!                                               !
! Funktion  9:  Drucke Zeichenkette (String)    !
!                                               !
! Eintrittsparameter:                           !
!  Register D0.W: 09H                           !
!  Register D1.L: Adresse eines Strings         !
!                                               !
+-----------------------------------------------+
```

Diese Funktion sendet eine Zeichenkette, die im Speicher abgelegt ist an die Konsole. Die Adresse des ersten Zeichens steht in D1, das Ende der Zeichenkette wird durch das $-Zeichen signalisiert. Wie in Funktion 2 werden Tabulatorzeichen expandiert und die Kontrollzeichen Control/P und Control/S abgeprüft.

```
+-----------------------------------------------+
!                                               !
! Funktion 10: Lesen Konsolpuffer               !
!                                               !
! Eintrittsparameter:                           !
!  Register D0.W: 0AH                           !
!  Register D1.L: Pufferadresse                 !
!                                               !
! Resultat:                                     !
!    Zeichen von der Konsole im Puffer          !
!                                               !
+-----------------------------------------------+
```

Diese Funktion liest eine Zeile von der Konsole, die bei der Eingabe korrigiert werden kann, und speichert sie in einem Puffer, der von D1 adressiert wird. Die Konsoleingabe wird beendet, wenn entweder der Puffer überläuft oder eines der beiden Zeichen Wagenrücklauf(CR) und Zeilenvorschub(LF) eingegeben wurde. Der Puffer hat die Form:

```
D1: +0 +1 +2 +3 +4 +5 +6 +7 +8 ... +n
   +--+--+--+--+--+--+--+--+--+------------------
   !mx!nc!c1!c2!c3!c4!c5!c6!c7!...!??!
   +--+--+--+--+--+--+--+--+--+------------------
```

wobei mx die Maximalzahl der Zeichen ist, die der Puffer aufnehmen kann (1..255) und nc die Zahl der tatsächlich vom BDOS gelesenen Zeichen ist (wird vom BDOS gesetzt). Falls nc<mx ist, werden die Speicherzellen, die auf das letzte Zeichen folgen (hier durch ?? markiert) unverändert. Während der Eingabe der Zeile werden die folgenden Steuerzeichen akzeptiert:

Rubout/Del	löscht und echot das letzte Zeichen
Control/C	Warmstart des CP/M (falls am Zeilenanfang)
Control/E	Physikalisches Zeilenende
Control/H	Rückwärtsschritt
Control/J	Zeilenvorschub - Ende der Eingabe
Control/M	Wagenrücklauf - Ende der Eingabe
Control/R	Gibt die letzte Eingabe in einer neuen Zeile nocheinmal aus
Control/U	Löscht die ganze Zeile
Control/X	wie Control/U

Anzumerken ist, daß alle Steuerzeichen, die einen Wagenrücklauf zur Folge haben, nur bis zu der Spalte, die auf den Prompt folgt, zurückgehen (in früheren Versionen wurde bis zum Zeilenanfang zurückgegangen). Diese Konvention erleichtert die Eingabe für den Bediener.

```
+-------------------------------------------+
!                                           !
! Funktion 11: Holen Konsolstatus           !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 0BH                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: Konsolstatus              !
!                                           !
+-------------------------------------------+
```

Der Konsolstatus prüft, ob ein Zeichen an der Konsole eingegeben wurde. Falls ein Zeichen ansteht, enthält Register D0 einen Wert ungleich 0, sonst 00H.

```
+-------------------------------------------+
!                                           !
! Funktion 12: Versionsnummer               !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 0CH                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: Versionsnummer            !
!                                           !
+-------------------------------------------+
```

Diese Funktion erlaubt versionsunabhängige Programme. Es wird ein 2-Byte-Wert zurückgegeben.

```
+-------------------------------------------+
!                                           !
! Funktion 13: Disksystem zurücksetzen      !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 0DH                       !
!                                           !
+-------------------------------------------+
```

Diese Funktion ermöglicht das programmgesteuerte Rücksetzen des Disksystems. Alle Disketten werden auf read/write (siehe Funkt. 28 und 29) gesetzt, Laufwerk A wird selektiert.

```
+-------------------------------------------+
!                                           !
! Funktion 14: Laufwerk wählen              !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 0EH                       !
!  Register D1.W: Laufwerknummer            !
!                                           !
+-------------------------------------------+
```

Die Laufwerknummer in Register D1 stellt das für die weiteren Operationen verwendete Laufwerk ein. D1=0 für Laufwerk A, D1=1 für Laufwerk B und so weiter bis D1=15 für Laufwerk P. Das entsprechende Laufwerk wird angewählt und dadurch das Directory dieses Laufwerks bis zum nächsten Kalt- oder Warmstart beziehungsweise zum nächsten Rücksetzen des Disksystems aktiviert. Beim Wechsel der Diskette bei einem angewählten Laufwerk wird dieses normalerweise schreibgeschützt (siehe Funktion 28). Bei allen Dateiangaben ohne explizit angegebenen Laufwerknamen (dr=00 im FCB) wird das voreingestellte Laufwerk verwendet, bei Angabe des Laufwerknamens (A-P; dr = 1...16 im FCB) wird das entsprechende Laufwerk direkt angewählt.

```
+-------------------------------------------+
!                                           !
! Funktion 15: Datei eröffnen               !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 0FH                       !
!  Register D1.L: FCB-Adresse               !
!                                           !
! Resultat:                                 !
!  Register D0.W: Directory-Code            !
!                                           !
+-------------------------------------------+
```

Diese Funktion wird verwendet um eine Datei zu aktivieren, die bereits auf der Diskette existiert. Das BDOS durchsucht das Directory des angewählten Laufwerks nach dem Dateinamen, der in den Positionen 1 bis 14 des FCBs steht, welcher durch die Register D1 adressiert wird (das Byte s1 wird automatisch null-gesetzt). Das ASCII-Zeichen '?' (3FH) steht für jedes beliebige Zeichen beim Directory-Vergleich. Normalerweise werden keine Fragezeichen im Namen angegeben und die Bytes ex und s2 des FCB sind Null.
Wird der Name im Directory gefunden, kopiert das BDOS die relevante Information aus dem Directory in den FCB um so den lesenden oder schreibenden Zugriff auf die Datei zu erlauben. Auf eine Datei darf erst zugegriffen werden, wenn die Funktion erfolgreich beendet ist. Sie gibt in Register D0 einen Code mit einem Wert zwischen 0 und 3 zurück, wenn die Operation erfolgreich war. Ist der Wert des Registers D0 = 0FFH, dann konnte die Datei nicht gefunden werden. Falls Fragezeichen im FCB auftreten, wird die erste passende Datei eröffnet. Das Feld cr muß im FCB auf 0 gesetzt werden, falls die Datei sequentiell gelesen werden soll.

```
+-------------------------------------------+
!                                           !
! Funktion 16: Datei schliessen             !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 10H                       !
!  Register D1.L: FCB Adresse               !
!                                           !
! Resultat:                                 !
!  Register D0.W: Directory Code            !
!                                           !
+-------------------------------------------+
```

Diese Funktion ist invers zur vorhergehenden. Angenommen, der FCB, der durch D1 adresssiert wird, wurde vorher durch eine der Funktionen 15 oder 22 aktiviert. Dann überträgt die Funktion 16 den FCB in das Directory der angewählten Diskette. Der Dateisuchprozess ist derselbe wie beim Eröffnen. Der Ergebniswert in Register D0 liegt im Erfolgsfall wieder zwischen 0 und 3, und ist 0FFH, falls die Datei nicht gefunden wurde. Eine Datei braucht nicht geschlossen werden, wenn ausschließlich lesend zugegriffen

wurde. Falls jedoch auf die Datei geschrieben wurde, ist die Abschlussoperation wichtig, da sie das Directory auf den neuen Stand bringt.

```
+-------------------------------------------+
!                                           !
! Funktion 17: Durchsuchen DIR nach erstem  !
!              Eintrag                      !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 11H                       !
!  Register D1.L: FCB Adresse               !
!                                           !
! Resultat:                                 !
!  Register D0.W: Directory Code            !
!                                           !
+-------------------------------------------+
```

Diese Funktion durchsucht das Directory nach einem Dateinamen, der im FCB steht, der durch D1 adressiert wird. Wird die Datei nicht gefunden, enthält D0 den Wert 0FFH. Im anderen Fall zeigt der Inhalt von D0 (0,1,2 oder 3) an, daß die Datei vorhanden ist. Dann wird der DMA-Puffer mit dem Record gefüllt, der die Directory-Information enthält. Die relative Startadresse ist dabei D0*32. Obwohl sie normalerweise nicht gebraucht wird, kann sie so aus dem Puffer extrahiert werden.
Taucht auf einer Position des FCBs (Felder f1 bis ex) ein Fragezeichen auf ('?', ASCII 3FH /63 dez.), deckt es sich mit jedem Zeichen des Directory-Eintrags auf dem ausgewählten Laufwerk. Enthält das Feld dr ein Fragezeichen, wird das voreingestellte Laufwerk selektiert und die Funktion liefert jeden passenden Eintrag unabhängig davon, ob die Datei belegt ist und unabhängig vom User. Diese Funktion wird normalerweise nicht von Anwendungsprogrammen verwendet, sie erlaubt aber große Flexibilität beim Durchsuchen der Dateiverzeichnisse. Steht im Feld dr kein Fragezeichen, wird s2 auf 0 gesetzt.

```
+-------------------------------------------+
!                                           !
! Funktion 18: Durchsuchen DIR nach nächstem !
!              Eintrag                      !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 12H                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: Directory Code            !
!                                           !
+-------------------------------------------+
```

Diese Funktion ist der vorhergehenden ähnlich, es wird jedoch ab dem folgenden passenden Eintrag weitergesucht. Der Resultatwert entspricht den Angaben bei den Funktionen 17 und 18.

```
+-------------------------------------------+
!                                           !
! Funktion 19:  Datei löschen               !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 13H                       !
!  Register D1.L: FCB Adresse               !
!                                           !
! Resultat:                                 !
!  Register D0.W: Directory Code            !
!                                           !
+-------------------------------------------+
```

Diese Funktion entfernt die Dateien, deren Namen sich mit dem FCB-Eintrag decken. Der FCB wird durch D1 adressiert. Dateinamen und Dateityp dürfen Fragezeichen als Referenzen für alle Zeichen enthalten, nicht jedoch die Laufwerksangabe. Der Resultatwert hat denselben Aufbau wie bei den Funktionen 17,18,und 19.

```
+-------------------------------------------+
!                                           !
! Funktion 20: sequentiell lesen            !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 14H                       !
!  Register D1.L: FCB Adresse               !
!                                           !
! Resultat:                                 !
!  Register D0.W: Directory Code            !
!                                           !
+-------------------------------------------+
```

Angenommen der durch D1 adressierte FCB wurde durch die Funktionen 15 oder 22 aktiviert. Dann liest die Funktion 20 den nächsten Block zu 128 Byte aus der Datei in den DMA-Puffer. Der Block wird aus Position cr der Extension ex gelesen und cr auf die nächste Blocknummer erhöht. Tritt bei cr ein Überlauf auf, wird automatisch die nächste Extension eröffnet und cr auf 0 gesetzt. In D0 wird 00H zurückgegeben, wenn die Operation erfolgreich beendet wurde, während ein Wert ungleich Null geliefert wird, wenn keine Daten verfügbar waren (z.B. end-of-file).

```
+-------------------------------------------+
!                                           !
! Funktion 21: sequentiell schreiben        !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 15H                       !
!  Register D1.L: FCB Adresse               !
!                                           !
! Resultat:                                 !
!  Register D0.W: Directory Code            !
!                                           !
+-------------------------------------------+
```

Angenommen der durch D1 adressierte FCB wurde durch die Funktionen 15 und 22 aktiviert. Dann schreibt diese Funktion einen Block zu 128 Byte, der im DMA-Puffer steht auf die Diskettendatei, die vom FCB benannt wurde. Der Block wird auf Position cr der Datei geschrieben und danach cr erhöht. Wenn cr überläuft, wird automatisch die nächste logische Extension eröffnet und cr für den nächsten Schreibvorgang auf 0 gesetzt.Es kann in eine existierende Datei geschrieben werden, wobei neue Blöcke die früheren ersetzen. War die Schreiboperation erfolgreich, ist Register D0 = 00H.

Directory-Code:	01	Directory voll
	02	Diskette voll

```
+-------------------------------------------+
!                                           !
! Funktion 22: Datei kreieren               !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 16H                       !
!  Register D1.L: FCB Adresse               !
!                                           !
! Resultat:                                 !
!  Register D0.W:  Directory Code           !
!                                           !
+-------------------------------------------+
```

Diese Funktion ähnelt der Funktion 15 (Datei eröffnen), jedoch muß hier im FCB der Name einer Datei angegeben werden, die noch nicht im Directory der angewählten Diskette existiert. Das BDOS kreiert eine neue Datei und macht einen neuen Directory-Eintrag sowohl im FCB als auch auf der Diskette. Der Programmierer muß sicherstellen, daß kein doppelter Dateiname auftritt. Falls die Gefahr besteht, daß die Datei schon existiert, sollte vorher die Löschfunktion aufgerufen werden. Enthält nach Ausführung der Funktion das Registe D0 einen Wert zwischen 0 und 3, war die Operation erfolgreich, ist D0 = 0FFH, dann ist die Diskette voll.

```
+-------------------------------------------+
!                                           !
! Funktion 23: Datei umbenennen             !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 17H                       !
!  Register D1.L: FCB Adresse               !
!                                           !
! Resultat:                                 !
!  Register D0.W: Directory Code            !
!                                           !
+-------------------------------------------+
```

Diese Funktion verwendet den durch D1 adressierten FCB um alle Dateien mit dem Namen, der in den ersten 16 Bytes steht umzubenennen. Der neue Name steht in den folgenden 16 Bytes des FCB. Die Laufwerksnummer auf Position 0 wird verwendet um das gewünschte Laufwerk auszuwälen, wogegen für die Laufwerknummer auf Position 16 des FCB der Wert Null angenommen wird. Ist D0 gleich 0FFH, dann konnte keine Datei gefunden werden; im Erfolgsfall hat A einen Wert zwischen 0 und 3.

```
+-------------------------------------------+
!                                           !
! Funktion 24: Login Vektor liefern         !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 18H                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: Login Vektor              !
!                                           !
+-------------------------------------------+
```

Der Login Vektor ist ein 16-bit-Wert im Register D0, wobei das niederwertigste Bit von D0 dem Laufwerk A entspricht. Ist das dem Laufwerk entsprechende Bit = 0, kann dieses Laufwerk nicht angesprochen werden (nicht bereit oder nicht vorhanden), Ein 1-Bit zeigt an, daß das entsprechende Laufwerk bereit und aktivierbar ist (als Ergebnis einer expliziten oder impliziten Selektion des Laufwerks durch eine Dateioperation).

```
+-------------------------------------------+
!                                           !
! Funktion 25: Laufwerknummer angeben       !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 19H                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: Laufwerknummer            !
!                                           !
+-------------------------------------------+
```

Diese Funktion liefert die Nummer des Bezugslaufwerks, das gerade voreingestellt ist. Der Wert rangiert von 0 bis 15 entsprechend den Laufwerken A bis P.

```
+-------------------------------------------+
!                                           !
! Funktion 26: Setzen DMA-Adresse           !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 1AH                       !
!  Register D1.L: DMA Adresse               !
!                                           !
+-------------------------------------------+
```

Diese Funktion erlaubt es, die DMA-Adresse umzudefinieren und den DMA-Puffer in einen anderen Speicherbereich zu legen. Der Wert der Adresse bleibt gültig, bis er umdefiniert wird oder ein Kaltstart, Warmstart oder Rücksetzen des Disksystems erfolgt.Der Puffer muß auf einer geraden Adresse beginnen.

```
+-------------------------------------------+
!                                           !
! Funktion 28: Schreibschutz setzen         !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 1CH                       !
!                                           !
+-------------------------------------------+
```

Das gerade selektierte Laufwerk wird schreibgeschützt. Jeder Versuch vor dem nächsten Kalt- oder Warmstart auf diese Diskette zu schreiben liefert eine Fehlermeldung.

```
+-------------------------------------------+
!                                           !
! Funktion 29: Holen R/O Vektor             !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 1DH                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: Wert des Vektors          !
!                                           !
+-------------------------------------------+
```

Im Register D0 steht, welche Laufwerke schreibgeschützt sind. Wie in Funktion 24 ist das niederwerigste Bit dem Laufwerk A zugeordnet. Das entsprechende Schreibschutzbit wird entweder durch Funktion 28 gesetzt oder wenn CP/M feststellt, daß eine Diskette gewechselt wurde.

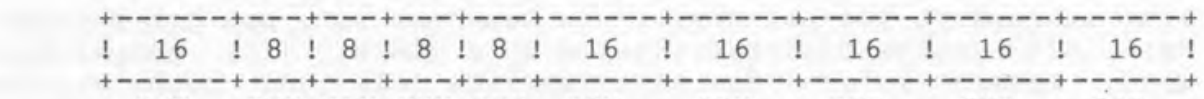
```
+-------------------------------------------+
!                                           !
! Funktion 30: Setzen Dateiattribute        !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 1EH                       !
!  Register D1.L: FCB Adresse               !
!                                           !
! Resultat:                                 !
!  Register D0.W: Directory Code            !
!                                           !
+-------------------------------------------+
```

Diese Funktion erlaubt es, vom Programm aus die Dateiattribute zu ändern. Insbesondere können Schreibschutz und das Systemattribut (t1' bis t3') gesetzt oder gelöscht werden. Die Register D1 adressieren einen FCB mit eindeutigem Dateinamen, in dem die entsprechenden Attribute gesetzt oder gelöscht sind. Funktion 30 durchsucht das Directory nach der Datei und ändert den Eintrag entsprechend. Die Indikatoren f1' bis f4' werden im Augenblick noch nicht verwendet, können aber nützlich für Anwendungsprogramme sein, da sie bei der Dateieröffnung (beim Schliessen) nicht mitverglichen werden . Die Indikatoren f5' bis f8' sind für Systemerweiterungen reserviert.

```
t1   RO  (nur lesen)
t2   SYS (Systemdatei)
t3   AR  (Archivdatei)
```

```
+-------------------------------------------+
!                                           !
! Funktion 31: Adresse Diskparameter        !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 1FH                       !
!  Register D1.L: CDBP-Adresse              !
!                                           !
! Resultat:                                 !
!  CDBP gesetzt                             !
!                                           !
+-------------------------------------------+
```

In D1 wird die Adresse des Disk Parameter Puffers übergeben. Diese Adresse kann für zwei Zwecke verwendet werden. Erstens lassen sich die Parameterwerte für die Anzeige und das Berechnen des Dateiplatzes verwenden, zweitens können Programme die Parameter ändern, wenn sich die Hardwareumgebung ändert. Normale Programmen brauchen diese Funktion nicht.



SPT Zahl der logische 128-Byte-Sektoren je Spur
BSH Blockverschiebungsfaktor
BLM Blockmaske
EXM Extensions-Maske
RES reserviert
DSM Gesamtzahl der Blöcke auf der Diskette
DRM Gesamtzahl der Directoryeinträge
RES reserviert
CKS Länge der Prüfsummenvektors
OFF Reservierte Systemspuren

```
+-----------------------------------------+
!                                         !
! Funktion 32: Holen/Setzen User          !
!                                         !
! Eintrittsparameter:                     !
!  Register D0.W: 20H                     !
!  Register D1.W: 0FFH (Holen) oder       !
!                 User Code (Setzen)      !
!                                         !
! Resultat:                               !
!  Register D0.W: User Code               !
!                                         !
+-----------------------------------------+
```

Ein Benutzerprogramm kann mit dieser Funktion die gerade aktive Benutzernummer (User Code) abfragen oder ändern. Falls Register D1 den Wert 0FFH besitzt, wird die aktuelle Benutzernummer (0 ... 15) in Register D0 geliefert. Ist Register D1 ungleich 0FFH, wird die Benutzenummer auf den Wert in D1 (modulo 16) gesetzt.

```
+-----------------------------------------+
!                                         !
! Funktion 33: Wahlfrei Lesen             !
!                                         !
! Eintrittsparameter:                     !
!  Register D0.W: 21H                     !
!  Register D1.L: FCB Adresse             !
!                                         !
! Resultat:                               !
!  Register D0.W: Fehlercode              !
!                                         !
+-----------------------------------------+
```

Diese Funktion ähnelt der Operation "sequentielles Lesen" früherer Versionen von CP/M, jedoch kann hier die Leseoperation für jede einzelne Blocknummer ausgeführt werden. Der gewünschte Block wird durch die Felder r0, r1 und r2 (Positionen 33-35 im

FCB) ausgewählt (24-bit-Wert). Zu beachten ist, daß der 24-bit-Wert mit seinem niederwertigsten Byte zuerst (r0) gespeichert wird. Danach folgen das mittelwertige (r1) und höchstwertige Byte(r2) der Blocknummer. CP/M verwendet r2 nur um die Größe einer Datei zu berechnen (Funktion 35). Byte r2 muß jedoch auf Null gesetzt werden, da ein Wert ungleich Null einen Überlauf über das Dateiende hinaus anzeigt.
Daher werden r0 und r1 als Doppelbyte - oder Wort - betrachtet, das die Blocknummer enthält. Die Nummer kann sich also im Bereich von 0 ... 3FFFFH bewegen und so jeden Block in einer 8-Megabyte-Datei adressieren. Um eine Datei mit wahlfreiem Zugriff (Random-Datei) zu bearbeiten, muss sie mit der Extension 0 eröffnet werden. Obwohl die Basisextension keine Daten enthalten muß, stellt sie sicher, dass die Datei im Directory richtig aufgeführt und beim DIR-Kommando aufgelistet wird. Nach dem Eröffnen wird die gewünschte Blocknummer in den Feldern r0 und r1 des FCB eingetragen und das BDOS aufgerufen um den Block zu lesen. Nach der Rückkehr enthält Register D0 den Wert 00, wenn die Operation erfolgreich war oder einen Fehlercode (siehe unten). Im ersten Fall enthält der DMA-Puffer den wahlfrei gelesenen Block. Im Gegensatz zum sequentiellen Lesen wird hier die Blocknummer nicht verändert. Daher lesen aufeinanderfolgende Random-Zugriffe immer denselben Block, wenn die Werte r0 und r1 nicht gändert werden. Bei jedem Random-Zugriff werden die Felder ex und cr automatisch gesetzt. daher kann die Datei auch sequentiell gelesen werden, nachdem durch Random-Zugriff auf einen Block positioniert wurde. Dabei wird beim Umschalten auf sequentielles Lesen der mit Random-Zugriff gelesene Block als erster nocheinmal gelesen. Der Programmierer kann selbstverständlich auch einfach die Blocknummer erhöhen um die Random-Datei sequentiell zu lesen.

Fehlernummern in Register D0 beim wahlfreien Lesen:

01	Lesen eines nicht beschriebenen (leeren) Blocks
02	-
03	Augenblickliche Extension kann nicht geschlossen werden
04	Suche nach nicht beschriebenener Extension
05	-
06	Suche hinter dem physikalischen Dateiende (Record-Nummer > 3FFFFH)

Die Fehler 01 und 04 treten auf, wenn der Versuch erfolgt, einen Block zu lesen, der nie vorher beschrieben wurde, oder eine Extension zu eröffnen, die nicht kreiert wurde. Fehler 03 tritt normalerweise nicht auf. Falls doch, kann er beseitigt werden, indem Extension 0 nocheinmal gelesen oder eröffnet wird (vorausgesetzt die Diskette ist nicht physikalisch schreibgeschützt). Fehler 06 tritt immer dann auf, wenn r2 ungleich Null ist. Normalerweise können die Fehlercodes als Anzeige für fehlende Daten aufgefaßt werden, während D0 = 00H anzeigt, das die Leseoperation

erfolgreich war.

```
+-------------------------------------------+
!                                           !
! Funktion 34: Wahlfrei Schreiben           !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 22H                       !
!  Register D1.L: FCB Adresse               !
!                                           !
! Resultat:                                 !
!  Register D0.W: Fehlercode                !
!                                           !
+-------------------------------------------+
```

Diese Operation wird ähnlich wie die vorhergehende initiiert, nur da hier die Daten nicht gelesen, sondern auf die Datei aus dem DMA-Puffer geschrieben werden. Weiterhin wird, falls der entsprechende Block oder die entsprechende Extension noch nicht eingerichtet wurden, dies vor dem Schreiben erledigt. Genau wie beim wahlfreien Lesen wird auch hier die Blocknummer nicht verändert. Die Felder ex und cr werden entsprechend der Blocknummer im FCB gesetzt. Es kann nach einer wahlfreien Schreiboperation sequentiell gelesen oder geschrieben werden, wobei der wahlfrei zugegriffene Block bei der ersten sequentiellen Operation gelesen oder überschrieben wird. Der Benutzer kann auch hier die Blocknummer erhöhen um einen sequentiellen Zugriff zu erreichen.

Die Fehlercodes sind die gleichen wie beim wahlfreien Lesen, wobei noch ein Fehler hinzukommt:

05 Neue Extension läßt sich nicht kreieren, da das Directory überläuft.

```
+-------------------------------------------+
!                                           !
! Funktion 35: Dateigröße berechnen         !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 23H                       !
!  Register D1.L: FCB Adresse               !
!                                           !
! Resultat:                                 !
!    Blockzahl im FCB gesetzt               !
!                                           !
+-------------------------------------------+
```

Zum Berechnen der Dateigrösse muss durch D1 ein FCB adressiert werden, der die Felder r0 bis r2 sowie einen eindeutigen Dateinamen enthölt. Nach Rückkehr vom BDOS enthält der FCB die virtuelle Dateilänge, die die Blocknummer des auf das Dateiende folgenden Blocks ist. Wenn nach einem Aufruf der Funktion 35 das

Feld r1 des FCB = 01 ist, hat die Datei die maximale Blockzahl von 65535 erreicht. Im anderen Fall, r2 = 0, bilden r0 und r1 einen 16-bit-Wert (r0 ist wieder das niederwertige Byte), der die Dateigröße angibt. D0 wird gelöscht.
An eine vorhandene Datei lassen sich Daten anfügen, indem durch Aufruf der Funktion 35 auf das Dateiende positioniert wird und nun durch eine Folge von wahlfreien Schreiboperationen an der ermittelten Blocknummer begonnen wird.
Die virtuelle Dateigröße entspricht der physikalischen Dateigröße, wenn die Datei sequentiell beschrieben wurde. Wenn die Datei wahlfrei kreiert wurde und "Löcher" in der Belegung existieren, kann die Datei weniger Blöcke enthalten, als die Dateigröße angibt. Wenn zum Beispiel nur der letzte Block einer 8-Megabyte-Datei geschrieben wurde, so ist die virtuelle Dateigröße 65536 Blöcke, obwohl nur ein Block tatsächlich belegt wurde.

```
+-----------------------------------------+
!                                         !
! Funktion 36: Setzen wahlfreien Block    !
!                                         !
! Eintrittsparameter:                     !
!  Register D0.W: 24H                     !
!  Register D1.L: FCB Adresse             !
!                                         !
! Resultat:                               !
!    Random Feld im FCB gesetzt           !
!                                         !
+-----------------------------------------+
```

Diese Funktion veranlaßt das BDOS, die Nummer eines wahlfrei zugreifbaren Blocks aus der augenblicklich beim sequentiellen Bearbeiten der Datei erreichten Position zu berechnen.Diese Funktion kann in zwei Fällen nützlich sein.
Erstens ist es oft notwendig, eine Datei erst sequentiell zu lesen um die Position einiger Schlüsselfelder aufzufinden. Wenn jeder Schlüssel gefunden ist, kann die Funktion 36 verwendet werden, um die Blocknummer des Schlüssels für einen direkten Zugriff zu berechnen. Ist die Datenrecordgröße 128 Byte, kann die resultierende Blocknummer des Schlüssels für spätere Abfragen in einer Tabelle gespeichert werden. Nachdem die Datei vollständig durchsucht wurde und alle Schlüssel in einer Tabelle gespeichert wurden kann sofort mit einem wahlfreien Lesezugriff über den Schlüssel auf die Daten zugegriffen werden.
Eine zweite Anwendung für die Funktion 36 bietet sich beim Wechsel vom sequentiellen zum wahlfreien Zugriff. Eine Datei wird sequentiell bis zu einem bestimmten Punkt bearbeitet, dann Funktion 36 aufgerufen (die die Blocknummer berechnet) und danach weitere Zugriffe wahlfrei, von dem berechneten Punkt ausgehend, durchgeführt.

```
+-------------------------------------------+
!                                           !
! Funktion 37: Zurücksetzen Laufwerk        !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 25H                       !
!  Register D1.L: Laufwerkvektor            !
!                                           !
! Resultat:                                 !
!  Register D0.W: 00H                       !
!                                           !
+-------------------------------------------+
```

Diese Funktion erlaubt das Rücksetzen der spezifizierten Laufwerke. Der in D1 übergebene Parameter ist ein 16-bit-Vektor der Laufwerke, die zurückgesetzt werden sollen. Das niederwertigste Bit entspricht dem Laufwerk A, das höchstwertige dem Laufwerk P.

```
+-------------------------------------------+
!                                           !
! Funktion 40: Wahlfrei schreiben, mit      !
!              Nullen füllen                !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 28H                       !
!  Register D1.L: FCB Adresse               !
!                                           !
! Resultat:                                 !
!  Register D0.W: Fehlercode                !
!                                           !
+-------------------------------------------+
```

Diese Funktion arbeitet wie Funktion 34, es werden jedoch vorher nicht zugegriffene Blöcke mit Nullbytes aufgefüllt, bevor die Daten geschrieben werden.

```
+-------------------------------------------+
!                                           !
! Funktion 46: Freien Diskettenplatz        !
!              bestimmen                    !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 2EH                       !
!  Register D1.W: Laufwerksnummer           !
!                                           !
! Resultat:                                 !
!  DMA-Puffer   : Freie Sektoren            !
!                                           !
+-------------------------------------------+
```

Diese Funktion liefert die Anzahl der noch freien logischen Sektoren von 128 Byte Größe in den ersten 4 Bytes des DMA-Puffers. Die Laufwerksnummer in D1 reicht von 0 (= Laufwerk A) bis 15, wogegen die Laufwerksnummern im DMA-Puffer den Bereich von 1

bis 16 verwenden.

```
+-----------------------------------------+
!                                         !
! Funktion 47: Folgeprogramm aufrufen     !
!                                         !
! Eintrittsparameter:                     !
!  Register D0.W: 2FH                     !
!                                         !
+-----------------------------------------+
```

Das augenblicklich laufende Programm wird beendet und ein anderes Programm geladen und gestartet. Dazu wird eine Kommandozeile im DMA-Puffer erwartet, die genauso aufgebaut ist, als würde sie von der Tastatur aus eingegeben. Die Zeile sieht folgendermaßen aus:

```
+---+-------------------------------------------------+---+
! n ! n Bytes Kommandotext  n<127                     ! 0 !
+---+-------------------------------------------------+---+
```

```
+-----------------------------------------+
!                                         !
! Funktion 48: Puffer wegschreiben        !
!                                         !
! Eintrittsparameter:                     !
!  Register D0.W: 30H                     !
!                                         !
! Resultat:                               !
!  Register D0.W: Fehlercode              !
!                                         !
+-----------------------------------------+
```

Diese Funktion ruft die BIOS-Funktion 21 auf. Sämtliche Dateipuffer geänderter oder neu beschriebener Dateien werden auf die Diskette zurückgeschrieben. D0 ist 0, wenn der Vorgang erfolgreich abgeschlossen wurde. Diese Funktion stellt sicher, das beim Schreiben auf eine Datei keine Information verloren geht.

```
+-----------------------------------------+
!                                         !
! Funktion 50: Direkter BIOS-Aufruf       !
!                                         !
! Eintrittsparameter:                     !
!  Register D0.W: 32H                     !
!  Register D1.L: BPB-Adresse             !
!                                         !
! Resultat:                               !
!  Register D0.W: BIOS-Rückmeldung        !
!                                         !
+-----------------------------------------+
```

Diese Funktion erlaubt den direkten Aufruf von BIOS-Funktionen (was den Nachteil hat, daß Programme, die diese Funktion verwenden, nicht mehr allgemein lauffähig sind). Das Register D1 enthält die Adresse eines 5 Wort langen Speicherpuffers, der die BIOS-Parameter enthält. Wenn das BIOS einen Wert zurückliefert, steht der in D0. Der Puffer muß auf einer geraden Adresse beginnen.

BPB:

Funktionsnummer des BIOS	
Wert P1	
Wert P2	

Die Parameter P1 und P2 werden in den Registern D1.L und D2.L an das BIOS übergeben.

Funktion 59: Programm laden
Eintrittsparameter:
Register D0.W: 3BH
Register D1.L: LPB Adresse
Resultat:
Register D0.W: Directory-Code

Diese Funktion lädt ein ausführbares Programm in den Speicher. In D1 wird die Adresse des Lade-Parameter-Blocks übergeben. Nach dem Laden enthält D0 einen Fehlercode:

00 erfolgreich geladen

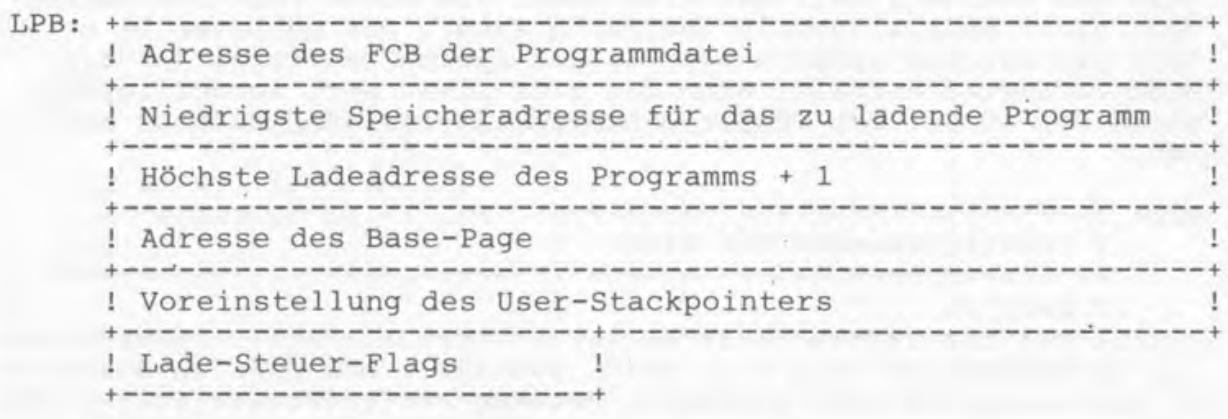
01 zuwenig Speicher oder Dateikopf falsch

02 Lesefehler beim Laden

03 fehlerhaftes Relocation-Bit

Vor dem Aufruf dieser Funktion muß die Programmdatei mit der Open-Funktion (Nummer 15) eröffnet werden.

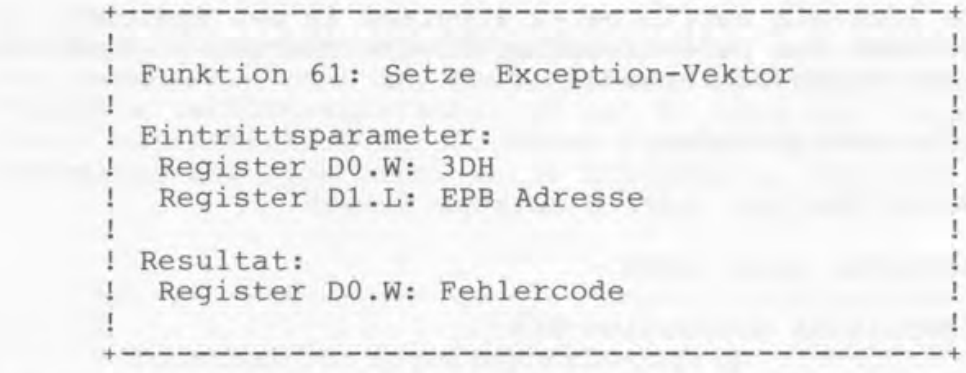
Der LPB besteht aus 11 Worten und hat folgenden Aufbau:



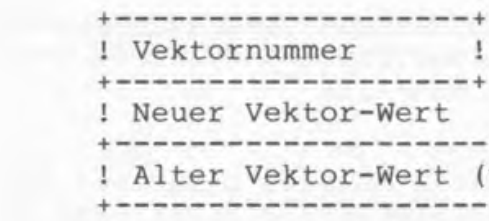
Die Lade-Steuerflags haben folgende Bedeutung:

Bit 0=0	Lade das Programm ab der niedrigsten Ladeadresse
Bit 0=1	Lade Programm in den höchsmöglichen angegeben Bereich

Die anderen Bits sind reserviert und sollen auf 0 gesetzt werden. Der CCP setzt seine Startadresse auf den Stack, sodaß das Programm mit RTS zurück in den CCP springen kann. Der CCP setzt außerdem die Base-Page-Adresse auf den Stack. Diese Adresse kann mit dem Offset 4(A7) erreicht werden. Die Adressen 0 bis 24H der Base-Page werden vom CCP initialisiert, die Adressen 25H bis 37H werden bereitgestellt, sind aber undefiniert. Außerdem reserviert das BDOS einen User-Stack-Bereich im höchsten RAM-Bereich.



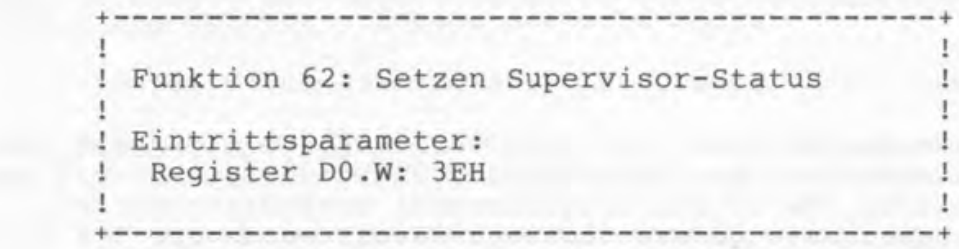
Diese Funktion erlaubt es, die Vektoren für die Ausnahmebehandlung umzubesetzen. D1 zeigt auf einen Parameterblock, der auf einer geraden Adresse beginnen muß und der folgenden Aufbau besitzt und 5 Worte lang ist:



Die Vektornummer spezifiziert die Exception (siehe Datenblatt 68000). Der Neue Vektor-Wert spezifiziert die Adresse der neuen Ausnahme-Behandler-Routine im Programm, im letzten Langwort wird der bisherige Wert (für die spätere Restaurierung) zurückgegeben. Das BDOS ersetzt den Exception Vektor und gibt die Kontrolle an das aufrufende Programm zurück.
Tritt eine Ausnahme auf, für die kein Exception-Handler existiert, meldet das BDOS:

```
Exception  nn  at user address  aaaaaaaa. Aborted.
```

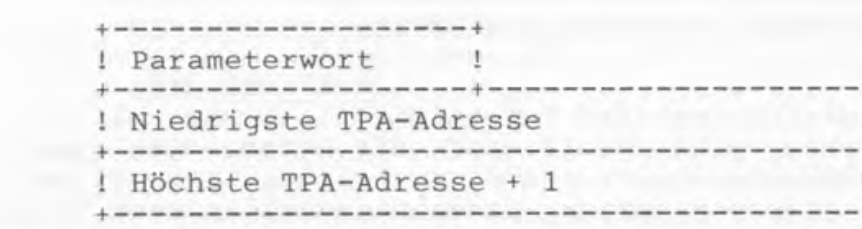
und bricht danach das Programm ab.



Diese Funktion setzt das aufrufende Programm in den Supervisorstatus. Sie sollte nur von fortgeschrittenen Programmierern für spezielle Anwendungen verwendet werden.
Der Supervisor-Stack wird von BDOS und BIOS verwendet, er ist 300 Langworte (1200 Byte) lang.

Warnung: Es ist möglich, daß diese Funktion in späteren Versionen von CP/M 68 K nicht mehr verfügbar ist.

Diese Funktion erlaubt das Festlegen des Arbeitsspeicher-Bereichs für Programme. Der Puffer ist 5 Worte lang und hat folgende Belegung:



Die Bits 2 - 15 des Parameterworts sind reserviert und müssen auf 0 gesetzt sein. Die Bits 0 und 1 des Parameterwortes haben folgende Bedeutung:

Bit 0 Wert 0 Gib die derzeitigen TPA-Grenzen in den Adressfeldern zurück.

Wert 1 Setze die TPA auf die angegebenen Grenzen.

Bit 1 Wert 0 Setze die TPA nach einen Warmstart auf die alten Adressen zurück.

Wert 1 Ersetze die TPA-Grenzen auf Dauer.

Man erhält also zum Beispiel mit 0000000000000000 die derzeitigen TPA-Grenzen mitgeteilt. Mit 0000000000000001 würde die TPA bis zum nächsten Warmstart (meist das Programmende) verändert und anschließend restauriert und mit 0000000000000011 würde die TPA bis zur nächsten Änderung neu definiert.

5.3 Geräte und IOBYTE

CP/M kennt vier logische Geräte, die natürlich ganz nach Belieben völlig anderen physikalischen Geräten zugeordnet werden können:

CONSOLE Die Konsole ist das Gerät, über das der Bediener mit dem CP/M kommuniziert. (Funktionen CONST, CONIN und CONOUT. Eine typische Konsole ist zum Beispiel ein Bildschirmgerät oder ein Fernschreiber (Teletype).

LIST Das logische Auflistungsgerät. Wenn es beim Benutzer vorhanden ist handelt es sich um einen Drucker oder einen Fernschreiber.

PUNCH Prinzipiell der Lochstreifenstanzer. Wenn er beim Benutzer existiert, ein schneller Stanzer, ein Fernschreiber oder auch ein Kassetten-Recorder.

READER Prinzipiell der Lochstreifenleser. Ein optischer Streifenleser, ein Fernschreiber oder ein Kassettenrecorder.

Den logischen Geräten LIST, PUNCH und READER kann ein einziges Peripheriegerät gleichzeitig zugewiesen werden (z.B.: Teletype mit LS-Leser und -Stanzer). Wenn den logischen Einheiten kein Gerät zugewiesen ist, sollte das CBIOS der Benutzers eine entsprechende Fehlermeldung ausgeben, sodaß sich das System nicht "aufhängt", wenn das Gerät von PIP oder einem anderen Programm angesprochen wird. Eine andere Möglichkeit wäre, daß die Routinen für LIST und PUNCH aus einem einzigen RET-Befehl bestehen und die READER-Routine mit dem Wert 1AH (end-of-file, Control/Z) im Register A zurückkehrt.
Um die Flexibilität zu erhöhen, kann der Benutzer die IOBYTE-Funktion implementieren, die ein Zuweisen von logische an physikalische Geräte erlaubt (siehe STAT-Kommando). Die Definition der IOBYTE-Funktion entspricht dem Intel-Standard: eine einzige Speicherzelle namens IOBYTE enthält die Information darüber, welches logische Gerät welchem physikalischen zugeordnet ist. Diese Zuordnung wird durch Aufteilen des IOBYTE in vier Felder zu je zwei bit erreicht. Die Felder erhalten die Gerätenamen CONSOLE, LIST, PUNCH und READER:

IOBYTE:

höchstwertiges bit niederwertigstes bit

7	6	5	4	3	2	1	0
LIST		PUNCH		READER		CONSOLE	

Der Wert jeden Feldes kann zwischen 0 und 3 liegen, der das entsprechende physikalische Gerät bezeicnet. Die möglichen Zuordnungen sind:

CONSOLE
- 0 die Konsole ist ein Konsoldrucker (TTY:)
- 1 die Konsole ist ein Sichtgerät (CRT:)
- 2 Batch-Modus: Verwende READER als Konsoleingabe und LIST als Konsolausgabe (BAT:)
- 3 Benutzerdefiniertes Konsolgerät (UC1:)

READER
- 0 der Streifenleser ist die Teletype (TTY:)
- 1 der Streifenleser ist ein schneller Leser (PTR:)
- 2 benutzerdefinierter Leser (UR1:)
- 3 benutzerdefinierter Leser (UR2:)

PUNCH
- 0 Stanzer ist die Teletype (TTY:)
- 1 Stanzer ist ein schneller LS-Stanzer (PTP:)

```
      2       benutzerdefinierter Stanzer (UP1:)
      3       benutzerdefinierter Stanzer (UP2:)

   LIST
      0       Auflisten auf der Teletype (TTY:)
      1       Auflisten auf Bildschirm (CRT:)
      2       Auflisten auf Drucker (LPT:)
      3       benutzerdefiniertes LIST-Gerät (UL1:)
```

Die Implementierung des IOBYTE ist optional und betrifft nur die Organisation des CBIOS. Außer PIP, das den Zugriff auf E/A-Einheiten erlaubt, und STAT, das die Gerätezuordnung auf Kommandoebene ermöglicht (siehe Kapitel 1) verwendet kein Modul oder Programm von CP/M das IOBYTE.

6 Das BIOS

6.1 Einführung

Um von der Hardware unabhängig zu sein, wurde CP/M in drei Einzelmodule geteilt:

BIOS	E/A-Kern, der hardwareabhängig ist,
BDOS	Diskettentreiber, der hardwareunabhängig ist,
CCP	Kommandointerpreter, der das BDOS verwendet.

Von diesen Modulen ist nur das BIOS von der Hardware des speziellen Computers abhängig. Das bedeutet, daß der Benutzer die Urversion von CP/M "patchen" kann, indem er ein neues BIOS zur Verfügung stellt, das eine Schnittstelle zwischen den anderen CP/M-Modulen und der benutzereigenen Hardware bildet.
Der Speicher des Computers wird unter CP/M 68 K folgendermaßen aufgeteilt:

CCP, BDOS und BIOS sind schon zur Sprache gebracht worden. Der Speicher unterhalb des CP/M heißt TPA (Transient Program Area), ist also der Bereich in dem die Programme ablaufen. Dabei werden absolute Programme (Dateityp .68K) und relokative Programme (Dateityp .REL) unterschieden. Absolute Programme müssen an einer festen Speicherstelle geladen werden, damit sie ablauffähig sind,

relokative Programme sind im Speicher frei verschieblich. Ein Programm besteht aus den Bereichen BSS, Datensegment und Textsegment.

BSS (block storage segment)
Dieser Bereich enthält nicht initialisierte Daten. Er wird beim Laden des Programms reserviert, ist aber nicht Teil der Programmdatei (wodurch Platz gespart wird.

Daten-Segment (data segment)
In Diesem Speicherbereich befinden sich die vordefinierten Variablen des Programms.

Text-Segment (text segment)
In diesem Teil des Speichers befindet sich das eigntliche Programm - der Maschinencode.

Base-Page
Dieser Speicherbereich dient der Kommunikation zwischen Programm und CCP.

6.2 Base-Page

Die Base-Page enthält die wichtigsten Angaben über das Programm, die ersten beiden Dateikontrollblöcke (FCBs) und den voreingestellten DMA-Puffer:

rel. Adresse	Inhalt
0000 - 0003	Niedrigste TPA-Adresse
0004 - 0007	Höchste TPA-Adresse + 1
0008 - 000B	Startadresse Textsegment
000C - 000F	Länge des Text-Segments in Bytes
0010 - 0013	Startadresse des Datensegments
0014 - 0017	Länge des Datensegments
0018 - 001B	Startadresse des BSS
001C - 001F	Länge des BSS
0020 - 0023	Länge des freien Speichers
0024	Laufwerk, von dem das Programm geladen wurde
0025 - 0037	reserviert
0038 - 005B	2. FCB (wird vom CCP ausgefüllt)
005C - 007F	1. FCB (wird vom CCP ausgefüllt)
0080 - 00FF	Voreingestellter DMA-Puffer (enthält auch die Kommandozeile)

6.3 Format der Kommandodatei

Eine Kommandodatei, also eine Datei mit einem ausführbaren Programm wird vom Linker in einem ganz bestimmten Format angelegt. Eine Kommandodatei besitzt einen Kopf (Header) und zwei Segmente, das eine für das Programm-Textsegment und das andere für das Datensegment. Optional können noch eine Symboltabelle und ein Segment mit der Verschiebe-Information (Relocation-Information)

hinzukommen. Der BSS-Bereich wird nicht in der Datei gespeichert, da er keine vordefinierten Daten enthält und seine Länge im Header festgelegt ist.
Es gibt zwei Header-Typen, einen für Programme, deren Text- und Datensegmente aufeinanderfolgen (was die Regel ist) und einen für Programmme, bei denen Text- und Datensegmente getrennt sind.

Zusammenhängende Programme:

rel. Adresse	Inhalt
0000	601AH (Kennung)
0002 - 0005	Zahl der Bytes im Textsegment
0006 - 0008	Zahl der Bytes im Datensegment
000A - 000D	Zahl der Bytes im BSS
000E - 0011	Zahl der Bytes in der Symboltabelle
0012 - 0015	reserviert, immer 0
0016 - 0018	Startadresse
001A	Relocation-Flag 0: relokativ <>0: absolut

Nicht zusammenhängende Programme:

rel. Adresse	Inhalt
0000	601BH (Kennung)
0002 - 0005	Zahl der Bytes im Textsegment
0006 - 0008	Zahl der Bytes im Datensegment
000A - 000D	Zahl der Bytes im BSS
000E - 0011	Zahl der Bytes in der Symboltabelle
0012 - 0015	reserviert, immer 0
0016 - 0018	Startadresse
001A	Relocation-Flag =0: relokativ ≠: absolut
001C - 001F	Startadresse des Datensegments
0020 - 0023	Startadresse des BSS

6.4 Symbol-Tabelle

Die Symboltabelle enthält alle im Programm definierten Symbole. Jeder Eintrag besteht aus 7 Worten:

Wort 1 - Wort 2	Name des Symbols	+-definiert	8000H
Wort 3 - Wort 4	null	! Wert per EQU	4000H
Wort 5	Typ ----------+	! global	2000H
Wort 6 - Wort 7	Wert (Adresse)	! Registerequated	1000H
		! external	800H
		! relokativ bez. Daten	400H
		! relokativ bez. Text	200H
		+-relokativ bez. BSS	100H

Die Symboltabelle eines Programms kann mithilfe des Kommandos NM68 ausgegeben werden. Das Kommando hat die Form:

```
NM68 Dateiname.O>LISTING
```

Wird keine Listing-Datei angegeben kommt die Information auf den Bildschirm.

6.5 Systemgenerierung

Die Systemgenerierung wird hier nur kurz beschrieben, näheres muß dem CP/M 68 K - System-Guide entnommen werden. CCP und BDOS liegen als relokative Dateien vor, es muß also nur mit Hilfe des Linkers das BIOS hinzugebunden werden. Das fertige System steht dann auf einer Datei mit dem Namen

```
CPM.SYS
```

Zunächst wird das BIOS geschrieben und übersetzt, danach kann die so erzeugte Datei BIOS.O in das System eingebunden werden. Die Kommandofolge dafür ist:

```
LO68 -R -UCPM -O CPM.REL CPMLIB BIOS.O

SIZE68 CPM.REL
```

Die hier ausgegebene Größe müssen Sie nun vom maximal vorgesehenen Speicher abziehen und 1 addieren. Damit haben Sie die höchstmögliche Adresse für das System : aaaaaa

```
RELOC -Baaaaaa CPM.REL CPM.SYS
```

Damit ist das System generiert. Als nächstes muß der Bootstraplader erzeugt werden. Dieses Programm steht auf den Systemspuren und lädt das CPM.SYS nach.
Der Lader ist im Prinzip ein abgemagertes BIOS, das nur die Funktionen zum Einlesen der Datei CPM.SYS und zum Setzen der Exception-Vektoren enthält. Im Monitor oder Boot-Rom des Computers muß dann nur noch ein Programm stehen, das die Systemspuren einliest und das eingelesene Boot-Programm startet. Angenommen, der Bootstrap-Lader soll ab Adresse 400H geladen und gestartet werden, dann kann das LDRBIOS mit dem Kommando

```
LO68 -S -T400 -ULDR -O CPMLDR.SYS BOOT.O LDRLIB LDRBIOS.O
```

erzeugt werden. Mit dem Kommando

```
PUTBOOT CPMLDR.SYS A
```

oder

```
XPUTBOOT CPMLDR.SYS A
```

wird das Boot-Programm auf die Systemspuren von Laufwerk A kopiert (Sie können natürlich auch ein anderes Laufwerk angeben).

6.6 BIOS-Aufrufe

Das BIOS wird ähnlich wie beim BDOS über den TRAP-Befehl aufgerufen, wobei vorher in den Registern D0 bis D2 die Parameter untergebracht werden. In D0 wird das Ergebnis, so eines vorhanden, zurückgegeben.

Der BIOS-Aufruf erfolgt mit TRAP #3.

```
+-------------------------------------------+
!                                           !
! Funktion 01: BIOS initialisieren          !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 00H                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: User-Bereich/Laufwerk     !
!                                           !
+-------------------------------------------+
```

Diese Funktion initialisiert das BIOS. In Register D0 wird im niederwertigen Byte das voreingestellte Laufwerk (0...15) und im höherwertigen Byte die Usernummer zurückgegeben.
Diese Funktion ist die einzige, die nicht mit TRAP #3, sondern mit einem Sprung nach _INIT angesprungen wird. Der Minimalcode für diese Funktion würde so aussehen:

```
_INIT:
     MOVE.L    #TRAPHNDL,$8C  ;SETZE TRAP #3-HANDLER
     CLR.L     D0
     RTS
```

```
+-------------------------------------------+
!                                           !
! Funktion 01: Warmstart                    !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 01H                       !
!                                           !
+-------------------------------------------+
```

Diese Funktion führt einen Warmstart des Systems durch. Es kann eine Wieder-Initialisierung der Hardware erfolgen. Diese Funktion wird nicht mit RTS beendet, sondern sie springt direkt in den CCP:

```
WBOOT:
          ...       ;HIER CODE
          JMP  _CCP
```

```
+-------------------------------------------+
!                                           !
! Funktion 02: Konsolstatus                 !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 02H                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: Status : 00H -> ready,    !
!                          FFH -> not ready !
!                                           !
+-------------------------------------------+
```

Diese Funktion liefert den Status des gerade eingestellten Konsol-Gerätes. Register D0 ist 0, wenn keine Eingabe ansteht.

```
+-------------------------------------------+
!                                           !
! Funktion 03: Zeichen von Konsole lesen    !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 03H                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: Zeichen von der Konsole   !
!                                           !
+-------------------------------------------+
```

Diese Funktion liefert das nächste Zeichen von der Konsole. Wenn kein Zeichen ansteht, wird auf das nächste Zeichen gewartet.

```
+-------------------------------------------+
!                                           !
! Funktion 04: Zeichen auf Konsole ausgeben !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 04H                       !
!  Register D1.W: Zeichen                   !
!                                           !
+-------------------------------------------+
```

Das Zeichen in D1 wird auf der Konsole ausgegeben.

```
+-------------------------------------------+
!                                           !
! Funktion 05: Zeichen auf Drucker ausgeben !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 05H                       !
!  Register D1.W: Zeichen                   !
!                                           !
+-------------------------------------------+
```

Diese Funktion gibt ein Zeichen auf das LST:-Gerät, meist der Drucker, aus.

```
+-------------------------------------------+
!                                           !
! Funktion 06: Ausgabe auf AUX:             !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 06H                       !
!  Register D1.W: Zeichen                   !
!                                           !
+-------------------------------------------+
```

Diese Funktion gibt ein Zeichen auf dem Zusatz-Ausgabegerät aus, zum Beispiel auf Lochstreifen oder Kassette.

```
+-------------------------------------------+
!                                           !
! Funktion 07: Eingabe von AUX:             !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 07H                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: Eingabezeichen            !
!                                           !
+-------------------------------------------+
```

Diese Funktion liest ein Zeichen vom Zusatzeingabegerät (Lochstreifen, Kassette o. ä.) ein. End-of-File wird als CONTROL/Z (1AH) gemeldet.

```
+-------------------------------------------+
!                                           !
! Funktion 08: Spur 0                       !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 08H                       !
!                                           !
+-------------------------------------------+
```

Diese Funktion fährt den Schreib/Lesekopf des selektierten Laufwerks auf Spur 0.

```
+-----------------------------------------------+
!                                               !
! Funktion 09: Laufwerk selektieren             !
!                                               !
! Eintrittsparameter:                           !
!  Register D0.W: 09H                           !
!  Register D1.B: Laufwerksnummer               !
!  Register D2.B: Login-Flag                    !
!                                               !
! Resultat:                                     !
!  Register D0.L: DPH des neuen Laufwerks       !
!                                               !
+-----------------------------------------------+
```

Diese Funktion selektiert ein neues Laufwerk. D1 hat einen Wert zwischen 0 und 15. Wenn versucht wird, ein nicht existierendes Laufwerk zu selektieren, ist D0 gelöscht. War ein Laufwerk schon einmal selektiert, ist D2 gleich 1, sonst ist D2 gleich 0.

```
+-----------------------------------------------+
!                                               !
! Funktion 10: Setze Spurnummer                 !
!                                               !
! Eintrittsparameter:                           !
!  Register D0.W: 0AH                           !
!  Register D1.W: Spurnummer                    !
!                                               !
+-----------------------------------------------+
```

Diese Funktion setzt auf dem angemeldeten Laufwerk den Kopf auf die gewünschte Spur. Es ist günstig, bei den Funktionen 9, 10 und 11 noch nicht auf das Laufwerk zuzugreifen, sondern die Angaben zu puffern und beim ersten Schreib-/Lesevorgang auszuführen.

```
+-----------------------------------------------+
!                                               !
! Funktion 11: Sektornummer setzen              !
!                                               !
! Eintrittsparameter:                           !
!  Register D0.W: 0BH                           !
!  Register D1.W: Sektornummer                  !
!                                               !
+-----------------------------------------------+
```

Diese Funktion setzt die Nummer des gewünschten Sektors auf der selektierten Diskette (siehe auch die Funktionen 9 und 10).

```
+-------------------------------------------+
!                                           !
! Funktion 12: DMA-Adresse setzen           !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 0CH                       !
!  Register D1.W: DMA-Adresse               !
!                                           !
+-------------------------------------------+
```

Diese Funktion liefert die Adresse des DMA-Puffers für die Disketten-Lese- und -Schreiboperationen. Der Puffer ist 128 Byte groß.

```
+-------------------------------------------+
!                                           !
! Funktion 13: Sektor lesen                 !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 0DH                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: 0, falls kein Fehler,     !
!                 1, falls Lesefehler       !
!                                           !
+-------------------------------------------+
```

Diese Funktion liest einen logischen Sektor in den DMA-Puffer. Sind die physikalisch gelesenen Sektoren größer als 128 Bytes, ist ein Blockungsalgorithmus im BIOS notwendig.

```
+-------------------------------------------+
!                                           !
! Funktion 14: Sektor schreiben             !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 0EH                       !
!  Register D1.W: 0 = normal schreiben      !
!                 1 = Directory-Sektor      !
!                 2 = erster Sektor eines   !
!                     neuen Blocks          !
! Resultat:                                 !
!  Register D0.W: 0 = kein Fehler           !
!                 1 = Fehler beim Schreiben !
!                                           !
+-------------------------------------------+
```

Diese Funktion schreibt einen Sektor von 128 Byte aus dem DMA-Puffer auf die Diskette (bei größeren physikalischen Sekoren ist ein Blockungsalgorithmus notwendig). Ist D1 = 1 sollte auf jeden Fall sofort physikalisch geschrieben werden. Der Wert D1 = 2 ist

bei einen Blockungsalgorithmus von Interesse.

```
+-------------------------------------------+
!                                           !
! Funktion 15: Status des Druckers          !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 0FH                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: 0, falls Drucker bereit   !
!                 FFH, falls Dr. nicht bereit!
!                                           !
+-------------------------------------------+
```

Diese Funktion liefert den Status des LST:-Gerätes.

```
+-------------------------------------------+
!                                           !
! Funktion 16: Sektor-Zuordnung             !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 10H                       !
!  Register D1.W: logische Sektornummer     !
!  Register D2.W: Adresse der Zuordnungs-   !
!                 Tabelle                   !
!                                           !
! Resultat:                                 !
!  Register D0.W: Physikalische Sektornummer !
!                                           !
+-------------------------------------------+
```

Diese Funktion liefert die physikalische Sektornummer zu einer logischen Sektornummer. Sie ist wichtig, wenn ein Interleaving-Faktor für schnelleren Diskettenzugriff eingebaut worden ist. Existiert keine Zuordnungstabelle, muß D2 gleich Null sein. In diesem Fall wird D1 nach D0 kopiert.

```
+-------------------------------------------+
!                                           !
! Funktion 18: Adresse der MRT              !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 12H                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: Adresse der MRT           !
!                                           !
+-------------------------------------------+
```

Aus Gründen der Kompatibilität mit anderen CP/M-Systemen ist

diese Funktion notwendig. Sie liefert die Adresse der Memory Region Table (MRT). Bei CP/M 68 K wird die einzige MRT, die TPA geliefert. Die MRT muß immer auf einer geraden Adresse beginnen, ihr Format ist:

```
Zahl der Einträge = 1                    16 Bit
Basisadresse der ersten Region           32 Bit
Länge der ersten Region                  32 Bit
```

```
+-------------------------------------------+
!                                           !
! Funktion 19: IOBYTE holen                 !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 13H                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: Wert des IOBYTE           !
!                                           !
+-------------------------------------------+
```

Diese Funktion liefert den aktuellen Wert des IOBYTE. Das IOBYTE wurde an anderer Stelle ausführlich behandelt.

```
+-------------------------------------------+
!                                           !
! Funktion 20: Setzen IOBYTE                !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 14H                       !
!  Register D1.W: Neuer Wert des IOBYTE     !
!                                           !
+-------------------------------------------+
```

Mit dieser Funktion kann das IOBYTE auf einen neuen Wert gesetzt werden (siehe Funktion 19).

```
+-------------------------------------------+
!                                           !
! Funktion 21: Puffer leeren                !
!                                           !
! Eintrittsparameter:                       !
!  Register D0.W: 15H                       !
!                                           !
! Resultat:                                 !
!  Register D0.W: 0000H = erfolgreich       !
!                 FFFFH = Fehler            !
!                                           !
+-------------------------------------------+
```

Diese Funktion veranlasst das BIOS, alle noch nicht auf die

Diskette übertragenen Puffer wegzuschreiben und das Directory gegebenenfalls zu aktualisieren. Dies gilt sowohl für den DMA-Puffer von 128 Byte Länge, als auch für einen größeren Sektorpuffer des Blockungsalgorithmus.

```
+-----------------------------------------------+
!                                               !
! Funktion 22: Setze Adresse des Ausnahme-      !
!              Behandlers                       !
!                                               !
! Eintrittsparameter:                           !
!  Register D0.W: 16H                           !
!  Register D1.W: Nummer der Exception          !
!  Register D2.L: Vektoradresse                 !
!                                               !
! Resultat:                                     !
!  Register D0.W: alte Vektoradresse            !
!                                               !
+-----------------------------------------------+
```

Diese Funktion setzt die Adresse einer Routine für die Behandlung von Ausnahmen (Exception Handler). Für spätere Restaurierung wird die alte Vektoradresse zurückgeliefert. Im Gegensatz zur gleichen Funktion beim BDOS, kann diese Routine alle Exceptions bearbeiten.

6.7 Parametertafeln für die Diskettenlaufwerke

Im BIOS enthalten sind Tabellen, die bestimmte Charakteristiken des Diskettensystems enthalten. Im Allgemeinen gehört zu jedem Laufwerk ein 16 Byte langer "disk parameter header" (Laufwerkparameterblock), der Informationen über das Laufwerk enthält und einen Arbeitsbereich für einige BDOS-Operationen zur Verfügung stellt. Das Format dieses Parameterblocks ist:

```
+-------+------+------+------+-------+-------+-------+-------+--
!  XLT  ! 0000 ! 0000 ! 0000 ! DIRBUF !  DPB  !  CSV  !  ALV  !
+-------+------+------+------+-------+-------+-------+-------+--
```

Dabei stellt jedes Element ein 16-bit-Wort dar. Die Bedeutung der Elemente ist im Einzelnen:

XLT — Adresse des Vektors für die Übersetzung von logischen in physikalischen Sektor für das Laufwerk. XLT hat den Wert 0000, falls keine Übersetzung stattfindet (logischer Sektor = physikalischer Sektor).

0000 — Arbeitsspeicher für BDOS.

DIRBUF — Adresse eines Arbeitsbereichs von 128 Byte für die Directory-Operationen des BDOS. Alle Parameter-

Blöcke verwenden denselben Bereich.

DPB — Adresse eines Parameter-Blocks für das Laufwerk. Alle Laufwerke mit identischer Charakteristik verwenden denselben DPB.

CSV — Adresse eines Arbeitsbereichs für das softwaremässige Prüfen des Diskettenwechsels. Für jeden Block separat.

ALV — Adresse eines Arbeitsbereichs,der vom BDOS für die Belegungsinformation der Diskette verwendet wird. Für jeden Block separat.

Bei n Laufwerken sind die DPHs in einer Tabelle angeordnet, deren erste 16 Byte zu Laufwerk 0 gehören, bis zur letzten Zeile, die dann zu Laufwerk n-1 gehört. Die Tabelle sieht dannn so aus:

DPBASE:

00	XLT 00	0000	0000	0000	DIRBUF	DPB 00	CSV 00	ALV 00
01	XLT 01	0000	0000	0000	DIRBUF	DPB 01	CSV 01	ALV 01
02	XLT 02	0000	0000	0000	DIRBUF	DPB 02	CSV 02	ALV 02
.								
.				und so weiter				
.								
n-1	XLTn-1	0000	0000	0000	DIRBUF	DPBn-1	CSVn-1	ALVn-1

Dabei markiert DPBASE die Basisadresse der DPH-Tabelle.

SELDISK muß die Basisadresse des DPH für das entsprechende Laufwerk liefern. Die Übersetzungsvektoren (XLT 00 bis XLT n-1) werden irgendwo im BIOS angeordnet und korrespondieren einfach eins-zu-eins mit den logischen Sektornummern von 0 bis (Sektorenzahl-1). Der Laufwerkparameterblock (DPB) eines jeden Laufwerks ist komplexer. Ein bestimmter DPB, der von einem oder mehreren DPHs adressiert werden kann, hat die allgemeine Form:

SPT	BSH	BLM	EXM	0	DSM	DRM	res	CKS	OFF
16B	8B	8B	8B	8B	16B	16B	16B	16B	16B

wobei Worte mit 16B und Bytes mit 8B markiert sind.

SPT — ist die Gesamtzahl der Sektoren je Spur

BSH — ist ein Belegungsblock-Verschiebungsfaktor,

abhängig von der Blockgröße.
=> LOG2(<Blockgröße> / 128)

BLM ist die Blockmaske (2 /BSH-1/).

EXM ist die Extensionsmaske, die festlegt wieviele logische Einträge in einem physikalischen Directory Eintag erfaßt werden.

DSM ist die totale Speicherkapazität der Diskette.

DRM ist die Anzahl der möglichen Directory-Einträge.

CKS Directory-Prüfvektor (Erkennen des Diskettenwechsels).

OFF Anzahl der für das System reservierten Spuren.

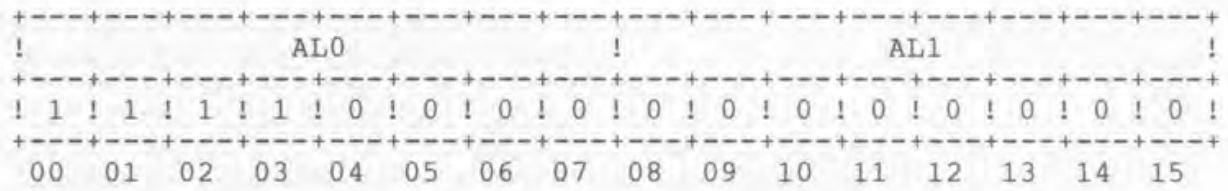
Die Werte von BSH und BLM bestimmen implizit die Blockgröße BLS, die nicht im DPH steht. Die Abhängigkeit von BLS, BSH und BLM ist:

BLS Größe	BSH Faktor	BLM Maske
1024	3	00000111B
2048	4	00001111B
4096	5	00011111B
8192	6	00111111B
16384	7	01111111B

Der Wert von EXM hängt von BLS und davon ab, ob der Wert von DSM kleiner als 256 oder größer als 255 ist.

DSM < 256		DSM > 255	
BLS	EXM	BLS	EXM
1024	00000000B	1024	-
2048	00000001B	2048	00000000B
4096	00000011B	4096	00000001B
8192	00000111B	8192	00000011B
16384	00001111B	16384	00000111B

Der Wert von DSM ist die maximale Blockzahl dieses Laufwerks. Das Produkt BLS*(DSM+1) ist die Maximalzahl der Bytes, die das Laufwerk zu speichern in der Lage ist. Dieser Wert muß kleiner oder gleich dem physikalischen Fassungsvermögen der Diskette nach Abzug der Systemspuren sein.
Der DRM Wert ist gleich der Maximalzahl der Directoryeinträge minus 1.Die Werte von AL0 und AL1 hängen jedoch von DRM ab. Die beiden Werte können als eine Folge von 16 Bits aufgefaßt werden:



wobei Position 00 mit dem höchstwertigen Bit von AL0 übereinstimmt und 15 mit dem niederwertigtsten Bit von AL1. Jede Bitposition reserviert einen Datenblock für Directory-Einträge, sodaß maximal 16 Blöcke reserviert werden können. Jeder Directory-Eintrag belegt 32 Byte, sodaß sich die folgenden Tabelle ergibt:

BLS	Anzahl Einträge
1024	32 * Anzahl bits
2048	64 * Anzahl bits
4096	128 * Anzahl bits
8192	256 * Anzahl bits
16384	512 * Anzahl bits

Wenn DRM = 127 (128 Directory-Einträge) und BLS = 1024 ist, gibt es 32 Directory-Einträge pro Block und es werden 4 reservierte Blöcke benötigt. In diesem Fall sind die 4 Höchstwertigen Bits von AL0 gesetzt (siehe Bild oben), also ist AL0 = 0F0H und AL1 = 00H.
Der Wert von CKS wird folgendermaßen ermittelt: Wenn das Speichermedium auswechselbar ist (z.B.: Diskette), dann ist CKS = (DRM + 1)/4, wobei DRM die Nummer des letzten Directory-Eintrags ist. Handelt es sich um ein Laufwerk mit fest montierter Platte (oder auch um Bubble Memories) ist CSK = 0.
OFF legt schließlich die Zahl der Spuren fest, die zu Beginn der Diskette oder Magnetplatte für das CP/M-System reserviert sind. Dieser Wert wird bei einem Aufruf von SETTRK automatisch addiert. Dieses Feld bietet auch die Möglichkeit große Platten in kleinere Einheiten zu unterteilen.
Verschiedene DPHs können schließlich denselben DPB adressieren, wenn ihre Laufwerkscharakteristiken übereinstimmen. Weiterhin kann der DPH dynamisch geändert werden, wenn ein neues Laufwerk selektiert wird indem ganz einfach der Zeiger auf den DPH geändert wird, denn das BDOS kopiert die DPB-Werte in einen lokalen Arbeitsbereich wenn SELDSK gerufen wird.
Wenn wir zum DPH eines speziellen Laufwerks zurückkehren, bleiben noch die Adressen CSV und ALV. Beide adressieren einen nichtinitialisierten Speicherbereich hinter dem BIOS. Diese Speicherbereiche müssen für jedes Laufwerk getrennt angelegt sein. Ihre Größe richtet sich nach den Werten im DPB.
Die Größe des von CSV adressierten Speicherbereichs beträgt CKS Bytes, was ausreicht, die Directory-Prüfdaten zu halten. Wenn CKS = (DRM + 1)/4 ist, müssen (DRM + 1)/4 Bytes reserviert werden; ist CKS = 0, wird kein Speicher belegt.
Die Größe des von ALV adressierten Speicherbereichs beträgt (DSM/8)+1 und ist festgelegt als die Maximalzahl der Datenblöcke, die für das Laufwerk erlaubt sind.

Die Disketten-E/A wird immer durch eine Aufruffolge der einzelnen Disketten-Zugriffsoperationen (Setzen Laufwerk, Spur und Sektor, Setzen DMA-Adresse) erledigt. Nachdem die Diskettenparameter gesetzt wurden, erfolgt dann ein Aufruf von READ oder WRITE für den eigentlichen Datentransfer. Es gibt oft einen einzigen Aufruf von SELDSK um ein Laufwerk zu aktivieren, von dem dann mehrfach gelesen oder auf das dann mehrfach geschrieben wird. Genauso ist nur ein Aufruf für das Setzen der DMA-Adresse nötig, der dann von beliebig vielen Lese- und Schreibaufrufen gefolgt werden kann. Die Routinen zur Wahl von Spur und Sektor werden jedoch vor jeder Schreib- oder Leseoperation aufgerufen.
Die Routinen READ und WRITE sollten im Fehlerfall mehrfach das Lesen oder Schreiben versuchen, ehe eine Fehlermeldung an das BDOS erfolgt (5 Versuche sind hier der Standard). Wenn die Fehlernachricht an das BDOS übergeben wird, meldet dieses den Fehler an den Bediener. Das Unterprogramm HOME muß nicht unbedingt wirklich eine Suche nach Spur 0 auslösen (das hängt auch vom Floppycontroller ab). Der Wichtige Punkt bei HOME ist, daß für die nächste Operation die Spur 0 ausgewählt wird. HOME kann also auch wie SETTRK mit Parameter 0 aufgefaßt werden.

6.8 Sector Blocking und Deblocking

Bei jedem Aufruf des BIOS WRITE, gibt das BDOS Information mit, die es erlaubt, effektiv die Sektoren auf Datenblöcke abzubilden und umgekehrt (Blocking und Deblocking), wenn das Plattensystem eine Sektorengröße besitzt, die ein Vielfaches der Grundeinheit 128 Byte beträgt. Hier sollen generell anwendbare Algorithmen vorgestellt werden, die sich in das BIOS einfügen lassen und die Information des BDOS verwenden, um die Abbildungen automatisch vorzunehmen.
Bei jedem WRITE-Aufruf liefert das BDOS die folgende Angabe:

```
0    = normales Schreiben eines Sektors
1    = Directory-Sektor schreiben
2    = Schreiben des ersten Sektors eines neuen Blocks
```

Der Fall 0 tritt immer bei einer Schreiboperation in einen vorher schon beschriebenen Bereich auf, etwa beim Ändern des Blocks einer Random-Datei, wenn nicht der erste Sektor eines noch nicht belegten Blocks geschrieben wird und wenn nicht in das Directory geschrieben wird. Der Fall 1 tritt auf, wenn ein Eintrag in das Directory erfolgt. Der Fall 2 tritt nur dann auf, wenn der erste Sektor eines vorher noch nicht belegten Blocks beschrieben wird. In den meisten Fällen schreiben oder lesen Anwenderprogramme mehrere Sektoren zu 128 Byte hintereinander. Daher ist der Mehraufwand durch das Blocking/Deblocking relativ klein, weil Leseoperationen vor dem Schreiben vermieden werden können.

6.9 Setzen Autostart-Kommando

Es ist möglich, beim Kaltstart nach dem Einschalten ein Kommando automatisch ausführen zu lassen. Dazu muß beim Linken des CPM.SYS

im CCP das Kommando eingetragen werden. Das geschieht am besten in der BIOS-Initialisierungs-Routine (Nr. 0), mit dem Namen

'_BIOS:INIT'.

Folgendes ist zu tun:

Das Byte auf Adresse _AUTOST muß auf 01H gesetzt werden.

Die Kommandozeile muß im Speicher ab der Adresse _USERCMD abgelegt werden. Der Kommandotext muß in Großbuchstaben geschrieben werden und er muß als letztes Zeichen ein Nullbyte enthalten. Die Zeile darf nicht mehr als 127 Zeichen enthalten.

7 Der C-Compiler

Dem CP/M 68 K liegt ein Compiler für die Sprache C bei. C ist eine höhere Programmiersprache, die speziell für die Zwecke der Systemprogrammierung entwickelt wurde. C ist gut lesbar und auf andere Systeme portabel. Zum Lernen der Sprache ist das im Vorwort genannte Buch von Kernighan und Ritchie zu empfehlen. Der vorliegende C-Compiler ist auf den 68000-Prozessor zugeschnitten, er unterscheidet sich daher in folgenden Punkten vom C-Standard, wie er unter UNIX verwendet wird:

- Der Typ int ist 16 Bit lang, Pointer sind 32 Bit lang. Alle Funktionen, die Langworte verwenden, müssen diese richtig deklariert haben.
- Die Registervariablen vom Typ long, int und char werden in D-Registern untergebracht. Es stehen 5 Register zur Verfügung.
- Registervariablen vom Typ pointer werden in A-Registern untergebracht. Es stehen 3 Register zur Verfügung.
- Alle lokalen Deklarationen müssen vor der ersten Anweisung stehen.
- Initialisierung von Strukturen werden so bearbeitet, als sei die Struktur ein Array von short integers.
- Die ersten 8 Zeichen eines Namens bzw die ersten 7 Zeichen eines Externals müssen sich voneinander unterscheiden.
- Es gibt keine Routinen für Gleitpunktoperationen.
- Zuweisungen an Strukturen sowie Strukturen als Prozedurparameter oder -Resultate werden nicht unterstützt.
- automatic variables werden nicht initialisiert.
- Es gibt keine Aufzählungstypen.

Das C-System besteht aus mehreren Programmen:

CP68	Der Preprozessor, der alle Makros wie z.B. #define bearbeitet.
C068	Der Parser für die Syntaxprüfung.
C168	Der Codegenerator

und der der Bibliothek mit dem Laufzeitsystem.

Damit der Benutzer die Kommandos nicht von Hand hintereinander aufrufen muß, existieren Submit-Dateien mit dem Namen C:SUB und CLINK.SUB. Diese Dateien sollten immer zum Aufruf der C-Compilation verwendet werden:

```
SUBMIT C <Parameter>
```

oder

```
C <Parameter>
```

Um danach den Linker aufzurufen, gibt man das Kommando:

```
SUBMIT CLINK <Parameter>
```

oder

```
CLINK <Parameter>
```

Der Parametertei besteht aus einem oder mehreren Dateinamen, die jeweils durch Leerzeichen getrennt werden. Wollen Sie zum Beispiel die C-Quellprogramme auf den Dateien EINS.C, ZWEI.C und DREI.C übersetzen und binden, geben Sie die Kommandos:

```
SUBMIT C EINS
SUBMIT C ZWEI
SUBMIT C DREI
SUBMIT CLINK EINS ZWEI DREI
```

In C-Programmen werden die einzelnen Prozeduren durch JSR aufgerufen, wobei die Argumente in umgekehrter Reihenfolge auf dem Stack abgelegt werden. Funktionsresultate werden immer im Register D0 zurückgegeben. Der Compiler ergänzt den Namen einer externen Funktion oder Variablen durch ein davorgesetztes Unterstreichungszeichen ('_').
Die Interfaceroutine mit argc/argv erlaubt die Übergabe von Argumentwerten in der Kommandozeile. So würde die Zeile:

```
KOMMANDO ARG1 ARG2 ARG3 ... ARGn
```

die folgende Belegung liefern:

```
argc      n+1
argv(0)   'CRuntime'   (der Kommandoname ist nicht erreichbar)
argv(1)   ARG1
argv(2)   ARG2
argv(3)   ARG3
...
argv(n)   ARGn
```

Das CP/M 68 K-C kennt drei Standard-Dateien:

Standardeingabe	STDIN	stdin
Standardausgabe	STDOUT	stdout
Fehlerausgabe	STDERR	stderr

Sowie die Geräte CON: und LST:.

Man kann bei C-Programmen, wie in Unix mit den Zeichen '<' und '>' die Ausgabe umleiten. Dabei darf zwischen der spitzen Klammer und dem Dateinamen kein Leerzeichen stehen. Das folgende Kommando würde ein Programm namens TEST.68K ausführen und dabei die Eingabedaten von der Datei EINS lesen und die Ausgabe auf den Drucker leiten. Die Argumentliste ist B C D E:

```
TEST <EINS >LST: B C D E
```

Sie sollten sich zunächst eine Diskette mit den wichtigsten Dateien für die Arbeit mit C zusammenstellen und dann alle .REL-Dateien in .COM-Dateien umwandeln, indem Sie den Relocator verwenden, zum Beispiel RELOC C128.REL C128.68K. Das Inhaltsverzeichnis Ihrer Diskette sollte dann etwa so aussehen:

```
A: CLINK    SUB : C        SUB : CE       SUB : CP68     68K
A: C068     68K : C168     68K : LIBF     A   : OSATTR   H
A: OSIF     H   : LIBE     A   : CPM      H   : PORTAB   H
A: SETJMP   H   : ERRNO    H   : STDIO    H   : ASSERT   H
A: OSIFERR  H   : SIGNAL   H   : SGTTY    H   : OPTION   H
A: CTYPE    H   : BDOS     S   : CBIOS    S   : LOADR    O
A: S        O   : OVHDLR   O   : W        O   : AS68INIT
A: CLIB         : ED       68K : FLOATDEM C   : AS68SYMB DAT
A: TIME     S   : AS68     68K : LO68     68K : STAT     68K
A: PIP      68K : CPM      SYS : CLINKE   SUB : CLINKF   SUB
```

Der C-Compiler besteht aus drei Komponenten, dem Preprozessor (CP68), dem Parser (C068) und dem Codegenerator (C168). Diese Programme werden übert die oben erwähnten SUBMIT-Jobs aufgerufen. Sie kommen normalerweise mit den Programmen nicht in Berührung. Es interessiert Sie aber sicher, welche Aufgaben die einzelnen Komponenten wahrnehmen:

Der Preprozessor CP68 nimmt Ihre C-Quelldatei und produziert eine Datei, in der alle #define- und #include-Anweisungen abgesättigt sind. Dass Kommando hat die Form

```
CP68 [-I d:] datei.C datei.I
```

Die -I-Option erlaubt die Wahl eines anderen Laufwerks. Sie können also die Dienstprogramme auf Laufwerk A und die C-Quellen auf Laufwerk B haben.

Der Parser C068 nimmt die vom CP68 erzeugte Datei und produziert zwei Dateien mit Zwischencode. Das Kommando lautet

```
C068 datei.I datei.IC datei.ST
```

datei.IC enthält den Zwischencode des Programms und datei.ST Konstantendefinitionen.

Der Codegenerator C168 nimmt die von C068 erzeugten Dateien und produziert daraus eine Assembler-Quelldatei. Das Kommando hat die Form

```
C168 datei.IC datei.S [-LD]
```

Die -L-Option legt alle Adressangaben auf 32 Bit Länge fest, die -D-option veranlasst den Compiler, die Zeilennummern aus der C-Quelle als Kommentare in die erzeugte Assembler-Quelle zu schreiben.
An dieser Stelle können Sie die Assembler-Quelle noch nachbearbeiten.

Als nächster Schritt erfolgt der Aufruf des Assemblers AS68 in der Form

```
AS68 -L -U [-F d:] [-S d:] datei.S
```

Der Assembler produziert datei.O.

Nun folgt der Linker LO68, der eine relokative Datei erzeugt. Das Kommando lautet

```
LO68 -R [-F d:] -O datei.68K S.O datei.O CLIB
```

Es gibt mehrere Submit-Dateien mit ähnlicher Aufgabe, die für die unterschiedlichen Formen der Gleitpunktarithmetik vorgesehen sind:

C.SUB und CLINKF.SUB sind für die Motorola-Gleitpunktarithmetik
CE.SUB und CLINKE.SUB sind für die IEEE-Gleitpunktarithmetik
C:SUB und CLINK.SUB sind für Programme ohne Gleitpunktarithmetik

Noch eine Anmerkung zum Schluß: Bei deutschen Tastaturen ist der "Anti-Schrägstrich", der bei C für die Angabe von Steuerzeichen verwendet wird, das große Ö.

Beispiel:
Das C-Programm:

```
test()  {
        long      a;
        int       b;
        char      c;
        register  d;

              .
              .
              .
              .

        b = blivot(c,a);
        }
```

erzeugt den folgenden Code:

```
_test:
     link      a6,#-8
     movem.l   d6-d7,-(a7)
          .
          .
          .
          .
     move.l    -4(a6),(a7)
     move.b    -8(a6),d0
     ext.w     d0
     move.w    d0,-(a7)
     jsr       _blivot
     add.l     #2,a7
     move.w    d0,-6(a6)
     tst.l     (a7)+
     movem.l   (a7)+,d7
     unlk      a6
     rts
```

Anhang

Das folgende BIOS wurde für den NDR-Klein-Computer geschrieben, es soll hier als Beispiel dafür dienen, wie ein BIOS aussehen könnte.

```
*********************************************
* BIOS 68k 1.2 fuer  NDR-Klein-Computer     *
* Version 2.0  // ROM //   R-D.Klein 850321 *
* // 8 " Laufwerke // 5 1/4 80 Spur //      *
* RAM-Floppy                                *
* Version fuer AS68      850323    1.0      *
* CPM.SYS  erzeugen                         *
* dazu  AS68 ndrbios.s                      *
* LO68 -r -ucpm cpm.rel cpmlib ndrbios.o    *
* RELOC -B19000 cpm.rel cpm.sys             *
* dann auf Systemdiskette copieren.         *
* bei AS68 und LO68 ggf. Laufwerke angeben  *
* ebenfalls mit -s x: die Init-Datei, siehe *
* CP/M Manual                               *
*********************************************

 .globl _init
 .globl _ccp
 .globl cpm     * Startadresse CP/M (erste Adresse)
*                 Damit ist das Bios,
*                 und damit CPM.REL verschiebbar.
* man muß nur bei RELOC eine neue Adresse angeben
* bei 128K  -B19000  , Kontrolle durch SIZE68 moeglich.
* bei 256K  -B39000

* CP/M Start, bei reloc
* angeben, und auch hier ggf.
* aendern, sonst reagiert das cp/m nicht
* denn es holt sich die Information aus
* der Memory region table
* kontrolle des wertes durch size68 durchfuehren
* wenn bios geaendert wird.

* 8 Zoll Version  // wird automatisch eingestellt
*                 // abhaengig von der Diskette
* Laufwerk   A,B,C,D       8 Zoll je 243K A,C = Vorderseite
*            E,F           5 1/4    je 800K
*            G             Ramfloppy

* 5 1/4 Zoll Version // wird automatisch eingestellt
*                    // abhaengig von der Diskette
* Laufwerk   A,B           5 1/4 " je 800K
*            C,D,E,F       8 " je 243K C,E=Vorderseite
*            G             Ramfloppy

* 8 "    koennen zweiseitig sein,
*        aber einfache Dichte, 128 Bytes / Sektor
```

```
*        mit 26 Sektoren pro Spur und 77 Spuren
*
* 5 1/4" (auch 3 1/2" oder 3")
*        muessen zweiseitig sein, 80 Spuren haben
*        und 1024 Bytes pro Sektor mit 5 Sektoren pro Spur
*        ( doppelte Dichte)
*
*
* die Ramfloppy ist fuer 256 K Byte ausgelegt
* sie beginnt bei der Adresse RAMFLO und
* muß vor Benutzung mit E5 vorgeloescht werden
* ein ERA *.* genuegt nicht.
* Ein BAD SEKTOR tritt auf, wenn ein Fehler
* auf der RAM-Karte festgestellt wurde. Dabei
* wird aber nur grob geprueft.

ramflo equ $80000 * Ramfloppy Startadr.


* I/O Byte wird ausgewertet
* 7     6     5     4     3     2     1     0
* -list-      -auxo-      -auxi-      -cons--
*
*
* cons:  *00=key/gdp 01=si/so 10=(auxi,list) 11= key/gdp
* auxi:   00=key    *01=si    10=ri          11= key
* auxo:   00=gdp    *01=so    10=po          11= gdp
* list:   00=gdp     01=so   *10=lo          11= gdp
*    * = Default nach Start
* andere Anordnungen durch STAT programmieren
* siehe CP/M68 Manual.


************************************************

_init:
*               zuerst Grundprogramm suchen
*               aber erst hinter der TPA
 lea $20400,a0 * Start der Suche 128K RAM davor annehmen
loop:
 cmp.l #$5aa58001,(a0)  * Kennung im Grundprogramm
 beq.s gefunden
 adda.l #$1000,a0       * Alle 4K Suchen nach dem Anfang
 bra.s loop
gefunden:
 cmp #$6000,$20(a0)
 bne.s loop             * dort muss Sprung stehen
 cmp #$6000,$24(a0)     * dort muss Sprung stehen
 bne.s loop             * ok Test scheint ok
*                         a0 -> Grundprog + $400
 suba.l #$400,a0        * BASIS-Adresse
 move.l a0,gru68k       * dort merken
 adda.l #$420,a0        * Adresse fuer Trap-Sprung
 move.l a0,gruj68k      * auch merken
* Bildschirm und Size sind im Boot vorbereitet
```

```
*                               Laufwerks-Dichte bestimmen
 move #1,d4                   * Laufwerk A
 move #76,d7                  * GETFLOP
 bsr trapexe
 move.b d4,drvcode            * $11=Mini DD, $21=Maxi SD
*
 clr.b mwrtflg                * alles voreinstellen
 move.b #$ff,lastsso          * damit wird Steprate zuerst eingest.
 clr.b mdrvakt                * 0=ungueltiges Laufwerk in Buffer
 clr cursor                   * Cursor ist jetzt aus
 move.b #0,stepmaxi           * PATCH PATCH PATCH maxi-Steprate
 move.b #0,stepmini           * PATCH PATCH PATCH mini-Steprate
*                               Rest vorbereiten
 move.b #$1e,d0               * 9600 Baud, 8 Bit, 1 Stopbit
 move.b #$0b,d1               * keine Paritaet, RTS high, T,R Freigabe.
 move #108,d7                 * SIINIT
 bsr trapexe                  * SER-Baugruppe fuer AUX (RDR,PUN).
*
 move.w #%10010100,iobyte * LO, SI/SO, KEY/GDP einstellen.
 move.l #traphndl,$8c     * trap #3 handler
 clr.l d0                 * Laufwerk A, User 0 ist Start
 rts

traphndl:
 cmpi #nfuncs,d0
 bcc trapng
 lsl #2,d0
 ext.l d0
 add.l #biosbase,d0
 movea.l d0,a0
 movea.l (a0),a0
 jsr (a0)
trapng:
 rte

biosbase:
 dc.l _init      * 0
 dc.l wboot      * 1
 dc.l constat    * 2
 dc.l conin      * 3
 dc.l conout     * 4
 dc.l lstout     * 5
 dc.l pun        * 6
 dc.l rdr        * 7
 dc.l home       * 8
 dc.l seldsk     * 9
 dc.l settrk     * 10
 dc.l setsec     * 11
 dc.l setdma     * 12
 dc.l read       * 13
 dc.l write      * 14
 dc.l listst     * 15
 dc.l sectran    * 16
 dc.l setdma     * 17
 dc.l getseg     * 18
```

```
 dc.l getiob     * 19
 dc.l setiob     * 20
 dc.l flush      * 21
 dc.l setexc     * 22
 dc.l getbasis   * 23   neuer Bios-Eingang. a4 belegen mit BASIS

nfuncs equ (*-biosbase)/4

getbasis:                * BIOS-Eingang, nur fuer Grundprg. Operationen
 move.l gru68k,a4        * aus Hilfszelle holen und in A4 als Ergebnis.
 rts

wboot:
 bsr puttrk              * falls noch aussteht
 clr.b mdrvakt           * und Laufwerk ungueltig danach
 jmp _ccp

constat:
 move iobyte,d0
 and #3,d0
 beq csts            * 00 = key
 cmp #1,d0
 beq sists           * 01 = si
 cmp #2,d0
 beq allfalse        * 10 = batch, auxi, kein Status
 bra csts            *

allfalse:            * kein Status bei BATCH
 clr d0              * nicht abbrechbar
 rts



alltrue:
 move #$00ff,d0
 rts

conin:
 move iobyte,d0
 and #3,d0
 beq ci              * 00 = key
 cmp #1,d0
 beq si              * 01 = si
 cmp #2,d0
 beq rdr             * 10 = batch, auxi
 bra ci              *


conout:
 move iobyte,d0
 and #3,d0
 beq co              * 00 = gdp
 cmp #1,d0
 beq so              * 01 = so
 cmp #2,d0
```

```
 beq lstout          * 10 = batch, auxi
 bra co              *


rdr:
 move iobyte,d0
 and #%00001100,d0
 beq ci              * 00 = key
 cmp #%00000100,d0
 beq si              * 01 = si
 cmp #%00001000,d0
 beq ri              * 10 = ri
 bra ci              *


pun:             * Ausgabe nur dann, wenn DSR auf
 move iobyte,d0
 and #%00110000,d0
 beq co              * 00 = gdp
 cmp #%00010000,d0
 beq so              * 01 = so
 cmp #%00100000,d0
 beq po              * 10 = po
 bra co              *


lstout:
 move iobyte,d0
 and #%11000000,d0
 beq co              * 00 = co
 cmp #%01000000,d0
 beq so              * 01 = so
 cmp #%10000000,d0
 beq lo              * 10 = lo
 bra co


liststst:
 move iobyte,d0
 and #%11000000,d0
 beq alltrue         * 00 = co, immer bereit
 cmp #%01000000,d0
 beq sosts           * 01 = so
 cmp #%10000000,d0
 beq losts           * 10 = lo
 bra alltrue         * Ausgabe immer bereit



csts:
 tst cursor
 bne conlstat             * Cursor ein, dann skip
 move #1,cursor
 move #61,d7
 bsr trapexe     * CURSEIN
```

```
conlstat:
 move #60,d7
 bsr trapexe     * Autoflip
 move #13,d7
 bsr trapexe     * CSTS
 beq.s noton
 moveq.l #$1,d0
 rts
noton:
 clr.l d0
 rts


ci:
 tst cursor
 beq conlin
 clr cursor
 move #62,d7
 bsr trapexe     * CURSAUS
conlin:
 move #12,d7
 bsr trapexe
 and.l #$7f,d0   * Wort wird ausgewertet
 rts


co:
 tst cursor
 beq conlout
 clr cursor
 move #62,d7
 bsr trapexe     * CURSAUS
conlout:
 move d1,d0
 move #33,d7
 bsr trapexe
 rts


lo:
 move d1,d0
 move #22,d7     * LO
 bsr trapexe
 rts

losts:
 move #117,d7    * LSTS
 bsr trapexe     * 0=nicht ready, $00ff=ready
 and #$ff,d0     * savety
 rts

so:
 move d1,d0      * HIGH liegt (12V).
 move #105,d7    * SO, Ausgabe seriell
 bsr trapexe     * d0.b bleibt Zeichen
```

```
 rts

sosts:
 move #107,d7
 bsr trapexe     * FF = True
 and #$ff,d0
 rts

si:
 move #104,d7    * SI, Eingabe seriell
 bsr trapexe     * d0.b = Zeichen
 rts

sists:
 move #106,d7    * SISTS, FFFFFFFF=Zeichen da
 bsr trapexe
 and #$ff,d0     * nur FF wenn TRUE
 rts

ri:
 move #14,d7     * RI
 bsr trapexe
 rts

po:
 move d1,d0
 move #15,d7     * PO
 bsr trapexe
 rts


home:
 clr.b track
 rts

seldsk:          * je nach Boot-Quelle
 moveq #0,d0
 cmp.b #maxdsk,d1 * max disk
 bpl selrtn
 move.b d1,seldrv
 bsr convdrv     * nach d0, Langwort 0,1,2,3,4,5,6
*                  unabh. von Boot 0..3 = maxi, 4..5=mini
 mulu #dphlen,d0 * v 2.0 alter Fehler behoben.
 add.l #dph0,d0
selrtn:
 rts

settrk:
 move.b d1,track
 rts

setsec:
 move.b d1,sector  * 1..256 moegl. 256->0 abgebildet
 rts
```

```
sectran:
 tst.l d2
 beq nosect
 movea.l d2,a0
 ext.l d1
 move.b 0(a0,d1),d0
 ext.l d0
 rts
nosect:
 move d1,d0
 add #1,d0       * 1..256 / 1..40 / besser fuer allg.
 rts             * Routinen, wie PUTBOOT, da in Manual so.


setdma:
 move.l d1,dma
 rts

dcode:           * Codieren Drive + Dense
 bsr convdrv     * d0= 0,1,2,3  immer. Laufwerkscode
 and #$3,d0      * sicherheitshalber
 move.l a0,-(a7)         * unabh. von gewaehltem Format
 lea dctab1,a0
 cmp.b #$21,drvcode      * wenn aber MINI geladen
 bne.s d1code
 lea dctab2,a0           * dann andere Tabelle
d1code:
 move.b 0(a0,d0.l),d4    * Ergebnis in D4
 movea.l (a7)+,a0
 rts


dctab1:          * MAXI BOOT erfolgte
 dc.b $11,$12,$91,$92

dctab2:          * MINI BOOT erfolge
 dc.b $14,$18,$94,$98


read:
 bsr convdrv     * 0..3 = maxi, 4..5 = min, constant
 cmp.b #4,d0
 bge readmini
 bsr delsso      * maxi einstellen ggf.
 move #1,d1      * Lesen
 clr d2
 clr d3
 move.b sector,d2
 move.b track,d3
 bsr dcode
 movea.l dma,a0
 move #75,d7
 bsr trapexe
 beq.s rdok
```

```
 move.w #1,d0
 rts
rdok:
 clr.w d0
 rts


write:          * d1= 0=norm, 1= dirwrt, 2= first
 move.b d1,alloc
 bsr convdrv    * 0..3=maxi, 4..5=mini, 6=ramflo
 cmp.b #4,d0
 bge writemini
 bsr delsso
 move #2,d1     * Schreiben
 clr d2
 clr d3
 move.b sector,d2
 move.b track,d3
 bsr dcode
 movea.l dma,a0
 move #75,d7
 bsr trapexe
 beq.s wrok
 move.w #1,d0
 rts
wrok:
 clr.w d0
 rts


readmini:
 cmp.b #6,d0
 beq readram
 bsr calc
 move.b mdrvakt,d0
 cmp.b d0,d4    * Laufwerk gleich
 bne rload      * nein, dann laden
 move.b mtrkakt,d0
 cmp.b d0,d3
 bne rload
 move.b msekakt,d0
 cmp.b d0,d2
 bne rload
r1rd:
 lea buffer,a1  * Quelle, adr. berechnen
 clr.l d1
 move.b sector,d1  * 1..40
 sub.b #1,d1    * 0..39
 and #$7,d1     * 0..7
 asl.l #7,d1    * *  128
 adda.l d1,a1   * ist Quelle
 movea.l dma,a0 * Ziel
 move #128-1,d3
r1lp1:
```

```
 move.b (a1)+,(a0)+
 dbra d3,r1lp1
 clr d0
 rts

rload:
 bsr puttrk     * falls alte Spur da, rueckschreiben
 bcs errflo
 bsr calc
 move.b d4,mdrvakt
 move.b d3,mtrkakt
 move.b d2,msekakt
 bsr gettrk
 bcc r1rd       * und dann transfer
errflo:         * GLOBAL
 move #1,d0
 rts



writemini:
 cmp.b #6,d0
 beq writeram
 bsr calc
 move.b mdrvakt,d0
 cmp.b d0,d4    * Laufwerk gleich
 bne wload      * nein, dann laden
 move.b mtrkakt,d0
 cmp.b d0,d3
 bne wload
 move.b msekakt,d0
 cmp.b d0,d2
 bne wload
w1wr:
 lea buffer,a1  * Ziel adr. berechnen
 clr.l d1
 move.b sector,d1 * 1..40
 sub.b #1,d1    * 0..39
 and #$7,d1     * 0..7
 asl.l #7,d1    * *  128
 adda.l d1,a1   * ist Ziel
 movea.l dma,a0 * Quelle
 move #128-1,d3
wr1lp1:
 move.b (a0)+,(a1)+
 dbra d3,wr1lp1
 move.b #1,mwrtflg * merken, das Sektor geschrieben
 cmp.b #1,alloc
 bne wr2
 bsr puttrk
 bne errflo
 clr.b mdrvakt   * ungueltiger Buffer
wr2:
 clr d0
```

```
 rts


wload:
 bsr puttrk       * falls alte Spur da, rueckschreiben.
 bcs errflo
 bsr calc
 move.b d4,mdrvakt
 move.b d3,mtrkakt
 move.b d2,msekakt
 bsr gettrk
 bcs errflo      * und dann transfer
 bra w1wr        * und weiter dann



* Buffer Verwaltung
* fuer 1K Sektoren

calc:            * Berechnet Code, LW, SEK, TRK...
*                  SEK->D2 TRK->D3 LW->D4
 bsr convdrv     * 4,5 ist immer code unabh. von Boot
 cmp.b #$21,drvcode * pruefen ob Mini-Boot.
 beq calmini
*                  transfer     E,F  ist nr
 move.b #$24,d4 * LW
 sub.b #4,d0     * 0,1 Ergebnis
 beq calc1
 move.b #$28,d4 * LW
calc1:
 bra calweiter
calmini:
 move.b #$21,d4 * dann A,B physikalisches Laufwerk
 sub.b #4,d0
 beq calweiter
 move.b #$22,d4
calweiter:
 move.b sector,d2 * 1..40
 sub.b #1,d2     * 0..39
 ror.b #3,d2     * nnn = 0..4
 and  #%111,d2   * Bereich Sektor
 add #1,d2       * 1..5 erlaubt
 move.b track,d3
 ror.b #1,d3
 bcc.s calc2     * Vorderseite
 or #$80,d4      * Rueckseite ansprechen// Achtung SSO verwenden.
calc2:
 and #$7f,d3     * Track gueltig
 rts

*                  MINIFLOPPY
selsso:          * sso setzen, falls noetig. Und Steprate
 cmp.b #$80,lastsso
 beq nosels
```

```
 clr d1          * step sel cmd
 move.b stepmini,d3
 or #$80,d3
 move #75,d7
 bsr trapexe
 move.b #$80,lastsso
nosels:
 rts


*                MAXIFLOPPY
delsso:         * sso ruecksetzen. Und Steprate
 cmp.b #$0,lastsso
 beq nosels
 clr d1
 move.b stepmaxi,d3
 move #75,d7
 bsr trapexe
 clr.b lastsso
 rts


gettrk:
 bsr selsso
 clr.b mwrtflg
 lea buffer,a0  * Ziel
 clr d2
 clr d3
 move.b msekakt,d2
 move.b mtrkakt,d3
 move.b mdrvakt,d4  * drive code valid
 move #1,d1      * LESEN
 move #75,d7
 bsr trapexe     * Floppy zugriff
 rts             * incl. Fehlercode


puttrk:
 tst.b mwrtflg
 beq noput
 bsr selsso
 clr.b mwrtflg
 lea buffer,a0  * Quelle
 clr d2
 clr d3
 move.b msekakt,d2
 move.b mtrkakt,d3
 move.b mdrvakt,d4  * drive code valid
 move #2,d1      * SCHREIBEN
 move #75,d7
 bsr trapexe     * Floppy zugriff
 rts             * incl. Fehlercode
```

```
noput:
 clr d0
 rts            * nicht ablegt, no error


readram:        * RAM FLOPPY   STARTET ab $80000 max 256K
 lea ramflo,a1  * Quelle
 clr.l d1       * savety
 move.b track,d1
 asl.w #8,d1    * 0..n
 move.b sector,d1 * 1..256(0) ueberlauf reverse
 sub.b #1,d1    * 0..255
 asl.l #7,d1    * * 128 (da Blockgroesse)
 and.l #$3ffff,d1 * max
 adda.l d1,a1   * Adresse fertig
 movea.l dma,a0 * Ziel
 move #128-1,d3
ramget:
 move.b (a1)+,(a0)+
 dbra d3,ramget
 clr.w d0       * no errors
 rts


writeram:       * RAM FLOPPY   STARTET ab $80000 max 256K
 lea ramflo,a1  * Ziel
 clr.l d1       * savety
 move.b track,d1
 rol.w #8,d1    * 0..n
 move.b sector,d1  * 1..256(0) Ueberlauf reverse
 sub.b #1,d1    * 0..255
 asl.l #7,d1    * * 128 (da Blockgroesse)
 and.l #$3ffff,d1
 adda.l d1,a1   * Adresse fertig
 movea.l dma,a0 * Quelle
 clr.b (a1)     * Speichertest kurz
 tst.b (a1)
 bne ramerr
 move.b #$ff,(a1)
 cmp.b #$ff,(a1)
 bne ramerr
 move #128-1,d3
ramput:
 move.b (a0)+,(a1)+
 dbra d3,ramput
 clr.w d0       * no errors
 rts

ramerr:         * trotzdem copieren
 move #128-1,d3
ram1put:
 move.b (a0)+,(a1)+
 dbra d3,ram1put
 move #1,d0
 rts
```

```
flush:
 bsr puttrk      * incl. flags, ggf zurueckschreiben
 bne flush1
 clr.b mdrvakt   * Buffer ist dann ungueltig
 clr d0          * ok
 rts
flush1:
 move #$ffff,d0
 rts

getseg:
 move.l #memrgn,d0
 rts

getiob:
 move iobyte,d0
 rts

setiob:
 move d1,iobyte
 rts

setexc:
 andi.l #$ff,d1
 lsl #2,d1
 movea.l d1,a0
 move.l (a0),d0
 move.l d2,(a0)
noset:
 rts

*
convdrv:        * ergibt logisches Laufwerk in D0
*                 abhaengig von drvcode
 cmp.b #$21,drvcode      * ok Mini
 beq.s conv1drv          * dann dorthin
 cmp.b #$11,drvcode      * Maxi
*                          hier Fehlerbehandlung moeglich
 clr.l d0                * Langwort gueltig
 move.b seldrv,d0        * Laufwerkscode holen 0,1,2,3,4,5,6
 rts                     * sonst direkt uebernehmen
conv1drv:                * 0,1,2,3 -> 4,5 und 4,5 -> 0,1
 clr.l d0
 move.b seldrv,d0
 move.l a0,-(a7)
 lea convtab,a0
 move.b 0(a0,d0.l),d0    * und Wert holen, Langwort gueltig
 movea.l (a7)+,a0
 rts                     * d0, gueltig

convtab:
 dc.b 4,5,0,1,2,3,6      * Konfiguration MINI-Boot.
 ds 0
```

```
trapexe:        * Grundprog ausfuehren ueber Sprung
 movem.l a5/a6,-(a7)
 move.l gruj68k,a6 * Zieladresse auf Stack
 jsr (a6)          * Sprung auf TRAP-Ersatz
 movem.l (a7)+,a5/a6
 rts
*

 .data


stepmaxi: dc.b 0 * 0=schnellste ,1,2,3
stepmini: dc.b 0 * 0=schnellste ,1,2,3
iobyte:   dc.w 0 * nur Byte gueltig.

* drvcode bestimmt das gebootete Laufwerk
* er entspricht der internen Codierung,
* also $11 bei Maxi, SD
* und  $21 bei Mini, DD
* nur diese beiden Formate werden z.Z. ausgewertet

drvcode: dc.b 0
         dc.b 0   * angleich EVEN

**
cursor:  dc.w 0   * Merker Cursor 1=ein, 0=aus
dma:     dc.l 0
track:   dc.b 0
sector:  dc.b 0

seldrv:  dc.b 0
lastsso: dc.b $FF * Merker letzter SSO-Zustand (0,$80) FF=unguelt.
*                   0=ohne SSO, $80=SSO war gesetzt
mdrvakt: dc.b 0   * Intern code 0=unguelt.
mtrkakt: dc.b 0   * fuer Bufferverwaltung
msekakt: dc.b 0
alloc:   dc.b 0

mwrtflg: dc.b 0   * <> 0 dann schreiben noch noetig

 ds 0

gru68k:  dc.l 0         * Basis-Adresse Grundprogramm
gruj68k: dc.l $420      * Basis-Adresse + $420 fuer Sprung


memrgn:
 dc.w 1              * nur eine TPA vorhanden (immer so.)
 dc.l $400           * Start der TPA
 dc.l cpm-$400       * 1024 Bytes gehen durch Vektoren verloren

* disk parameters
```

```
maxdsk equ 7
dphlen equ 26

dph0:    * 8 "  a
 dc.l xlt
 dc.w 0
 dc.w 0
 dc.w 0
 dc.l dirbuf
 dc.l dpb
 dc.l ckv0
 dc.l alv0

dph1:    * 8 " b
 dc.l xlt
 dc.w 0
 dc.w 0
 dc.w 0
 dc.l dirbuf
 dc.l dpb
 dc.l ckv1
 dc.l alv1

dph2:    * 8 " c
 dc.l xlt
 dc.w 0
 dc.w 0
 dc.w 0
 dc.l dirbuf
 dc.l dpb
 dc.l ckv2
 dc.l alv2

dph3:    * 8 " d
 dc.l xlt
 dc.w 0
 dc.w 0
 dc.w 0
 dc.l dirbuf
 dc.l dpb
 dc.l ckv3
 dc.l alv3

dph4:  * 5 1/4 " e
 dc.l 0          * no table
 dc.w 0
 dc.w 0
 dc.w 0
 dc.l dirbuf
 dc.l dpbmini
 dc.l ckv4
 dc.l alv4


dph5:    * 5 1/4 " f
```

```
 dc.l 0  * no table
 dc.w 0
 dc.w 0
 dc.w 0
 dc.l dirbuf
 dc.l dpbmini
 dc.l ckv5
 dc.l alv5


dph6:    * Ramfloppy g
 dc.l 0  * no table
 dc.w 0
 dc.w 0
 dc.w 0
 dc.l dirbuf
 dc.l dpbram
 dc.l 0  * no check
 dc.l alv6


dpb:
 dc.w 26
 dc.b 3
 dc.b 7
 dc.b 0
 dc.b 0
 dc.w 242
 dc.w 63
 dc.w $c000
 dc.w 16
 dc.w 2

dpbmini:
 dc.w 40         * 40 log. 128 Byte Sektoren
 dc.b 4          * BSH    2048 Bytes / Block
 dc.b 15         * BLM
 dc.b 0          * EXTNT MASK
 dc.b 0
 dc.w 388        * Kapazitaet * 2048
 dc.w 255        * Direktory Eintraege-1
 dc.w $f000      * Direktory mask (reserved )
 dc.w 64         * check size
 dc.w 4          * offset


dpbram:
 dc.w 256        * 256 log. 128 Byte Sektoren
 dc.b 3          * BSH    1024 Bytes / Block
 dc.b 7          * BLM
 dc.b 0          * EXTNT MASK
 dc.b 0
 dc.w 255        * Kapazitaet * 1024 , max 256K patch
 dc.w 63         * Direktory Eintraege-1
 dc.w $c000      * Direktory mask (reserved )
```

```
 dc.w 0          * no check
 dc.w 0          * offset

xlt:
 dc.b 1,7,13,19
 dc.b 25,5,11,17
 dc.b 23,3,9,15
 dc.b 21,2,8,14
 dc.b 20,26,6,12
 dc.b 18,24,4,10
 dc.b 16,22

 ds 0

 .bss

dirbuf:
 ds.b 128
ckv0: ds.b 16
ckv1: ds.b 16
ckv2: ds.b 16
ckv3: ds.b 16
ckv4: ds.b 64
ckv5: ds.b 64

alv0: ds.b 32
alv1: ds.b 32
alv2: ds.b 32
alv3: ds.b 32
alv4: ds.b 50          * (49+1), Achtung, alvs auf even
alv5: ds.b 50          * sonst Fehler in STAT
alv6: ds.b 32


buffer: ds.b 1024      * Sektor Speicher fuer Mini-Laufwerk


 ds 0

 .end
```

Das folgende Listing zeigt das Programm XPUTBOOT, mit dem die Systemsputen der Diskette beschrieben werden können.

```
*************************************************************
*                                                           *
*       Program to Write Boot Tracks for CP/M-68K (tm)      *
*                                                           *
*       Copyright Digital Research 1982                     *
*                                                           *
*************************************************************
*
*
*
prntstr =       9       BDOS Functions
dseldsk =       14
open    =       15
readseq =       20
dsetdma =       26
*
seldsk  =       9       BIOS Functions
settrk  =       10
setsec  =       11
isetdma =       12
write   =       14
sectran =       16
flush   =       21
*
bufcnt  =       $80
bufsize =       $80*bufcnt
*
        .text
*
start:  link    a6,#0
        move.l  8(a6),a0        base page address
        lea     $5c(a0),a1
        move.l  a1,fcb
        clr.b   hflag
        add     #$81,a0         first character of command tail
scan:   cmpi.b  #$20,(a0)+      skip over blanks
        beq     scan
        sub.l   #1,a0
scan1:  tst.b   (a0)
        beq     erxit
        cmpi.b  #$2d,(a0)+      check for -H flag
        bne     nohyph
        cmpi.b  #$48,(a0)+
        bne     erxit
        tst.b   hflag
        bne     erxit
        move.b  #$ff,hflag
        sub.l   #$24,fcb        change to 2nd default fcb
        bra     scan
nohyph: cmpi.b  #$20,(a0)
        bne     scan1
```

```
scan2:  cmpi.b  #$20,(a0)+
        beq     scan2
        cmpi.b  #$61,-(a0)      get disk letter
        blt     upper           upshift
        sub     #$20,(a0)
upper:  cmpi.b  #$41,(a0)       compare with range A - P
        blt     erxit
        cmpi.b  #$50,(a0)
        bgt     erxit
        move.b  (a0),d0
        ext.w   d0              put disk letter into range 0 - 15
        sub.w   #$41,d0
        move.w  d0,dsk
*
*       open file to copy
*
        move.w  #open,d0
        move.l  fcb,d1
        trap    #2
        cmpi.w  #$00ff,d0
        bne     openok
        move.l  #opnfl,d1
        jmp     erx
openok: move.l  fcb,a0
        clr.b   32(a0)
*
*       read
*
        move.l  #buf,d2
        clr.w   count
rloop:  move.w  #dsetdma,d0
        move.l  d2,d1
        trap    #2
        move.w  #readseq,d0
        move.l  fcb,d1
        trap    #2
        tst.w   d0
        bne     wrtout
        add.l   #128,d2
        add.w   #1,count
        cmpi.w  #bufcnt,count
        bgt     bufoflx
        bra     rloop
*
*       write
*
wrtout: move.w  #seldsk,d0      select the disk
        move.w  dsk,d1
        clr.b   d2
        trap    #3
        tst.l   d0              check for select error
        beq     selerx
        move.l  d0,a0
        move.l  14(a0),a0       get DPB address
        move.w  (a0),spt        get sectors per track
```

```
        move.w  14(a0),off      get offset
        clr.w   trk             start at trk 0
        move.w  #1,sect         start at sector 1
        lea     buf,a0
        tst.b   hflag
        bne     wrt1
        cmpi.w  #$601a,(a0)
        bne     wrt1
        add.l   #28,a0
wrt1:   move.l  a0,bufp
*
wloop:  tst.w   count
        beq     exit
        move.w  sect,d1         check for end-of-track
        cmp.w   spt,d1
        ble     sok
        move.w  #1,sect         advance to new track
        move.w  trk,d0
        add.w   #1,d0
        move.w  d0,trk
        cmp.w   off,d0
        bge     oflex
sok:    move.w  #settrk,d0      set the track
        move.w  trk,d1
        trap    #3
        move.w  sect,d1         set sector
        move.w  #setsec,d0
        trap    #3
        move.w  #isetdma,d0     set up dma address for write
        move.l  bufp,d1
        trap    #3
        move.w  #write,d0       and write
        clr.w   d1
        trap    #3
        tst.w   d0              check for write error
        bne     wrterx
        add     #1,sect         increment sector number
        sub     #1,count
        add.l   #128,bufp
        bra     wloop
*
exit:   move.w  #flush,d0       exit location - flush bios buffers
        trap    #3
        unlk    a6
        rts                     and exit to CCP
*
erxit:  move.l  #erstr,d1       miscellaneous errors
erx:    move.w  #prntstr,d0     print error message and exit
        trap    #2
        bra     exit
*
selerx: move.l  #selstr,d1      disk select error
        bra     erx
wrterx: move.l  #wrtstr,d1      disk write error
        bra     erx
```

```
bufoflx: move.l #bufofl,d1      buffer overflow
         bra    erx
oflex:   move.l #trkofl,d1
         bra    erx
*
*
         .bss
*
         .even
*
buf:     .ds.b  bufsize+128
*
fcb:     .ds.l  1               fcb address
spt:     .ds.w  1               sectors per track
sect:    .ds.w  1               current sector
trk:     .ds.w  1               current track
dsk:     .ds.w  1               selected disk
off:     .ds.w  1               1st track of non-boot area
count:   .ds.w  1
bufp:    .ds.l  1
hflag:   .ds.b  1
*
         .data
*
erstr:   .dc.b  'Invalid Command Line',13,10,'$'
selstr:  .dc.b  'Select Error',13,10,'$'
wrtstr:  .dc.b  'Write Error',13,10,'$'
opnfl:   .dc.b  'Cannot Open Source File',13,10,'$'
bufofl:  .dc.b  'Buffer Overflow',13,10,'$'
trkofl:  .dc.b  'Too Much Data for System Tracks',13,10,'$'
*


         .end
```

Sachverzeichnis

Plate
Anwenderhandbuch CP/M-68K

Jürgen Plate

Wer mit dem Betriebssystem CP/M-68K arbeitet, der sollte zu diesem Handbuch greifen. Bereits die ersten Schritte werden ihm dann leicht gemacht.
Die wichtigsten Informationen über das System sind hier in deutscher Sprache zusammengetragen und aufbereitet: Die Kommandos und deren Handhabung. Der Aufbau des Betriebssystems aus BIOS, BDOS und CCP. Der Editor, der Assembler und der Debugger. Weiterhin werden die Funktionen der Systemschnittstellen BDOS und BIOS im einzelnen und genau beschrieben.
Dem Programmierer bietet dieses Handbuch Beschreibungen und Informationen, die ihm helfen, sicher an das Ziel zu kommen. Nämlich selbständig Programme unter CP/M-68K zu schreiben. Alles in allem, wird mit diesem Handbuch eine ideale und kompakte Arbeitshilfe geboten.

ISBN 3-7723-9751-4

Franzis'