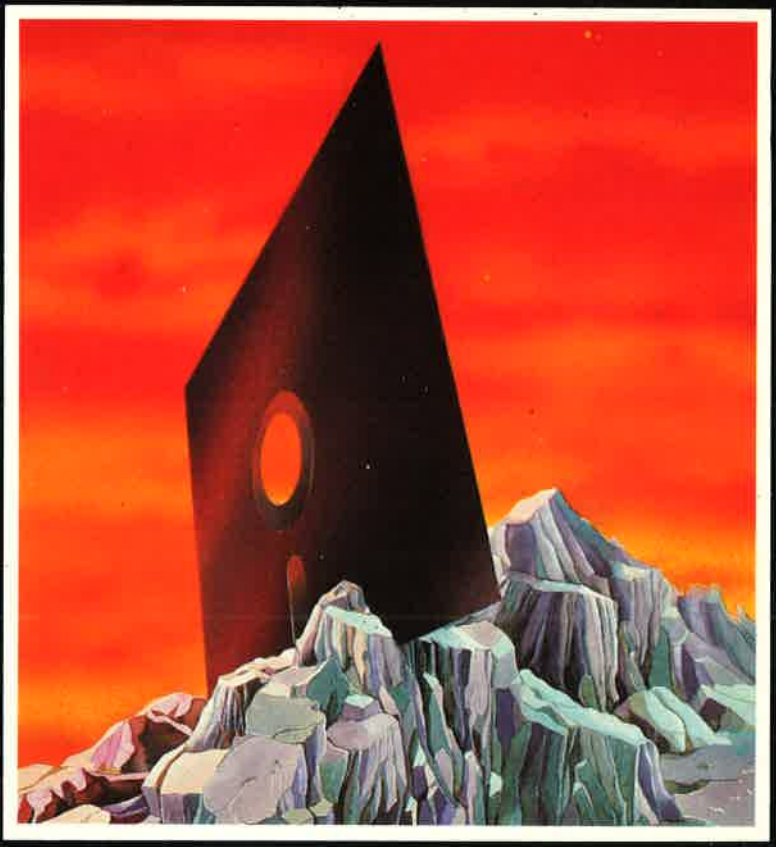
SYBEX

Programmieren mit CP/M



Alan R. Miller

Alan R. Miller

PROGRAMMIEREN MIT CP/M

BERKELEY · PARIS · DÜSSELDORF

Anmerkungen:
CP/M ist ein eingetragenes Warenzeichen von Digital Research, Inc.
Grammatik ist ein Warenzeichen von Aspen Software Co.
Lifeboat ist ein Warenzeichen von Lifeboat Associates
MAC ist ein Warenzeichen von Digital Research, Inc.
MACRO-80 ist ein Warenzeichen von Microsoft Corporation
MBASIC ist ein Warenzeichen von Microsoft Corporation
SID ist ein Warenzeichen Digital Research, Inc.
Spellguard ist ein Warenzeichen von Sorcim Corporation
WordStar ist ein Warenzeichen von MicroPro International Corporation
Z80 ist ein eingetragenes Warenzeichen von Zilog, Inc.

Originalausgabe in Englisch
Titel der amerikanischen Ausgabe: "Mastering CP/M"

Deutsche Übersetzung: Peter Mühlbach
Umschlagentwurf: Daniel Le Noury
Satz: tgr – typo-grafik-repro gmbh., remscheid
Gesamtherstellung: Druckerei Hub. Hoch, Düsseldorf

ISBN 3-88745-077-9
1. Auflage 1984
2. Auflage 1985

Inhaltsverzeichnis

Vorwort

CP/M ist inzwischen zum Standard-Betriebssystem für Z80-, 8080- und 8085-Mikrocomputer geworden. Als Folge davon entstand eine große Zahl von Programmen, die unter CP/M laufen. Hier eingeschlossen sind Assembler, Editorprogramme, Programme zum Auffinden von Schreibfehlern, Übersetzer-Programme für die technischen Sprachen BASIC, Pascal, FORTRAN und APL, sowie allgemeine kommerzielle Programmpakete.

Einige CP/M-Programme können automatisch ausgeführt werden, so daß nur geringes Wissen über CP/M nötig ist. Andere Programme jedoch erfordern größere Kenntnis des Betriebssystems. In jedem Fall erfordern einige Routineaufgaben, wie das Formatieren von Disketten und das Erstellen von Sicherungskopien wichtiger Disketten, bessere Kenntnis des Betriebssystems. Unglücklicherweise ist es schwierig, aus den mitgelieferten Unterlagen die Arbeitsweise von CP/M zu erlernen. Es gibt hierüber einführende Bücher, jedoch behandeln sie die Interna von CP/M nicht ausführlich genug. Darüber hinaus gibt es zahlreiche Inkonsistenzen und besondere Eigenarten im CP/M, die nur darauf warten, dem unaufmerksamen Programmierer eine Falle zu stellen.

Der Autor arbeitet mit CP/M seit seiner Entstehung (Version 1.3). Er hat viele Techniken entwickelt, um die Brauchbarkeit von CP/M zu verbessern, sowohl durch teilweise Änderung von CP/M, als auch durch die Erstellung von Assembler-Hilfsprogrammen. Dieses Buch gibt die Erfahrung des Autors weiter. Es ist eine Anleitung für jemanden, der tiefergehende Kenntnis über die interne Arbeitsweise von CP/M erlangen möchte.

Obwohl die Wirkungsweise jedes Programms beschrieben wird, sollte der Leser bereits einige Erfahrung mit 8080-Assembler-Programmierung haben. Weitere Informationen über die Programmierung in der Assemblersprache findet man in dem ebenfalls bei SYBEX erschienenen Buch „Programmierung des Z80". (1) Um in den vollen Genuß des vorliegenden Buches zu gelangen, sollte man einen Computer mit CP/M, einen Editor, einen Makro-Assembler wie MAC oder MACRO-80 sowie einen Assemblersprache-Debugger wie SID oder DDT zur Verfügung haben.

Dieses Buch beginnt mit einer ausführlichen Beschreibung der Organisation und Arbeitsweise von CP/M. Die Themen schließen den System-Parameter-Bereich, die TPA, den CCP sowie BDOS und BIOS ein. Der

Gebrauch der eingebauten Kommandos, Steuerzeichen und Benutzerprogramme wird ebenfalls erläutert. Routinearbeiten wie das Formatieren von Disketten und Erstellen von Sicherungskopien werden in Kapitel 2 behandelt. Die Funktionen COPY, SYSGEN und SAVEUSER werden besprochen bis hin zur Vorgehensweise beim Ändern des CP/M-Systems und Speichern der geänderten Version auf die Diskette. In Kapitel 3 werden wir die Auswertung des IOBYTE zum BIOS hinzufügen.

Das Makrokonzept wird in Kapitel 4 vorgestellt. Makros zum Vergleichen, Umspeichern und Füllen von Speicherbereichen bilden die Grundlage einer Makrobibliothek. Der Gebrauch von BDOS-Operationen für Konsol-Ein-/Ausgabe mit Makros wird in Kapitel 5 gezeigt. Außerdem werden mehrere lauffähige Programme geschrieben.

Die Kapitel 6 und 7 beschreiben das CP/M-Disketten-Dateisystem. Die Makrobibliothek wird um BDOS-Operationen zum Lesen und Schreiben von Disketten-Dateien erweitert, und weitere lauffähige Programme werden geschrieben. Das letzte Kapitel befaßt sich mit Einzelheiten zum CP/M-Disketten-Inhaltsverzeichnis. Ein allgemeines Hilfsprogramm zum Anzeigen der Diskettenparameter, einer Blockzuordnungstabelle und zur ausführlichen Darstellung des Inhaltsverzeichnisses wird geschrieben.

Die Anhänge enthalten Hinweise für die Erstellung von 8080- und Z80-Assembler-Programmen. Anhang A zeigt den ASCII-Code in dezimaler, hexadezimaler und oktaler Form. Anhang B zeigt eine 64K-Speicherliste. Anhänge C und D enthalten den 8080-Befehlssatz in alphabetischer und numerischer Reihenfolge. Die Anhänge E und F geben den gesamten Z80-Befehlssatz in alphabetischer und numerischer Reihenfolge wieder, entsprechend dem offiziellen mnemotechnischen Code von Zilog. Befehle, die auch beim 8080 vorhanden sind, sind mit einem Stern (*) gekennzeichnet.

Die 8080-Befehle werden ausführlich im Anhang G behandelt, einschließlich möglicher Fallen und besonderer Techniken. Die Namen der entsprechenden Z80-Befehle sind ebenfalls vermerkt. Anhang H liefert eine ausführliche Beschreibung der Z80-Befehle. Anhang I faßt alle CP/M-BDOS-Aufrufe zusammen.

Alle in diesem Buch gezeigten Assemblerprogramme wurden auf einem Z80-Mikrocomputer mit drei 5-Zoll-Diskettenlaufwerken (Laufwerke A, B und C) und zwei 8-Zoll-Diskettenlaufwerken (Laufwerke D und E) entwickelt. Das Betriebssystem war die Lifeboat Version 2.2 von CP/M. Die Quellprogramme wurden mit MicroPro's Texteditor WordStar geschrieben. Übersetzt wurden sie sowohl mit dem MAC-Assembler von Digital Research als auch mit Microsoft's MACRO-80.

Kapitel 1

Der Aufbau und die Arbeitsweise von CP/M

EINFÜHRUNG

Die Aufgabe dieses ersten Kapitels ist es, die Organisation und Arbeitsweise des Betriebssystems CP/M zu besprechen. Als erstes werden wir die verschiedenen Teile von CP/M, den Systemparameterbereich, den Anwenderprogrammbereich, den Kommandoprozessor, das Basis-Diskettenbetriebssystem und das Basis-Ein-/Ausgabesystem behandeln. Danach erhalten wir eine Übersicht über die Arbeitsweise von CP/M, inklusive der eingebauten Kommandos, der Steuerzeichen und der System-Programme. (Weitere Einzelheiten zu diesem Thema erfährt man im „CP/M Handbuch".) Nach der Untersuchung von Standardprogrammen wie STAT und PIP werden wir ein neues Kommando, CONTIN, schreiben und sehen, wie und warum es läuft.

DIE SPEICHER-ORGANISATION

Die Hardware eines Computers kann man logisch in mehrere Teile untergliedern, wie die Zentraleinheit (CPU), den Hauptspeicher (RAM) und die Zusatzgeräte wie Konsole, Drucker, Telefonmodem und Diskettenlaufwerke. Das Disketten-Betriebssystem (DOS) ist ein Programm, das die Aufgaben eines Computers koordiniert. CP/M ist das am weitesten verbreitete DOS für die Prozessoren 8080, 8085 und Z80. Lassen Sie uns einen Blick auf die Arbeitsweise von CP/M werfen.

Die Prozessoren 8080, 8085 und Z80 sind sich sehr ähnlich. Alles, was in diesem Kapitel entwickelt wird, gilt gleichermaßen für alle drei. Die CPUs können maximal 64K Bytes RAM direkt adressieren (genau 65536 Bytes). Jedem Byte RAM ist genau eine Adresse von 0 bis 65535 zugeordnet. CP/M teilt diesen Bereich in fünf Regionen auf, nämlich:

- Den Systemparameterbereich
  Diese Region enthält Informationen wie die laufende Disketten-Laufwerksnummer und Anwendernummer, Gerätezuordnungen, die

Adressen des Basis-Ein-/Ausgabe-Systems und des Basis-Diskettenbetriebssystems, die Restartadressen und den Standardpufferbereich.

- Den Anwenderprogrammbereich (TPA)
  Dies ist der Arbeitsbereich des Speichers. Ausführbare Programme und ihre Daten werden hier gespeichert.
- Den Kommandoprozessor (CCP)
  Dieser Bereich enthält die Programme für die eingebauten Kommandos DIR, ERA, REN, SAVE, TYPE und USER.
- Das Basis-Diskettenbetriebssystem (BDOS)
  Hier liegen die allgemeinen Programme zum Bedienen der Zusatzgeräte.
- Das Basis-Ein-/Ausgabe-System (BIOS)
  Dieser Bereich enthält die vom Anwender angepaßten Routinen für den tatsächlichen Datenaustausch mit den Zusatzgeräten.

Die Bereiche BDOS und BIOS werden auch zusammenfassend Gesamt-Diskettenbetriebssystem (FDOS) genannt. Eine Übersicht über diese fünf Regionen zeigt Abb. 1.1.

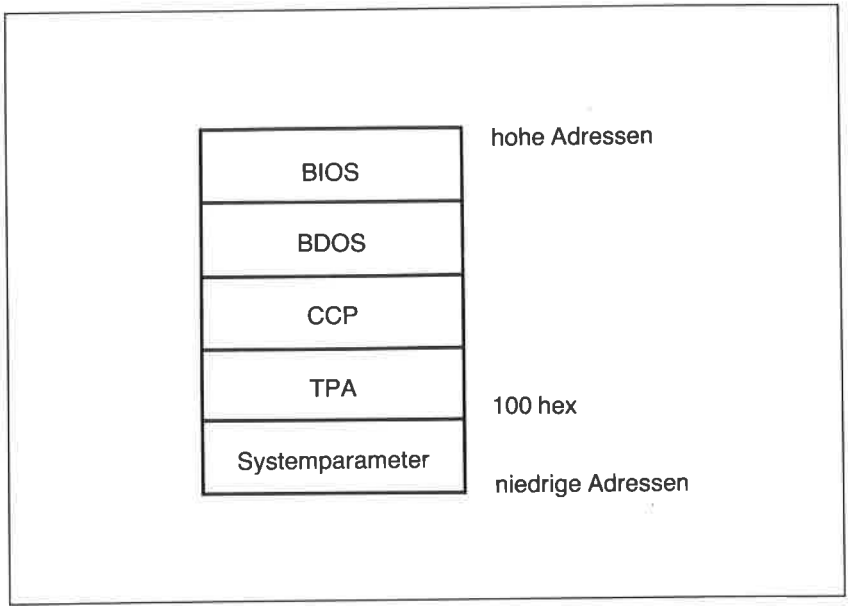


Abb. 1.1: Die Speichereinteilung von CP/M

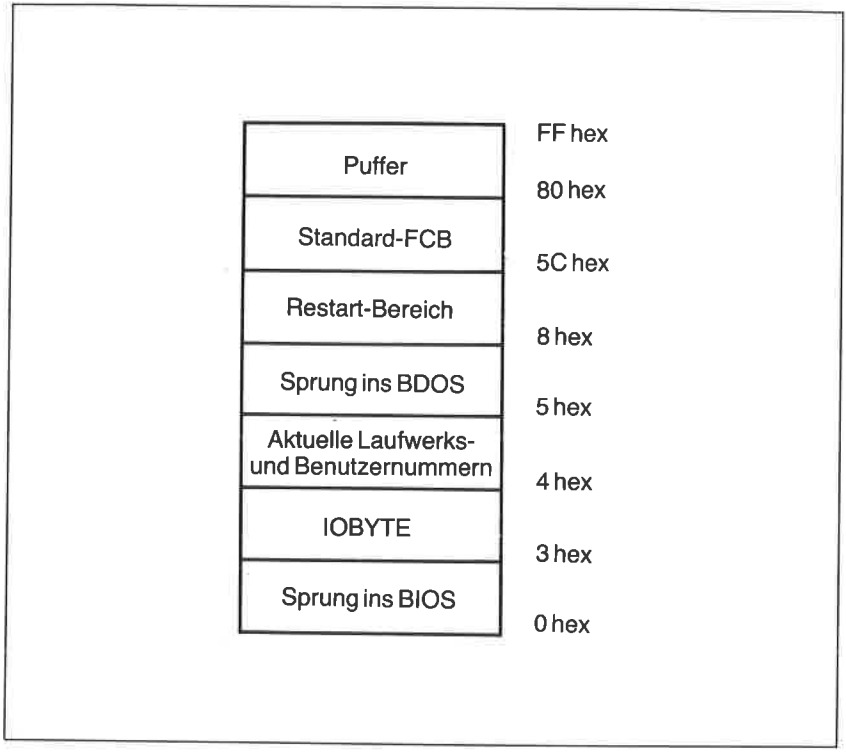


Abb. 1.2: Der Systemparameterbereich: 0 bis FF hex

Der Systemparameterbereich

Die Systemparameter, siehe Abb. 1.2, beginnen bei Adresse 0. Die ersten drei Bytes (Bytes 0-2) enthalten einen Sprung zum BIOS-Warmstartprogramm. Bei Ausführung dieses Befehls wird CP/M neu gestartet. Dadurch wird eine frische Kopie des CCP und des BDOS von der Diskette in den Speicher geladen. Die Diskettenlaufwerke werden in den Grundzustand gesetzt.

Das vierte Byte des Systemparameterbereiches (Adresse 3) nennt man IOBYTE. Es zeigt die laufende Gerätezuordnung für die Zusatzgeräte an: die Konsole, den Leser, den Stanzer und den Drucker. Der nächste Speicher, Adresse 4, enthält zwei Angaben: die momentane Laufwerksnummer und die momentane Anwendernummer. Ab Adresse 5 steht ein Sprungbefehl zum BDOS. Er wird ausgeführt, wenn Konsol-, Drucker- oder Disketten-Operationen gewünscht werden.

Der Bereich von Adresse 0 bis 38 umfaßt acht Speicher, auf die sich die 8080-Instruktionen RST0 bis RST7 beziehen:

Instruktion	*Hexa-Adresse*
RST0	0
RST1	8
RST2	10
RST3	18
RST4	20
RST5	28
RST6	30
RST7	38

Diese Befehle bewirken Unterprogrammaufrufe (CALLs) zu den entsprechenden Speicheradressen. Die Befehle können sowohl durch Hardware-Unterbrechungen als auch durch normale Unterprogrammaufrufe aktiviert werden. RST6 und RST7 werden von den Debuggern DDT und SID benutzt. Ausführung des Befehls RST0 führt einen Warmstart aus, da er einen Sprung zur Adresse 0 bewirkt.

Wenn ein Programm auf der Kommandoebene von CP/M ausgeführt wird, können ein oder zwei Dateinamen in der Kommandozeile angegeben werden. Mit dem Kommando EDIT könnte man z.B. zwei Parameter angeben:

```
EDIT FIRST.FIL B:SECOND.FIL
```

Den Bereich beginnend bei Adresse 5C hex nennt man Standard-Dateikontrollblock (DFCB), da der CCP diesen Bereich automatisch als Dateikontrollblock auswählt. Ein Dateikontrollblock ist ein 32-Byte langer Block, der eine Diskettendatei beschreibt. Der CCP nimmt den ersten Parameter, FIRST.FIL in diesem Beispiel, für den ersten FCB. Außerdem initialisiert der CCP einen zweiten FCB beginnend bei Adresse 6C hex. Der zweite Parameter, B:SECOND.FIL in diesem Beispiel, wird hierfür benutzt.

Der Bereich von 80 bis FF hex ist ein allgemeiner Pufferbereich. Der Rest des Kommandos, d.h. alle Zeichen nach dem Kommandonamen selbst, werden in diesem Bereich gespeichert. In dem obigen Beispiel sind das die beiden Dateinamen. Dieser Bereich wird standardmäßig auch für die Übertragung von Daten zu und von Disketten benutzt.

Der Systemparameterbereich wird weitergehend erläutert in Kapitel 3.

Die TPA und der CCP

Der Anwenderprogrammbereich umfaßt gewöhnlich den größten Teil des Speichers. Beginnend bei Adresse 100 hex werden hierhin ausführbare Programme geladen.

Der Kommandoprozessor enthält Befehle zur Verarbeitung von mit der Konsole eingegebenen Kommandos. Der Speicher, der vom CCP belegt wird, wird nach dem Start eines Programms nicht mehr benötigt. Daher können Anwenderprogramme den Bereich überlagern, um die TPA zu erweitern. Ein Warmstart nach Beendigung des Programms bewirkt das Neuladen des CCP zusammen mit dem BDOS.

Das BDOS und das BIOS

Das Basis-Disketten-Betriebssystem enthält geräteunabhängige Routinen zum Arbeiten mit Konsole, Drucker und Diskettenlaufwerk. Dieser Teil ist im allgemeinen gleich für alle CP/M-Computer. Wir studieren seine Arbeitsweise ausführlich in den Kapiteln 5, 6 und 7.

Das Basis-Ein-/Ausgabe-System enthält Instruktionen zum Bedienen der Zusatzgeräte: Konsole, Drucker, Telefonmodem, Diskette, usw. Jedes BIOS muß an die anwenderspezifischen Geräte angepaßt werden. Daher werden sich die BIOS für zwei identische Computer mit unterschiedlichen Zusatzgeräten unterscheiden. Mehr über das BIOS werden wir in den Kapiteln 2 und 3 erfahren.

DIE ARBEITSWEISE VON CP/M

Wenn CP/M das erste Mal gestartet wird (boot), werden der CCP, das BDOS und das BIOS von der Diskette in den Speicher kopiert (normalerweise Laufwerk A). Diesen Vorgang nennt man Kaltstart. Nachdem das System in den Speicher geladen ist, übergibt der Kaltstartlader die Kontrolle an das BIOS. Das BIOS füllt dann den Systemparameterbereich von Adresse 0 bis 7. Dies umfaßt den Sprungbefehl zum BIOS (Adresse 0 bis 2), das IOBYTE (Adresse 3), die momentane Laufwerks- und Anwendernummer (Adresse 4) und den Sprungbefehl zum BDOS (Adresse 5 bis 7). Jetzt gibt CP/M ein Promptzeichen aus, um seine Bereitschaft zum Empfang eines Kommandos anzuzeigen.

```
A>
```

Residente Kommandos

CP/M unterstützt maximal 16 Diskettenlaufwerke. Sie werden mit den ersten 16 Buchstaben des Alphabets (A bis P) bezeichnet. Der Buchstabe

A in der Anzeige zeigt Laufwerk A als laufendes oder Standard-Laufwerk an. Die in den CCP eingebauten Kommandos können jetzt ausgeführt werden. Die folgenden Kommandos sind eingebaut:

Kommando	*Funktion*
DIR	Ausgabe des Disketten-Inhaltsverzeichnisses
ERA	Löschen einer Datei
REN	Umbenennen einer Datei
SAVE	Erzeugen einer neuen Datei aus dem Speicher
TYPE	Anzeigen einer ASCII-Datei auf der Konsole
USER	Ändern der Anwendernummer

Diesen Namen darf kein Laufwerksname vorangestellt werden, weil sie sich nicht auf ein bestimmtes Laufwerk beziehen. So ist z.B. das Kommando

```
A:DIR
```

ungültig. Bei einigen dieser Kommandos kann man jedoch Laufwerksnamen als Parameter angeben. Z.B.

```
DIR A:
```

Das Kommando DIR wird gegeben, um eine Liste der Dateinamen auf dem Standardlaufwerk zu erhalten. Die Liste ist vierspaltig angeordnet. Alle Dateinamen werden angezeigt, wenn kein Parameter angegeben wird.

Eine ganze Gruppe von Dateien kann angewählt werden durch die Benutzung der Zeichen * oder ?. Z.B. zeigt das Kommando

```
DIR *.ASM
```

die Namen aller Dateien vom Typ ASM auf dem Standardlaufwerk an. Der Stern steht für beliebige Kombinationen von Zeichen, einschließlich Zwischenraum. Der Doppelstern *.* selektiert alle Dateien einer Diskette. In vielen CP/M-Kommandos kann der Stern als nichteindeutiger Dateiname angegeben werden. Das Fragezeichen ersetzt ein beliebiges einzelnes Zeichen, einschließlich Zwischenraum. Daher bezieht sich der Dateiname SORT?.BAS kollektiv auf die folgenden Dateien:

```
SORT1.BAS
SORT2.BAS
SORT3.BAS
SORT.BAS
```

Eine Datei oder eine Gruppe von Dateien kann mit dem Kommando ERA gelöscht werden. Die Zeichen * und ? dürfen im Parameter von ERA benutzt werden. (Vorsicht!). Z.B.:

```
ERA NEW.ASM
ERA *.ASM
ERA *.*
```

Im dritten Beispiel sollen alle Dateien gelöscht werden. In diesem Fall jedoch verlangt CP/M eine Bestätigung des Kommandos, bevor alle Dateien einer Diskette gelöscht werden:

```
ALL (Y/N)?
```

Man muß ein Y eingeben für die Fortsetzung des Kommandos.

Man kann das Kommando REN benutzen, um einzelne Dateien umzubenennen. REN erfordert zwei eindeutige Dateinamen; d.h. die Zeichen * und ? dürfen nicht benutzt werden. Der neue Dateiname wird zuerst angegeben, gefolgt von einem Gleichheitszeichen und dem alten Dateinamen. Z.B. ändert das Kommando

```
REN NEW.ASM=OLD.ASM
```

den Namen von OLD.ASM in NEW.ASM.

Das Kommando SAVE erzeugt eine Diskettendatei aus dem Speicherinhalt. SAVE hat zwei Parameter. Der erste Parameter ist die Anzahl der 256-Byte-Blöcke, die zu schreiben sind. Der zweite Parameter ist der Dateiname. Z.B. erzeugt das Kommando

```
SAVE 4 NEWFIL
```

eine Datei mit dem Namen NEWFIL, die das erste Kilobyte des Anwenderprogrammbereichs enthält.

Eine ASCII-Datei kann auf der Konsole angezeigt werden mit dem Kommando TYPE. Der einzige Parameter muß ein eindeutiger Dateiname sein. Das Durchlaufen des Bildes kann durch Eingabe von Control-S gestoppt werden. (TYPE wird beendet, wenn irgendeine andere Taste während der Ausgabe gedrückt wird.) Die Ausgabe wird durch Drücken irgendeiner Taste fortgesetzt, jedoch wird Control-S empfohlen, um das Kommando nicht unabsichtlich abzubrechen.

Die Anwendernummer kann mit dem Kommando USER geändert werden. Der einzige Parameter ist eine Dezimalzahl von 0 bis 15. CP/M

unterscheidet 16 verschiedene Anwender, numeriert von 0 bis 15. Anwender 0 ist normalerweise eingestellt nach der Initialisierung des Systems. Jede neu erzeugte Datei wird mit der laufenden Anwendernummer codiert. Daher ist eine Datei jeweils einem Anwender zugeordnet. Normalerweise sind nur die Dateien des laufenden Anwenders verfügbar.

Steuerzeichen

Einige Konsoltasten haben bei CP/M eine besondere Bedeutung. Es folgen die Steuerzeichenkommandos:

Kommando	*Funktion*
Control-C	Warmstart
Control-E	Start einer neuen Zeile
Control-H	Cursor ein Zeichen zurück
Control-I	Tabulator-Taste
Control-J	Zeile ausführen (Line feed)
Control-M	Zeile ausführen (Carriage return)
Control-P	Drucker an- oder abschalten (Schalter)
Control-R	Laufende Zeile erneut anzeigen
Control-S	Durchlaufen des Bildes stoppen
Control-U	Abbruch der laufenden Zeile (Start neue Zeile)
Control-X	Abbruch der laufenden Zeile (Wiederhole Zeile)

Warmstart

Ein Warmstart wird durchgeführt, wenn Control-C gedrückt wird und der Cursor in der ersten Position einer Zeile ist. Dies ist ähnlich einem Kaltstart. Der CCP und das BDOS werden von der Diskette in Laufwerk A in den Speicher kopiert. Die Sprungbefehle zu BIOS und BDOS am Beginn des Speichers werden auch initialisiert, jedoch wird das Speicherabbild vom BIOS nicht verändert. Das aktuelle Diskettenlaufwerk und Laufwerk A werden zu diesem Zeitpunkt notiert.

Wenn CP/M das erste Mal auf ein Laufwerk zugreift, macht es eine Kopie des Inhaltsverzeichnisses und einiger Charakteristika der Diskette. Dieses Vorgehen bei Disketten kann man beobachten, wenn man nacheinander alle Laufwerke anspricht. Gibt man z.B. das Kommando

```
B:
```

so wird auf Laufwerk B umgeschaltet. Dies wird gewöhnlich durch eine rote Lampe an der Laufwerksvorderseite angezeigt. Das System-Prompt-

zeichen ändert sich in B>. Wer mehr Laufwerke hat, kann der Reihe nach alle ansprechen durch den Namen, gefolgt von einem Doppelpunkt. Wenn man mit dem Kommando

```
A:
```

zu Laufwerk A zurückgelangt, wird sich das System-Promptzeichen in A> ändern. Jedoch wird man keine Laufwerksaktivität bemerken, da das Laufwerk A bereits notiert ist. CP/M nimmt an, daß die Diskette zwischenzeitlich nicht gewechselt wurde.

Das Inhaltsverzeichnis wird bei weiteren Zugriffen nicht erneut eingelesen. Deshalb sollte man nach dem Auswechseln einer Diskette in einem Laufwerk einen Warmstart durchführen mit dem Kommando Control-C. Hierdurch wird das Inhaltsverzeichnis der neuen Diskette gelesen. Wird nach einem Diskettenwechsel kein Warmstart gemacht, kann CP/M die Diskette zwar vielleicht noch lesen, versucht man jedoch, auf diese Diskette zu schreiben, wird CP/M die Schreiboperation abweisen und eine Fehlermeldung ausgeben:

```
BDOS ERROR ON A: R/O
```

CP/M wird dann auf das Drücken der Return-Taste warten. Danach wird automatisch ein Warmstart ausgeführt, obwohl nicht Control-C gedrückt wurde. Da das neue Disketten-Inhaltsverzeichnis nun eingelesen wird, kann man dann auf die Diskette schreiben.

Nicht residente Programme

Die Anzahl der in den CCP eingebauten Befehle ist begrenzt. Daher besteht zusätzlich die Möglichkeit, separate Programme zu laden, die als COM-Dateien auf einer der Disketten gespeichert sind. Man startet diese Programme durch Eingabe des Laufwerksnamens und des Dateinamens (ohne den Dateityp COM). Der Laufwerksname darf fehlen, wenn das Programm auf dem Standard-Laufwerk ist. Wenn der Name eines Anwenderprogramms eingegeben wird, liest CP/M die Datei in den Speicher und startet es.

Auf der Diskette gespeicherte Programme werden durch den Dateinamen angesprochen. Ein CP/M-Dateiname besteht aus einem Primärnamen und einer Erweiterung. Der Primärname besteht aus ein bis acht alphabetischen oder numerischen (alphanumerischen) Zeichen. Andere Zeichen als die Buchstaben A bis Z und die Ziffern 0 bis 9 können in gewissen Fällen verwendet werden, aber im Zweifel ist es besser, sie zu vermeiden. Bei einigen Anwendungen ist ein Dateityp bzw. eine Dateina-

menerweiterung erforderlich, bei anderen nicht. Ist eine Erweiterung erforderlich, besteht sie aus ein bis drei Zeichen. Der Dateityp ist gewöhnlich eine mnemotechnische Bezeichnung, um die Art der Datei zu beschreiben. Z.B.:

Erweiterung	*Bedeutung*
ASM	Assembler-Quelldatei
COM	Ausführbares Programm
HEX	Hexadezimal-Datei
BAK	Backup-Datei
BAS	BASIC-Quelldatei
FOR	FORTRAN-Quelldatei
PAS	Pascal-Quelldatei
REL	Verschiebbare Binärdatei
ASC	ASCII Text-Datei
LST	ASCII Druck-Datei

Einige nicht residente Programme werden mit dem Betriebssystem CP/M mitgeliefert. Dies sind unabhängige ausführbare Programme vom Dateityp COM:

Dateiname	*Programmfunktion*
ASM	Assembler
DDT	Debugger
DUMP	Anzeige von ausführbaren Programmen
ED	Editor
LOAD	Konvertiert HEX-Dateien in COM-Dateien
MOVCPM	Ändern der Größe von CP/M
PIP	Kopieren von Dateien
STAT	Detaillierte Ausgabe des Inhaltsverzeichnisses
SYSGEN	Kopieren der Systemspuren auf andere Disketten
SUBMIT	Ausführen einer Gruppe von Kommandos
XSUB	Erweiterung von SUBMIT

Die folgenden Programme kann man käuflich erwerben:

Dateiname	*Programmfunktion*
BADLIM	Isolieren von defekten Disketten-Sektoren
COPY	Spurweises Kopieren
FILEFIX	Regenerieren gelöschter Dateien
FORMAT	Formatieren von Disketten

MAC	Makroassembler von Digital Research
MACRO-80	Makroassembler von Microsoft
MBASIC	Microsoft-BASIC
SAVEUSER	Lifeboat Programm zum Speichern des BIOS
SID	Symbolischer Debugger von Digital Research
WS	WordStar Text-Editor

Viele andere Programme kann man erwerben oder schreiben. Wir werden in diesem Buch viele nützliche Programme entwickeln.

EIN ERSTES AUSFÜHRBARES PROGRAMM

Wir wollen mit einem simplen Programm beginnen. Der CP/M-Kaltstart wird gegebenenfalls durchgeführt. Wenn dies abgeschlossen ist, wird das Promptzeichen

```
A>
```

auf der Konsole erscheinen. Mit dem eingebauten Kommando DIR wird geprüft, welche Programme auf der Diskette in Laufwerk A sind. Falls das Programm STAT.COM in der Liste erscheint, führen wir es mit

```
STAT *.*
```

aus. Ähnlich DIR produziert STAT eine Liste aller Dateien auf der Diskette. Darüber hinaus sind die Namen alphabetisch sortiert.

Wenn Sie einen Drucker haben, schalten Sie ihn ein. Drücken Sic Control-P (um die Konsolausgabe zum Drucker zu schicken), und wiederholen Sie das Kommando STAT. Auf dem Drucker erscheint dieselbe Liste wie auf der Konsole. Drücken Sie erneut Control-P, um den Drucker wieder abzuschalten. Reißen Sie das Papier am Drucker ab, und legen Sie die Liste für späteren Gebrauch in den Umschlag der Diskette.

STAT gibt noch weitere Informationen über die Dateien der Diskette aus. Betrachten Sie z.B. die folgenden Zeilen:

```
 58    8K 1 R/O A:PIP.COM
266   34K 3 R/W A:WSOVLY1.OVR
```

Die erste Zeile zeigt, daß die Datei PIP.COM auf dem Laufwerk A ist. Die Datei kann nur gelesen werden (angezeigt durch R/O); d.h. sie ist schreibgeschützt. Des weiteren besteht das Programm aus 58 (128-Byte-) Sätzen, die in 8K Bytes gespeichert sind. Die Datei besteht aus einem physischen Bereich.

Kleinere Dateien können durch einen einzigen Verzeichniseintrag angesprochen werden, den man physischen Bereich nennt (ein 16K-Speicherblock auf der Diskette). Wird mehr als ein Bereich für eine Datei benötigt, haben alle Bereiche denselben Dateinamen. Jedoch nur einer dieser Einträge wird mit DIR oder STAT angezeigt.

Die zweite Datei im obigen Beispiel, WSOVLY1.OVR, ist auch auf der Diskette in Laufwerk A. Sie kann sowohl gelesen als auch beschrieben werden (angezeigt durch R/W); sie ist nicht schreibgeschützt. Sie besteht aus 266 Sätzen, gespeichert in 34K Bytes, und erfordert 3 physische Bereiche. Am Ende der STAT-Liste wird der noch auf der Diskette verfügbare Speicherplatz ausgewiesen.

Oft ist es wünschenswert, ein bereits ausgeführtes Kommando nach einem Warmstart erneut auszuführen. Wir werden uns eine Methode dafür schaffen. Führen Sie das eingebaute Kommando

```
SAVE 0 CONTIN.COM
```

aus. Dies erzeugt einen neuen Eintrag im Verzeichnis auf der aktuellen Diskette. Da jedoch die Größe der Datei null ist, werden tatsächlich keine Daten geschrieben. Wenn Sie erneut STAT aufrufen, sollte der auf der Diskette verbleibende Platz derselbe sein wie vorher. Die Liste zeigt einen neuen Eintrag, CONTIN.COM, mit der Länge null. Wir werden herausfinden, daß dieses leere „Programm" sehr nützlich ist.

Jedesmal, wenn man CONTIN eingibt, wird CP/M versuchen, das entsprechende Programm zu laden, CONTIN.COM, und dann zum Beginn der TPA bei 100 hex zu verzweigen. Da das Programm CONTIN keine Daten hat, wird dieses Kommando nur das zuletzt benutzte Programm starten. Um dies zu zeigen, geben Sie das Kommando

```
PIP
```

ein. CP/M wird aufgefordert, PIP.COM zu laden und nach 100 hex zu springen. PIP antwortet mit einem Stern. Normalerweise würde man jetzt ein PIP-Kommando eingeben. Aber in diesem Fall drücken Sie nur RETURN. Dies wird PIP beenden und die Kontrolle an CP/M abgeben. CP/M führt einen Warmstart aus, gibt das Promptzeichen aus und erwartet wieder ein Kommando. Nun geben Sie

```
CONTIN
```

ein. Da dieses Programm keine Daten hat, wird lediglich nach Adresse 100 hex gesprungen, dem Anfang von PIP. Da das Speicherbild von PIP

nicht verändert worden ist, sollte der PIP-Stern wieder erscheinen. Man kann dies durch ein PIP-Kommando nachweisen. Z.B. geben Sie

```
PIP2.COM=PIP.COM[V]
```

ein. PIP wird sich selbst kopieren in die neue Datei PIP2.COM. Der Parameter V in eckigen Klammern bewirkt, daß PIP die Kopie auf Richtigkeit überprüft.

Diese Technik ist anwendbar auf viele, jedoch nicht alle Programme. Es wird z.B. mit STAT nicht gehen, da Datenbereiche bei einem Neustart nicht ordnungsgemäß initialisiert werden. Es ist jedoch anwendbar bei MBASIC.

Lassen Sie uns dieses Phänomen bei Microsoft's BASIC untersuchen. Rufen Sie BASIC auf, indem Sie seinen Namen eingeben. Dann schreiben Sie folgendes BASIC-Programm:

```
10 FOR I = 1 TO 9
20 PRINT I; I*I, 1/I, SQR(I)
30 NEXT I
40 END
```

Testen Sie das Programm mit dem Kommando RUN. Das Programm existiert nur im Arbeitsspeicher, deshalb geht es verloren, wenn Sie BASIC verlassen. Daher wird man normalerweise eine dauerhafte Kopie herstellen mit dem BASIC-Kommando

```
SAVE "FIRST"
```

Nehmen wir nun einmal an, Sie haben versehentlich das BASIC-Kommando SYSTEM gegeben, bevor Sie das Programm gesichert haben (versuchen Sie es). Sie befinden sich nun wieder auf der CP/M-Systemebene. Offensichtlich haben Sie Ihr BASIC-Programm verloren. Geben Sie nun das Kommando

```
CONTIN
```

ein. Sie werden wieder ins BASIC gelangen und zu dem Programm, das Sie geschrieben haben. Geben Sie das Kommando LIST ein, und Sie werden sehen, daß das Programm noch existiert. Geben Sie dann RUN ein, zur Überprüfung, ob es noch richtig läuft. Sie können nun das SAVE Kommando geben, wenn Sie Ihr Original-BASIC-Programm sichern wollen.

ZUSAMMENFASSUNG

In diesem Kapitel haben wir einen kurzen Überblick über die Grundlagen der Organisation und Arbeitsweise von CP/M erhalten. Eingeschlossen waren die Behandlung des Systemparameterbereichs, der TPA, des CCP, des BDOS und des BIOS. Die eingebauten Kommandos, die Steuerzeichen und einige Standardprogramme wurden betrachtet. Dann schrieben wir das „Programm" CONTIN, das benutzt werden kann, um die meisten Programme erneut zu starten.

Im nächsten Kapitel werden wir erfahren, wie man die Diskettenversion des CP/M-Betriebssystems kopiert und ändert.

Kapitel 2

Duplizieren und Ändern von CP/M-Disketten

EINFÜHRUNG

In Kapitel 1 haben wir die grundlegende Arbeitsweise von CP/M studiert und erfahren, wie der Speicher organisiert ist. Da CP/M ein Disketten-Betriebssystem ist, spielt die Diskette eine wichtige Rolle bei der Arbeitsweise. Lassen Sie uns daher in diesem Kapitel unsere Aufmerksamkeit verstärkt auf die Organisation der Diskette richten.

In diesem Kapitel werden wir lernen, wie man eine CP/M-Diskette dupliziert: durch Formatieren einer neuen Diskette und Kopieren der Daten und der Systemspuren. Danach werden wir herausfinden, wie man BIOS- oder USER-Routinen des CP/M-Systems ändert, wie man die neue Version übersetzt und testet und schließlich eine Kopie der neuen Version auf die Diskette schreibt.

DAS FORMATIEREN UND DUPLIZIEREN VON DISKETTEN

Das Diskettenlaufwerk ist eines der wichtigsten Geräte zum Speichern von Mikrocomputerinformationen. Die Oberfläche der Diskette ist ein magnetisches Material, das gelesen und beschrieben wird mit einem Lese-/Schreibkopf. (Die Wirkungsweise ist zu vergleichen mit der Tonaufnahme auf Magnetbändern.) Physikalisch betrachtet werden Disketten formatiert in konzentrischen Ringen, die man Spuren nennt. Jede Spur ist aufgeteilt in Teile, die man Sektoren nennt.

Es ist üblich, die CP/M-Systemdiskette mit den ausführbaren Programmen in das Laufwerk A einzusetzen, und eine Arbeitsdiskette in das Laufwerk B. Informationen kann man sicher auf Disketten speichern, vorausgesetzt, einige Sicherheitsmaßregeln werden beachtet. So sollten Disketten nicht in der Nähe von magnetischen Feldern oder in staubiger Umgebung aufbewahrt werden. Sogar bei großer Vorsicht kann ein Stromausfall während einer Schreiboperation zu Datenverlust führen. Deshalb ist es ratsam, eine Sicherungskopie von allen wichtigen Disketten anzufertigen. Wir wollen nun mehrere Methoden zum Duplizieren von Disketten untersuchen.

Neue Disketten muß man vor dem Gebrauch formatieren. Es gibt zwei gebräuchliche Diskettengrößen, 5 Zoll und 8 Zoll Durchmesser. Außerdem gibt es Disketten, die hart- und solche, die soft-sektoriert sind, mit einfacher oder doppelter Schreibdichte, einseitig oder zweiseitig. Die Anzahl der Spuren ist ebenfalls variabel.

Wenn Sie Disketten kaufen, müssen Sie den richtigen Durchmesser (5 Zoll oder 8 Zoll) und das richtige Format (hart- oder soft-sektoriert) wählen. Falls Sie hart-sektorierte benötigen, müssen Sie auch die richtige Anzahl der Sektoren pro Spur auswählen. Es gibt viele verschiedene Diskettenformate für einen bestimmten Diskettentyp. Es ist daher gewöhnlich erforderlich, jede Diskette zu formatieren, bevor sie das erste Mal benutzt werden kann.

Das Formatieren einer neuen Diskette

Disketten werden mit einem Programm formatiert, dessen Name FORMAT.COM, FORMAT5.COM oder FORMAT8.COM ist. Es sollte sich auf Ihrer Original-CP/M-Diskette befinden. Formatierprogramme sind auf den jeweils benutzten Diskettentyp zugeschnitten. Versuchen Sie nicht, eine Diskette mit einem Programm von einem fremden Computer zu formatieren, da es wahrscheinlich nicht richtig arbeitet.

Wenn Sie eine Diskette das erste Mal benutzen, stecken Sie sie zum Formatieren in das Laufwerk B. In Kapitel 1 sahen wir, daß wir nach einem Diskettenwechsel einen Warmstart ausführen sollten. Dies hier ist jedoch eine Ausnahme. Nach dem Einsetzen einer neuen, unformatierten Diskette führen Sie keinen Warmstart aus.

Falls A nicht das Standard-Laufwerk ist, geben Sie das Kommando 'A:', so daß A nun das Standard-Laufwerk ist. Versichern Sie sich, daß die Diskette A das Formatierprogramm enthält. Starten Sie es durch Eingabe seines Namens. Sie werden während der Programmausführung einige Fragen beantworten müssen. So wird gefragt, ob die Diskette in einfacher oder doppelter Schreibdichte, einseitig oder zweiseitig formatiert werden soll. Einige Systeme finden dies automatisch heraus, so daß der Dialog entfallen kann.

Wenn Sie eine neue Diskettenpackung anbrechen, ist es zweckmäßig, alle Disketten zu formatieren. Das FORMAT-Programm unterstützt dies gewöhnlich. Nach dem Formatieren einer Diskette wechseln Sie sie gegen die nächste frische Diskette aus und drücken RETURN. Dadurch wird der Formatiervorgang normalerweise wiederholt. Denken Sie daran, daß Sie nicht auf eine nichtformatierte Diskette schreiben können, sonst bekommen Sie eine BDOS-Fehlermeldung. Im nächsten Abschnitt

betrachten wir eine allgemeine Technik, um mit SYSGEN und PIP Disketten zu kopieren.

Das Duplizieren einer Diskette mit SYSGEN und PIP

CP/M-Disketten sind in zwei Bereiche aufgeteilt. Man bezeichnet sie als Systemspuren und Datenspuren. Die Systemspuren umfassen das CP/M-Betriebssystem, inklusive CCP, BDOS und BIOS. Zu diesem Bereich hat man gewöhnlich keinen Zugriff. Die Datenspuren sind aufgeteilt in die Bereiche für das Inhaltsverzeichnis und für die Speicherung von Programmen.

Da die Systemspuren normalerweise für den Benutzer nicht verfügbar sind, wird das Kommando

```
ERA *.*
```

alle normalen Benutzerdateien in den Datenspuren der Diskette löschen, nicht aber die Systemspuren.

Man benötigt jedoch einen Zugriff zu den Systemspuren, um eine Systemdiskette zu kopieren oder um das BIOS zu ändern. Nachdem eine neue Diskette formatiert wurde, können alle normalen Dateien mit PIP von der Original-Diskette auf die neue Diskette kopiert werden. Befindet sich die Original-Diskette in Laufwerk A und die neue Diskette in Laufwerk B, geben Sie das Kommando

```
PIP B:=A:*.*[V]
```

Danach enthält die neue Diskette alle Dateien der Original-Diskette. Sie kann nun im Laufwerk B benutzt werden, aber noch nicht in Laufwerk A. Es sind nämlich die Systemspuren, die den CCP, das BDOS und das BIOS enthalten, noch nicht auf die neue Diskette übertragen worden.

Ein Programm namens SYSGEN kann benutzt werden, um die Systemspuren von einer Diskette auf eine andere zu übertragen. Jedoch kann SYSGEN dies nicht auf direktem Wege tun. Es muß die Systemspuren erst von der Systemdiskette in den Speicher lesen. Dann kann es das Speicherabbild auf die Systemspuren einer anderen Diskette schreiben. Wir wollen nun sehen, wie das geschieht.

Starten Sie SYSGEN, und folgen Sie den Anweisungen. Es können von Version zu Version geringfügige Unterschiede auftreten, der Sinn ist jedoch derselbe. Wenn Sie SYSGEN gestartet haben, mag die folgende Ausgabe erscheinen:

```
SYSGEN Version 3.0
Distributed by Lifeboat Associates
for CP/M 2 on quad North Star.
Source drive NAME (or RETURN to skip)
```

Geben Sie den Namen des Quell-Laufwerks an, aber ohne den Doppelpunkt. Dies ist das Laufwerk, in dem sich die Original-Diskette befindet,

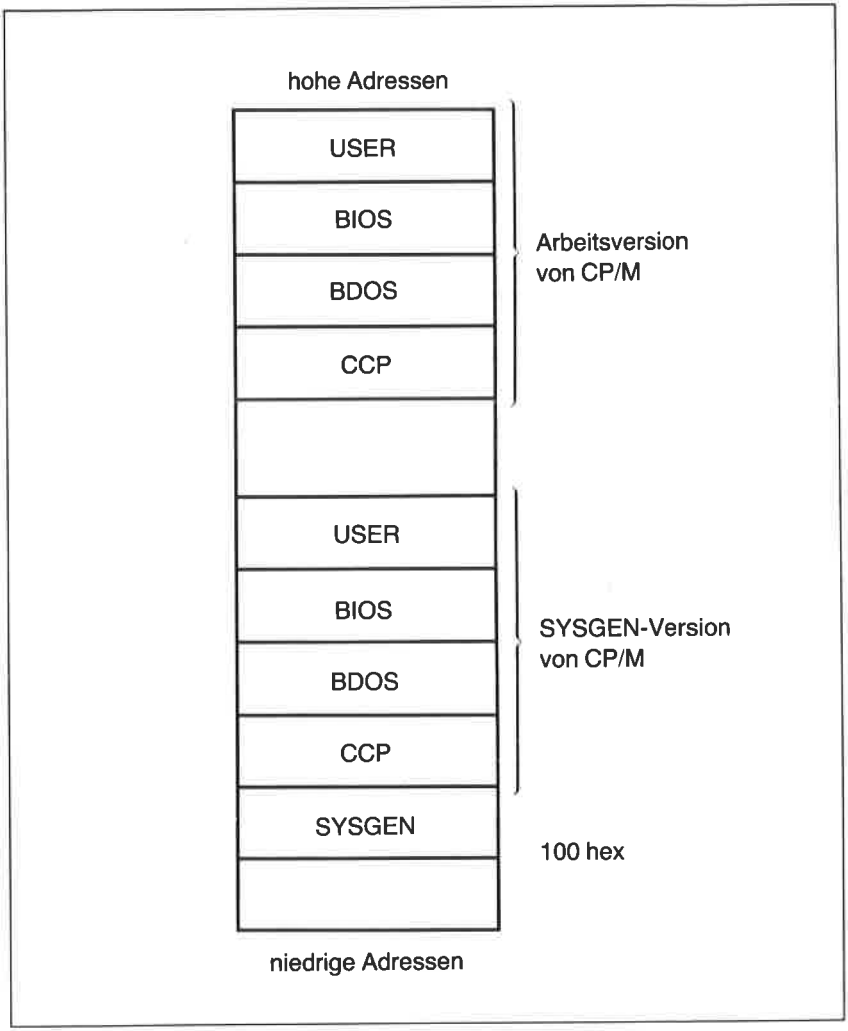


Abb. 2.1: Die SYSGEN-Version und die Arbeitsversion von CP/M

normalerweise Laufwerk A. SYSGEN bestätigt Ihre Antwort und verlangt ein weiteres RETURN. Haben Sie z.B. A eingegeben, könnte die folgende Zeile erscheinen:

```
Place Source disk on A, then type RETURN
```

Nach Eingabe des zweiten RETURN erscheint die Ausgabe:

```
Function complete
CP/M image in RAM at 900H is ready to write
or reboot and "SAVE 40 CPMxx.COM"
Destination drive NAME (or RETURN to reboot)
```

In diesem Schritt kopiert SYSGEN die CP/M-Systemspuren der Original-Diskette in den Speicher. Es sind nun zwei Kopien von CP/M im Speicher (siehe Abb. 2.1). Die Arbeitsversion am Speicherende und die SYSGEN-Version am Anfang hinter SYSGEN selbst.

Im nächsten Schritt wird die SYSGEN-Version auf die Systemspuren der Diskette geschrieben. Dazu muß SYSGEN allerdings wissen, wohin (auf welches Laufwerk) das System geschrieben werden soll. Geben Sie den Namen des Laufwerks ein, in dem sich die neue Diskette befindet. Das ist gewöhnlich Laufwerk B. Auch hier lassen Sie den Doppelpunkt weg. SYSGEN antwortet mit:

```
Place destination disk on B, then type RETURN
```

Nach Eingabe von RETURN kopiert SYSGEN das System auf die Systemspuren der neuen Diskette. Damit ist der Vorgang beendet. SYSGEN gibt dann die Zeilen:

```
Function complete
Destination drive NAME (or RETURN to reboot)
```

aus. Zu diesem Zeitpunkt können Sie eine Kopie von CP/M auf weitere formatierte Disketten schreiben. Entfernen Sie die neue Diskette, und setzen Sie eine weitere formatierte Diskette in das Laufwerk B. Drücken Sie die Taste B und RETURN. SYSGEN bestätigt dann das gewünschte Laufwerk und verlangt erneut RETURN. Dann kopiert SYSGEN das System aus dem Speicher auf die Systemspuren dieser Diskette. Auf diese Weise können Sie nach und nach leicht die Systemspuren mehrerer Disketten beschreiben. Wünschen Sie nur eine einzige Kopie, dann können Sie RETURN eingeben, und das Programm beendet sich. Im nächsten Abschnitt werden wir eine andere Methode zum Duplizieren von Disketten kennenlernen.

Duplizieren einer Diskette mit COPY

Im letzten Abschnitt benutzten wir PIP und SYSGEN, um ein Duplikat einer Diskette anzufertigen. In diesem Abschnitt werden wir eine zweite Methode betrachten, die einfacher und schneller ist. Hierfür braucht man jedoch ein Programm, das standardmäßig nicht vorhanden ist. Sein Name ist COPY. Prüfen Sie, ob Sie ein Programm namens COPY.COM, COPY5.COM oder COPY8.COM haben. Normalerweise kann ein solches Programm alle drei Dinge, wie Formatieren einer neuen Diskette, Kopieren der Systemspuren und Kopieren der Datenspuren in einem Zuge durchführen.

Legen Sie die Original-Diskette in das Laufwerk A und eine neue unformatierte Diskette in das Laufwerk B. Achten Sie darauf, daß Sie keinen Warmstart ausführen. Starten Sie das Programm COPY, und folgen Sie seinen Anweisungen. Beantworten Sie die Fragen nach den Laufwerken für Original und Kopie. In diesem Beispiel ist das Original in Laufwerk A und die Kopie in Laufwerk B. Dies kann auch die Standardannahme sein. Bevor Sie das endgültige RETURN eingeben, können Sie die Diskette in Laufwerk A auswechseln, wenn Sie eine Kopie von einer anderen Diskette wünschen.

Wenn dies erfolgreich war, haben Sie eine bequeme Methode zum Duplizieren von Disketten gefunden. Wenn Ihre Version von COPY jedoch eine formatierte Diskette erfordert, wird dieser Versuch scheitern. Trotzdem ist dies ein bequemer Weg, um Disketten zu kopieren, obwohl Sie die neue Diskette separat formatieren müssen, denn Sie können immerhin die System- und Datenspuren mit dem COPY-Programm kopieren. Wir werden nun sehen, wie man die Informationen auf den Systemspuren ändert.

ALLGEMEINES VORGEHEN ZUM ÄNDERN DES BIOS

In dem vorigen Beispiel in diesem Kapitel sahen wir eine Methode zum Kopieren einer Diskette, einschließlich der Systemspuren. Im nächsten Abschnitt werden wir die Möglichkeit haben, Teile von CP/M auf diesen Systemspuren zu ändern. Dies ist eine lästige Prozedur, denn zu den Systemspuren haben wir normalerweise keinen Zugriff. Daher stellen wir im Rest dieses Kapitels drei Wege vor, die CP/M-Systemspuren zu untersuchen. (Machen Sie sicherheitshalber eine Kopie der Systemdiskette, und arbeiten Sie mit der Kopie.)

Die Überarbeitung, die wir im nächsten Abschnitt durchführen, bezieht sich auf das BIOS von CP/M. Für die Änderungen benötigen wir ein Assemblerquellprogramm namens BIOS.ASM, BIOS.MAC, USER.ASM

oder USER.MAC. Eines dieser Programme sollte auf Ihrer Original-CP/M-Diskette vorhanden sein. Nach der Änderung des BIOS-Quellprogramms werden wir es übersetzen und damit die Arbeitsversion des BIOS überschreiben. Wenn wir mit der neuen Version zufrieden sind, werden wir eine permanente Kopie davon auf den Systemspuren der Diskette machen.

Obwohl wir in diesem Kapitel das BIOS nicht tatsächlich ändern, werden wir doch alle nötigen Schritte zeigen, um das geänderte BIOS in das CP/M-System zu speichern, das neue BIOS zu testen und auf die Systemspuren der Diskette zu schreiben. Das sind die folgenden Schritte:

1. Ändern der BIOS.ASM- oder USER.ASM-Quelle
2. Erzeugen der entsprechenden HEX- oder REL-Datei mit dem Assembler
3. Kopieren der HEX- oder REL-Datei an die richtige Stelle mit DDT oder SID
4. Austesten der neuen Funktionen
5. Kopieren der neuen Version auf die Systemspuren

Zunächst werden wir die Original-Version des BIOS übersetzen. Dann werden wir sie im Speicher installieren. Und endlich die „neue" Version auf die Systemspuren einer Diskette schreiben. Da wir das Original-BIOS nicht verändert haben, sollten wir keinen Unterschied bei unserem CP/M bemerken. Der Grund für diesen Schritt ist, zu lernen, wie man eine geänderte Version des BIOS testet und permanent auf die Systemdiskette schreibt. Als erstes müssen wir die Lage des BIOS im Speicher bestimmen.

DAS LOKALISIEREN DES BIOS

Wir sahen in Kapitel 1, daß das BIOS im CP/M die maßgeschneiderten Routinen für die verschiedenen an den Computer angeschlossenen Zusatzgeräte enthält. Die Routinen sind von Computer zu Computer verschieden. Der Rest von CP/M, wie das BDOS und der CCP, ist universell, d.h. unabhängig von einem bestimmten Computer. Dafür gibt es eine Reihe von Sprungbefehlen, die man Vektoren nennt, am Anfang des BIOS, die zu den wichtigen Routinen innerhalb des BIOS verzweigen. Daher ist es möglich, BIOS-Befehle zu ändern, ohne daß der Rest von CP/M davon betroffen ist.

Manchmal liegen diese Routinen in einem Extraspeicherbereich, dem USER-Bereich, einer Untermenge des BIOS. In jedem Fall ist eine permanente Kopie dieser Routinen auf den Systemspuren und eine tempo-

räre Kopie davon im Speicher. Wir können Änderungen dieser Routinen im Speicher machen, um zu sehen, ob die neue Version das tut, was wir wollen. Haben wir herausgefunden, daß die Änderungen korrekt sind, müssen wir eine permanente Kopie der neuen Version auf die Systemspuren in Laufwerk A schreiben.

Wir wollen nun die Lage des laufenden BIOS mit Hilfe des Debuggers bestimmen: die des BIOS am Ende des Speichers. Der Sprungbefehl am Beginn des Speichers bestimmt die Warmstart-Adresse im BIOS. Starten Sie den Debugger, und geben Sie das Kommando

```
L0 (Buchstabe L und eine Null)
```

ein. Der Buchstabe L bedeutet List. Dieses Kommando wird benutzt, um 8080-Befehle zu dissamblieren, d.h. sie in mnemotechnischer Form auszugeben. Die erste Zeile der Ausgabe könnte

```
JMP D303
```

sein. Das Symbol JMP ist der 8080-Name für einen unbedingten Sprungbefehl, und D303 ist der Operand, das Ziel des Sprunges. Dieser Befehl zeigt auf den Warmstarteingang im BIOS. Jedoch suchen wir den Kaltstarteingang bei Adresse D300 hex in diesem Beispiel.

Im nächsten Schritt untersuchen wir den laufenden BIOS-Bereich im Speicher. Geben Sie nun das Kommando

```
LD300
```

ein. Die Ausgabe wird nun eine Reihe von Sprungbefehlen sein. Z.B.:

```
D300 JMP D380 (Kaltstart-Adresse)
D303 JMP D39F (Warmstart-Adresse)
D306 JMP DA06 (Konsol-Status)
D309 JMP DA09 (Konsol-Eingabe)
D30C JMP D4E6 (Konsol-Ausgabe)
D30F JMP DA0F (Drucker-Ausgabe)
D312 JMP DA12 (Stanzer-Ausgabe)
D315 JMP DA15 (Leser-Eingabe)
D318 JMP D4CA (Anfang der Disketten-Routinen)
D31B JMP D499
D31E JMP D4CC
```

Nun müssen wir feststellen, ob ein separater USER-Bereich zusätzlich zum regulären BIOS vorhanden ist. Ist nur ein BIOS-Bereich vorhanden, dann zeigen alle Sprungvektoren zu nahe beieinander liegenden Adressen, d.h. innerhalb eines Bereiches von 800 hex Bytes. Beachten Sie, daß in der obigen Liste die ersten beiden Vektoren zu Adressen nahe dem Beginn des BIOS verzweigen (D380 und D39F hex). Mehrere andere jedoch zeigen auf weiter entfernte Adressen (DA06, DA09 usw.).

Lassen Sie uns diesen zweiten Bereich mit dem Debugger untersuchen. Nach Eingabe des Kommandos

```
LDA00
```

könnte folgende Ausgabe entstehen:

```
DA00 JMP DA1B    (Kaltstart-Adresse)
DA03 JMP DA41    (Warmstart-Adresse)
DA06 JMP DA7E    (Konsol-Status)
DA09 JMP DA96    (Konsol-Eingabe)
DA0C JMP DAB6    (Konsol-Ausgabe)
DA0F JMP DACC    (Drucker-Ausgabe)
DA12 JMP DB8B    (Stanzer-Ausgabe)
DA15 JMP DBD6    (Leser-Eingabe)
```

Wir haben einen weiteren Satz von Sprungvektoren gefunden. In diesem Fall beziehen sich alle Vektoren unmittelbar auf den Speicher. Wir haben den Hilfsbereich namens USER-Bereich gefunden, eine Untermenge der BIOS Routinen.

Beachten Sie die Eins-zu-Eins-Korrespondenz zwischen vielen Vektoren des BIOS und den entsprechenden im USER-Bereich. So zeigen einige der BIOS-Vektoren relativ zu derselben Position in dem anderen Bereich. Die Adresse D306, z.B., enthält einen Sprung nach DA06. Eine auffällige Ausnahme in der obigen Liste ist der Konsol-Ausgabevektor bei Adresse D30C. Er zeigt auf die Adresse D4E6. Verfolgt man es jedoch mit dem Debugger, findet man schließlich die entsprechende Adresse DA0C. Disketten-Routinen liegen normalerweise nicht im USER-Bereich, daher erwarten wir auch keine Sprungbefehle mehr ab XX15 hex.

Es ist wichtig, herauszufinden, ob Ihr System einen USER-Bereich hat. Hat es nämlich keinen, werden wir die Änderungen im BIOS-Bereich unter Benutzung eines Quellprogramms namens BIOS.ASM oder BIOS.MAC machen. Gibt es aber einen USER-Bereich, dann führen wir die Änderungen hier durch. Die Quelle heißt dann USER.ASM oder USER.MAC.

DAS ASSEMBLIEREN DES BIOS- ODER USER-QUELLPROGRAMMS

In diesem Abschnitt werden wir das Original-Quellprogramm für die BIOS- bzw. USER-Routinen übersetzen. (Falls Sie dieses Quellprogramm nicht finden, können Sie die Änderungen, die wir in Kapitel 3 besprechen, nicht durchführen.) Wir werden dann den übersetzten Code mit dem von CP/M benutzten vergleichen. Dadurch können wir sicherstellen, daß Ihr Quellprogramm mit dem von CP/M benutzten übereinstimmt.

Übersetzung des BIOS- bzw. USER-Quellprogramms mit Digital Research MAC

Suchen Sie auf Ihrer Original-CP/M-Diskette nach einem Programm mit dem Namen BIOS.ASM oder USER.ASM, und kopieren Sie es auf eine Arbeitsdiskette. Sehen Sie sich den Anfang dieses Programms mit dem Kommando TYPE oder mit dem Editor an. Finden Sie die ORG-Anweisung, die die Anfangsadresse von BIOS oder USER bestimmt. Stellen Sie sicher, daß die Adresse mit dem Wert übereinstimmt, den wir im vorherigen Schritt gefunden haben. Die Anweisung könnte folgendermaßen aussehen:

```
ORG     0DA00H      ; beginning of USER
```

Andererseits könnte der Operand der ORG-Anweisung ein Ausdruck sein, wie zum Beispiel

```
ORG MSIZE*400H-600H
```

In diesem Fall wird die BIOS-Anfangsadresse vom Assembler entsprechend der Speichergröße ausgerechnet. Finden Sie die EQU-Anweisung, die MSIZE definiert, und prüfen Sie nach der Übersetzung, ob der Wert korrekt ist. Sie können aber auch anhand der Assemblerliste prüfen, ob der Wert korrekt ist.

Übersetzen Sie das Quellprogramm mit einem der Kommandos

```
MAC BIOS
```

oder

```
MAC USER
```

Dieser Schritt liefert drei Dateien mit den Erweiterungen HEX, SYM und PRN. Die HEX-Datei enthält die übersetzten Befehle in hexadezi-

maler Form verschlüsselt. Die SYM-Datei zeigt die Programmsymbole und ihre Werte. Die PRN-Datei enthält die Liste des Original-Quellprogramms mit dem übersetzten Code und seinen Adressen.

Untersuchen Sie die Assemblerliste mit dem Kommando TYPE. Suchen Sie die Sprungvektoren nahe dem Anfang der Liste. Vergleichen Sie die Adressen der Sprungvektoren mit den Werten, die wir mit dem Debugger für das laufende BIOS ermittelt haben. Falls die Adressen unterschiedlich sind, müssen Sie den Operanden der ORG-Anweisung (oder den Wert vom MSIZE in der EQU-Anweisung) ändern, so daß die Werte nach der Übersetzung mit denen des laufenden BIOS übereinstimmen.

Vergleichen Sie auch die Ziele der Sprungvektoren mit denen der laufenden BIOS-Version. Sind nämlich die Adressen der Sprungvektoren korrekt, die Zieladressen jedoch unterschiedlich, entspricht Ihr Quellprogramm nicht dem des laufenden BIOS. Es mag aber trotzdem möglich sein, diese Version zu benutzen.

Wenn die Werte der Sprungvektoren aus der Assemblerliste mit denen der laufenden Version übereinstimmen, können wir die übersetzte Version austesten, indem wir das laufende BIOS mit ihr überschreiben. Geben Sie das Kommando

```
SID BIOS.HEX
```

oder

```
DDT USER.HEX
```

ein. Damit starten wir den Debugger und veranlassen, daß die HEX-Datei von BIOS oder USER an die richtige Stellen kopiert wird. CP/M benutzt nun die „neue" Version des BIOS. Vielleicht möchten Sie die neue Version mit dem L-Kommando des Debuggers näher untersuchen und entdecken, daß dies Kommando nicht mehr funktioniert. Der Debugger hat sich mit dem BIOS teilweise überschrieben. Wann immer dies passiert, deaktiviert der Debugger das L-Kommando automatisch. Abhilfe ist einfach. Gehen Sie zurück zu CP/M mit dem Control-C-Kommando, und starten Sie den Debugger erneut.

Das Assemblieren des BIOS oder USER mit Microsoft MACRO-80

Mit dem Microsoft Assembler ist es ein wenig schwieriger, das übersetzte BIOS zu installieren. Ein Weg ist, die ORG-Anweisung durch eine PHASE-Anweisung zu ersetzen, z.B.

```
.PHASE 0DA00H      ; absolute code
```

Hier ist der Operand der Anfang von BIOS oder USER. Beachten Sie, daß das Symbol PHASE mit einem Punkt beginnt.

Achten Sie darauf, daß der Typ des Quellprogramms MAC ist, und nicht ASM. Übersetzen Sie das Programm mit dem Kommando

```
M80 =BIOS/L
```

In diesem Beispiel zeigt L dem Assembler an, daß zusätzlich zur REL-Datei eine PRN-Datei angelegt werden soll.

Sehen Sie sich die PRN-Datei an, wie im vorigen Abschnitt beschrieben. Vergleichen Sie die Adressen der Sprungvektoren in der Liste mit denen der laufenden Version des BIOS. Wenn sie übereinstimmen, können Sie die übersetzte Version mit dem Link-Loader LINK-80 und dem Debugger installieren. Beginnen Sie mit dem Kommando

```
L80 BIOS/E
```

Dies Kommando konvertiert die Datei BIOS.REL in eine ausführbare Version und hinterläßt sie am Anfang des Benutzerbereiches TPA. Die Angabe E veranlaßt den Loader, nach Erstellung des Speicherabbildes zu CP/M zurückzugehen.

Die Situation ist jetzt ungewöhnlich. Das gerade übersetzte BIOS liegt im Speicher ab Adresse 100 hex. Der erste Befehl jedoch ist ein Sprung zum Start des BIOS (hier Adresse DA00 hex). Unser Programm schließlich beginnt bei Adresse 103H.

LINK-80 hat drei Zahlen in eckigen Klammern angezeigt. Z.B.

```
[DA00 39F 3]
```

Die erste Zahl (DA00 hex) ist die Anfangsadresse des BIOS. Die zweite Zahl ist die Endadresse des BIOS in der TPA. Die dritte Zahl, 3, ist die Programmgröße, d.h. die Anzahl der 256-Byte-Blöcke, die nötig ist, um das Programm abzuspeichern. Retten Sie eine Kopie des Speichers mit dem CP/M-Kommando

```
SAVE XX BIOS.COM
```

wobei XX die Anzahl der zu speichernden Blöcke ist.

Laden Sie die neue Datei in den Speicher zurück mit dem Debugger-Kommando

```
SID BIOS.COM
```

Denken Sie daran, daß die ersten drei Bytes ab Adresse 100 hex einen unerwünschten Sprungbefehl enthalten. Das tatsächliche Programm beginnt erst bei 103 hex. Das Ende des geladenen Programms war bei 39F hex.

Verschieben Sie den Speicherinhalt mit dem Debugger an die richtige Stelle, und achten Sie darauf, daß Sie bei Adresse 103 hex beginnen, nicht bei Adresse 100 hex. Das Kommando könnte z.B. so aussehen:

```
M103,39F,DA00
```

Die neue Version des BIOS ist nun bereit im Speicher und hat das Original-BIOS überlagert. Allerdings haben wir bisher noch nichts im BIOS oder USER geändert.

DAS KOPIEREN DES GEÄNDERTEN BIOS AUF EINE DISKETTE

In dem vorigen Beispiel haben wir die Original-Version von BIOS oder USER übersetzt und mit der übersetzten Version die laufende überschrieben. Im nächsten Kapitel werden wir neue Funktionen in das BIOS-Quellprogramm einfügen, bevor wir es übersetzen. Wir können dann die neuen Funktionen testen, nachdem die übersetzte Version die laufende ersetzt hat. Wenn Sie jedoch jetzt Ihren Computer abschalten, wird wieder die Original-Version nach dem Booten des CP/M-Systems geladen. Es ist also notwendig, die geänderte Version des BIOS vom Speicher auf die Systemspuren einer Diskette zu kopieren. In diesem Beispiel haben wir zwar noch keine Änderungen am BIOS vorgenommen, lassen Sie uns jedoch diesen Schritt durchführen, damit Sie den Vorgang sicher verstehen.

Es gibt drei Möglichkeiten, eine geänderte Version des BIOS auf die Systemspuren zu schreiben. Beginnen wir mit der einfachsten.

Schreiben des BIOS auf die Diskette mit SAVEUSER

Die einfachste Methode, die laufende Version des BIOS auf die CP/M-Systemspuren zu schreiben, bietet das Programm SAVEUSER. Dieses Programm ist jedoch kein reguläres CP/M-Programm, und daher besitzen Sie es möglicherweise nicht. SAVEUSER kopiert direkt die laufende Version des BIOS auf die Systemspuren der Diskette in Laufwerk A. Um die laufende Version des BIOS zu speichern, geben Sie nur SAVEUSER ein und beantworten alle Fragen. Wenn Sie das Programm SAVEUSER nicht finden, müssen Sie einen anderen Weg wählen, um eine geänderte Version zu speichern.

Kopieren des BIOS aus einer HEX-Datei auf die Diskette mit SYSGEN

Weiter vorn in diesem Kapitel sahen wir, daß mit SYSGEN die Systemspuren von einer Diskette auf eine andere kopiert werden können. Das geschieht in zwei Schritten. Die Systemspuren werden von der Diskette in den Speicher gelesen, dann wird der Speicherinhalt auf eine neue Diskette geschrieben.

Mit SYSGEN kann man auch die Systemspuren einer Diskette ändern. In diesem Fall wird jedoch der Vorgang in der Mitte unterbrochen, nachdem das System in einen SYSGEN-Bereich gelesen wurde. Dieser Bereich wird mit dem geänderten BIOS überschrieben. Danach wird das geänderte System mit SYSGEN auf die neue Diskette geschrieben. Betrachten wir den ersten Teil dieses SYSGEN-Vorgangs.

Beim Kopieren der Systemspuren von einer Diskette auf eine andere sahen wir die folgenden Meldungen, nachdem SYSGEN die Systemspuren von Laufwerk A in den Speicher gelesen hatte:

```
CP/M image in RAM at 900H is ready to write
or reboot and "SAVE 40 CPMxx.COM"
Destination drive NAME (or RETURN to reboot)
```

Damals gaben wir den Namen des Ausgabelaufwerks an. Diesmal aber beenden wir SYSGEN mit RETURN. Dann retten wir den SYSGEN-Speicher (inklusive SYSGEN selbst) in eine reguläre CP/M-Datei. In diesem Beispiel sagt uns SYSGEN, daß 40 Blöcke zu 256 Bytes benötigt werden, um den Speicher zu retten, jedoch mag diese Zahl von System zu System variieren.

Nach dem Laden des Systems in den Speicher gehen wir mit RETURN in die Kommandoebene zurück. Nach Erscheinen des Promptzeichens A> geben Sie das Kommando

```
SAVE 40 CPM2.COM
```

ein, um den Speicher in eine Datei namens CPM2.COM zu schreiben. Diese Datei enthält das komplette CP/M-System mit einem Urlader. Es enthält ebenfalls am Anfang eine Kopie von SYSGEN.

(Beachten Sie, daß man einen Dateityp wählt, um die Art der Datei damit auszudrücken. CP/M benutzt den Dateityp COM für ausführbare Programme. Das System, das wir gerade gespeichert haben, ist jedoch kein ausführbares Programm, da es mit einer Kopie von SYSGEN beginnt und die übrigen Teile nicht an der richtigen Stelle stehen. Daher ist eigentlich der Typ SYS geeigneter. Dies ist auch möglich, wenn Sie SID benutzen, aber DDT verlangt den Typ COM.)

Wir haben nun eine reguläre CP/M-Datei, die eine Kopie des Original-CP/M enthält. Nun laden wir mit dem Debugger DDT dieses Systemabbild in den Speicher zurück, um es ändern zu können, und zwar ab der Startadresse 100H, dem normalen Arbeitsspeicherbereich. Natürlich liegt das System nicht hier, wenn man es benutzt.

Nach dem Starten des Debuggers kopiert dieser unser System in den Speicher. Es gibt nun also zwei Kopien von CP/M im Speicher (siehe Abb. 2.1). Die normale Arbeitsversion liegt am Ende des Speichers. Das mit SYSGEN erstellte Duplikat liegt in der TPA direkt hinter SYSGEN. Wir wollen die beiden Kopien die Arbeitsversion und die SYSGEN-Version nennen.

Im nächsten Schritt überschreiben wir das Original-BIOS mit dem BIOS hinter dem SYSGEN. Zuvor müssen wir jedoch den Anfang des BIOS- bzw. USER-Bereichs in der SYSGEN-Version finden. Der Debugger wird uns dabei helfen.

Wir starten den Debugger mit dem Kommando

```
DDT CPM2.COM
```

Hierdurch wird unser System in den Speicher geladen. Achten Sie darauf, daß CPM2.COM auf dem Standard-Laufwerk ist, wenn Sie DDT benutzen. Das Kommando

```
A:DDT CPM2.COM
```

arbeitet richtig, das Kommando

```
DDT B:CPM2.COM
```

jedoch nicht. Dieses Problem tritt bei SID nicht auf.

Nach dem Laden unseres Systems in den Speicher ab Adresse 100H gibt der Debugger drei Zahlen aus. Z.B.

```
NEXT PC END
2900 0100 ACFF
```

Notieren Sie die unter NEXT ausgegebene Zahl (2900 in unserem Beispiel). Dies ist die Endadresse des geladenen Systems.

Die SYSGEN-Version des ab 100H geladenen CP/M-Systems sollte dieselbe sein wie die des laufenden Systems. Den Anfang des BIOS bzw. USER in der Arbeitsversion haben wir bereits im vorigen Abschnitt

bestimmt. Wir müssen nun den Anfang des entsprechenden Bereiches in der SYSGEN-Version finden. Dazu benutzen wir das Kommando L.

Nach dem Laden unseres Systems mit dem Debugger wurde die NEXT-Adresse ausgegeben. Da dies die Endadresse des BIOS angibt, beginnen Sie mit einer um 100 hex kleineren Adresse. Finden Sie dort nicht die Sprungvektoren, vermindern Sie die Adresse abermals um 100 hex. Wenn z.B. die NEXT-Adresse 2900 ist, geben Sie das Kommando

```
L2800
```

ein. Die Ausgabe könnte dann sein wie folgt:

```
2800  JNZ   DB35
2803  PUSH  H
2804  PUSH  B
2805  LXI   H,DB18
2808  MVI   B,16
280A  MOV   C,M
280B  CALL  DAC4
280E  INX   H
280F  DCR   B
2810  JNZ   DB0A
2813  POP   B
```

Wir suchen einen Satz von Sprungvektoren in BIOS oder USER. Dies ist er offensichtlich nicht. Die Sprungadressen werden mit denen übereinstimmen, die wir zuvor in der Arbeitsversion des BIOS gefunden haben. Wiederholen Sie den Vorgang mit einer um 100 hex kleineren Adresse. Versuchen wir z.B.

```
L2700
```

Es könnte folgendes erscheinen:

```
2700  JMP   DA1B
2703  JMP   DA41
2706  JMP   DA7E
2709  JMP   DA96
270C  JMP   DAB6
270F  JMP   DACC
2712  JMP   DB8B
2715  JMP   DBD6
2718  JMP   DAFC
```

Nach diesem Satz von Vektoren haben wir gesucht. Die Adressen der Sprungvektoren stimmen mit den zuvor gefundenen überein.

Im nächsten Schritt bestimmen wir den „Offset" (die Differenz) zwischen SYSGEN-Version und Arbeitsversion des BIOS bzw. USER. Wir benutzen dafür das Kommando H (für Hexadezimal-Arithmetik). Dies ist ein nicht beschriebenes Kommando von DDT und SID. Wir schreiben

```
H2700,DA00
```

Dieses Kommando subtrahiert DA00 hex von 2700 hex. Der Debugger antwortet sowohl mit der Summe als auch mit der Differenz:

```
0100 4D00
```

Die Differenz haben wir gewollt, den zweiten Wert 4D00 hex. Diesen Wert müssen wir auf die Adresse des normal übersetzten Codes (DA00 hex) addieren, um die neuen BIOS- oder USER-Funktionen in den richtigen SYSGEN-Bereich zu bringen (2700 hex).

Nach Änderung des Programms BIOS.ASM oder USER.ASM übersetzen wir es und erhalten eine entsprechende Datei BIOS.HEX oder USER.HEX. Hiermit wollen wir das Original ersetzen. Der Debugger lädt HEX-Dateien automatisch an die richtige Adresse (über die Arbeitsversion) mit den beiden Kommandos

```
IUSER.HEX
R
```

(Das Kommando I initialisiert einen FCB für die Datei USER.HEX. Das Kommando R liest die entsprechende Diskettendatei in den Speicher.)

In diesem Fall wollen wir jedoch die Datei in den SYSGEN-Bereich und nicht in den Arbeitsbereich laden. Daher geben wir das Kommando R mit dem berechneten Offset ein:

```
IUSER.HEX
R4D00
```

Der Debugger lädt nun die HEX-Datei in den gewünschten SYSGEN-Bereich, und nicht in den Arbeitsbereich.

Nun haben wir eine Kopie des Original-CP/M im SYSGEN-Bereich, allerdings wurde das Original-BIOS durch ein geändertes BIOS ersetzt.

Gehen Sie mit Control-C zurück zu CP/M. Die geänderte Version können Sie nun mit dem Kommando

```
SAVE 40 CPMREV.COM
```

speichern. (Achten Sie darauf, daß Sie einen anderen Namen verwenden als zuvor, um die Original-Version von der geänderten unterscheiden zu können.)

Sie können aber auch SYSGEN starten, indem Sie seinen Namen eingeben. Die SYSGEN-Version von CP/M ist ja bereits im Speicher. Daher geben Sie auf die erste Frage von SYSGEN

```
Source drive NAME (or RETURN to skip)
```

RETURN ein. Damit überspringen Sie den ersten Teil von SYSGEN, der die Systemspuren in den Speicher einliest. Legen Sie z.B. eine Diskette in das Laufwerk B, und beantworten Sie die nächste Frage mit B:

```
Destination drive NAME (or RETURN to reboot)
```

SYSGEN bestätigt Ihre Eingabe

```
Place DESTINATION disk on B, then type RETURN
```

Wenn Sie wiederum RETURN eingeben, wird das neue System auf die Systemspuren der Diskette in Laufwerk B geschrieben. Man kann natürlich auch jedes andere Laufwerk benutzen. Stellen Sie sicher, daß die Diskette formatiert ist.

Ob Ihre Änderungen erfolgreich waren, testen Sie, indem Sie Ihren Computer aus- und wieder einschalten und danach das System von der neuen Diskette laden.

Kopieren des geänderten BIOS der Arbeitsversion auf eine Diskette mit SYSGEN

Die dritte Methode zum Schreiben der Systemspuren einer Diskette ähnelt der zweiten. In diesem Fall benutzen wir nicht direkt eine BIOS- oder USER-Datei. Statt dessen kopieren wir das BIOS oder USER der Arbeitsversion in den SYSGEN-Bereich. In unserem Beispiel laden wir Debugger und SYSGEN-Version wie oben mit dem Kommando

```
DDT CPM2.COM
```

Die Adresse des BIOS bzw. USER im SYSGEN-Bereich bestimmen wir wie im vorigen Abschnitt. Dann geben Sie das Kommando M (move) ein:

```
MDA00,DDFF,2700
```

Mit diesem Befehl speichern wir einen Block um. Dadurch erhalten wir eine Kopie des regulären BIOS bzw. USER (DA00 - DDFF) im SYSGEN-Bereich (2700 - 2AFF). Mit Control-C kehren wir zu CP/M zurück und starten SYSGEN. Gehen Sie vor, wie im vorigen Abschnitt beschrieben, und schreiben Sie das Speicherabbild des Systems auf die Systemspuren einer Diskette.

Nun haben Sie also eine funktionsfähige Version Ihres BIOS bzw. USER, und wir können mit dem Einfügen neuer Funktionen beginnen. Seien Sie sicher, daß Sie eine Kopie des Originals haben. Wenn nämlich Ihre neue Version fehlerhaft ist, können Sie, ausgehend vom Original, von neuem beginnen.

ZUSAMMENFASSUNG

In diesem Kapitel haben wir gelernt, eine Diskette zu duplizieren. Wir haben eine Diskette formatiert, Datenbereiche auf sie kopiert und die Systemspuren neu geschrieben. Außerdem haben wir gelernt, wie man den BIOS- oder USER-Bereich des CP/M ändert, und wie man die Änderungen permanent macht, indem man die neue Version auf die Systemspuren schreibt. Nun werden wir in der Lage sein, einige neue Funktionen einzufügen, die im nächsten Kapitel beschrieben werden.

Kapitel 3

Erweiterung des BIOS

EINFÜHRUNG

In Kapitel 1 haben wir gelernt, daß das CP/M-Basis-Ein-/Ausgabe-System (BIOS) die Programme zur Bedienung der peripheren Geräte, wie Konsole, Drucker und Disketten, enthält. In Kapitel 2 haben wir gelernt, wie man den BIOS- oder USER-Bereich findet, um darin Änderungen vorzunehmen. In diesem Kapitel werden wir das BIOS im einzelnen studieren, und wir werden mehrere nützliche Erweiterungen einbauen, wie z.B. das Senden der Konsolausgabe zum Drucker und das Prüfen, ob der Drucker bereit ist.

Da nur ein kleiner Speicherbereich für die BIOS-Routinen zur Verfügung steht, muß man diese Erweiterungen in Assemblersprache schreiben und nicht in einer höheren Programmiersprache wie BASIC oder Pascal. Wir wollen uns daher die Arbeitsweise eines Assemblers noch einmal anschauen.

DAS PROGRAMMIEREN IN ASSEMBLER

Die Assemblersprache ist eine niedere Programmiersprache, in der die Befehle für eine CPU direkt durch einen mnemotechnischen Code (Opcode) angegeben werden. Die CPU 8080 hat drei allgemeine 16-Bit-Register mit den Namen HL, DE und BC. Der vollständige Befehlssatz sowohl für den 8080 wie für den Z80 sind in den Anhängen wiedergegeben.

Betrachten wir z.B. den folgenden Befehl:

```
JMP D303
```

Dieser Befehl weist die CPU an, zur Adresse D303 hex zu verzweigen. Der Assembler generiert den entsprechenden Binärcode. Es besteht eine Eins-zu-Eins-Korrespondenz zwischen dem Assemblerbefehl und dem Code, den er generiert.

Im Gegensatz hierzu werden bei einer höheren Programmiersprache, wie BASIC oder Pascal, aus einer Anweisung meist mehr als eine CPU-

Instruktion generiert. Obwohl jeder Compiler unterschiedlich arbeitet, könnte die Zeile

```
I = I + 5
```

in die folgenden CPU-Befehle übersetzt werden:

```
LXI D,5
LHLD I%
DAD D
SHLD I%
```

Diese Befehlssequenz lädt das Register DE mit dem Wert 5 und das HL-Register mit Wert bei Adresse I%. Die Werte in DE und HL werden addiert und die Summe in HL gespeichert. Das Ergebnis wird dann in den Speicher bei der Adresse I% geschrieben.

Assemblerprogramme werden geschrieben und gepflegt mit einem der vielen Editoren, wie ED, WordStar, WordMaster, Magic Wand, PMATE oder Benchmark. Das Quellprogramm wird dann mit einem Assembler übersetzt und so in ausführbaren Code umgesetzt. Mit CP/M wird ein Assembler geliefert (ASM). Er ist jedoch für viele der Programme in diesem Buch nicht geeignet, da er keinen Makroprozessor enthält. (Wir werden auf Makros und Makroprozessoren im Kapitel 4 eingehen.) Es gibt zwei verbreitete Assembler, die einen Makroprozessor enthalten, nämlich MACRO-80 von Microsoft und MAC von Digital Research. Jeder dieser beiden Assembler verarbeitet Standard-INTEL-8080-Befehle. Der Microsoft-Assembler akzeptiert auch Zilog-Z80-Befehle, der Digital Research-Assembler kann Z80-Befehle nur mit Makros generieren.

Ein einfaches Assemblerprogramm

Um sicherzustellen, daß Sie die Arbeitsweise Ihres Assemblers und der zugehörigen Programme beherrschen, werden wir ein einfaches Programm in 8080-Assemblersprache übersetzen und ausführen. Benutzen Sie einen CP/M-Editor, um das Quellprogramm in Abb. 3.1 zu erstellen, und geben Sie ihm den Namen BELL. Der Dateityp sollte ASM für den Digital Research-, MAC für den Microsoft-Assembler sein. Wenn Sie den Microsoft-Assembler benutzen, entfallen die ORG-Anweisung und die Apostrophe der TITLE-Anweisung. Dies ist eines der wenigen Programme, die Sie mit dem Digital Research-Assembler ASM übersetzen können. Benutzen Sie den Dateitype ASM, und lassen Sie die TITLE-Anweisung weg.

Beachten Sie die vier verschiedenen Spalten in der Liste des Quellprogramms. Man benutzt gewöhnlich das ASCII-Zeichen TAB, um diese Spalten zu erhalten. Der Digital Research-Assembler verlangt dies zwar nicht, die Liste wird dadurch jedoch leichter lesbar. Für einen normalen Befehl haben die vier Spalten folgendes Aussehen:

```
LABEL: MNEMONIC OPERAND ; Kommentar
```

LABEL ist eine Folge von alphanumerischen Zeichen und endet mit einem Doppelpunkt. (Der Doppelpunkt ist wahlfrei für den Digital Research-Assembler, erforderlich für den Microsoft-Assembler.) Auf LABEL kann man sich von beliebiger Stelle im Programm beziehen. MNEMONIC ist der jeweilige CPU-Befehl; seine Schreibweise kann von Assembler zu Assembler verschieden sein. Der Operand ist der Parameter für den CPU-Befehl; es kann ein Register, eine Konstante oder eine Speicheradresse sein. Der Kommentar, der mit einem Semikolon beginnt, beschreibt den Befehl.

Nicht alle Zeilen des Quellprogramms enthalten CPU-Befehle. Einige von ihnen enthalten Assembleranweisungen oder Pseudobefehle. Diese Zeilen generieren keine CPU-Befehle, sondern werden benutzt, um Konstanten zu definieren, Speicheradressen zu setzen oder dem Assembler Anweisungen zu geben. Die erste Zeile

```
TITLE 'Konsol-Glocke'
```

weist den Assembler an, den angegebenen Operanden als Überschrift für jede Seite zu benutzen. Die Anweisung

```
ORG 100H
```

setzt die Adresse des übersetzten Codes auf 100 hex. Die nächsten drei Zeilen nennt man Equates. Sie definieren die Werte der Symbole BEL, BDOS UND TYPEF. Z.B. wird mit

```
BEL EQU 7
```

das Symbol BEL dem Wert 7 gleichgesetzt. Der Doppelpunkt am Ende eines Definitionssymbols entfällt, weil es keine Speicheradresse darstellt.

Fünf Zeilen des Programms nach Abb. 3.1 generieren Computerbefehle. Die erste setzt den Stackpointer (Stapelzeiger) auf 100 hex:

```
LXI SP,100H
```

```
            TITLE 'Konsol-Glocke'
            ;(Tagesdatum)
ORG         100H                ;Digital Research Version
;
BEL         EQU     7           ;ASCII-Bell.
BDOS        EQU     5           ;DOS-Eingang
TYPEF       EQU     2           ;Zeichen auf die Konsole ausgeben
;
START:
            LXI     SP,100H
            MVI     C,TYPEF
            MVI     E,BEL
            CALL    BDOS
            JMP     0
;
            END     START
```

Abb. 3.1: Das Programm BELL für die Konsol-Glocke

(Der Stackpointer ist ein CPU-Register, das sich auf einen besonderen Speicherbereich bezieht. In unserem Beispiel initialisieren wir den Pointer auf 100 hex. Dieser Wert wird jedoch durch Befehle wie PUSH, POP, CALL und RET geändert.) Der zweite Befehl lädt den Wert 2 (TYPEF) in das C-Register:

```
MVI C,TYPEF
```

Der dritte Befehl lädt das E-Register mit dem Wert 7 (BEL):

```
MVI E,BEL
```

Der vierte Befehl generiert einen Unterprogrammaufruf nach Adresse 5 (BDOS). Der fünfte Befehl verzweigt nach Adresse 0.

Die letzte Anweisung des Programms gibt an, daß das Programm bei der Marke START beginnen soll.

Die Programmübersetzung

Nach der Erstellung Ihres Quellprogrammms mit dem Editor vergleichen Sie es mit der Liste in Abb. 3.1. Korrigieren Sie es nötigenfalls, und über-

setzen Sie es. Wenn Sie einen der Assembler MAC oder ASM von Digital Research benutzen, geben Sie das Kommando

```
MAC BELL
```

bzw.

```
ASM BELL
```

ein. Für den Microsoft-Assembler geben Sie

```
M80 =BELL/L
```

ein.

In diesem Schritt wird eine Listendatei namens BELL.PRN erzeugt. Vergleichen Sie diese Liste mit Abb. 3.2 (Digital Research-Assembler) bzw. Abb. 3.3 (Microsoft-Assembler).

Die Assemblerliste zeigt das Original-Quellprogramm zusammen mit dem Maschinencode und den Adressen, an denen sie zur Ausführung gespeichert sind. Der Maschinencode und die Adressen werden hexadezimal ausgegeben. Befehle wie JMP und CALL, die sich auf den Speicher beziehen, sind drei Bytes lang. Das zweite und dritte Byte enthält die Speicheradresse. Die niederwertige Hälfte erscheint im zweiten Byte, die höherwertige im dritten. D.h. die beiden Adreßbytes erscheinen vertauscht. Der Digital Research-Assembler gibt die Adresse in dieser vertauschten Reihenfolge wieder. Z.B. sieht ein CALL zum BDOS (Adresse 5) so aus:

```
CD0500 CALL   BDOS    ; Version Digital Research
```

Bei einer Adresse scheint jedoch die andere Reihenfolge „natürlicher" zu sein. Daher zeigt der Microsoft-Assembler das höherwertige Byte zuerst, und der CALL zum BDOS sieht dann so aus:

```
CD 0005 CALL    BDOS    ; Version Microsoft
```

Man muß allerdings daran denken, daß die Reihenfolge der Bytes im Speicher der Digital Research-Version entspricht und nicht der Microsoft-Version.

Der nächste Schritt ist die Ausführung des Programms. Das können wir aber noch nicht machen, da der Assembler keinen ausführbaren Code erzeugt hat. Der Digital Research-Assembler hat eine hexadezimale

```
                TITLE 'Konsol-Glocke'
                ;(Tagesdatum)
0100            ORG     100H            ;Digital Research-Version
                ;
0007 =          BEL     EQU     7       ;ASCII-Bell.
0005 =          BDOS    EQU     5       ;DOS-Eingang
0002 =          TYPEF   EQU     2       ;Zeichen auf die Konsole
                                        ausgeben
                ;
                START:
0100 310001             LXI     SP,100H
0103 0E02               MVI     C,TYPEF
0105 1E07               MVI     E,BEL
0107 CD0500             CALL    BDOS
010A C30000             JMP     0
                ;
010D                    END     START
```

Abb. 3.2: Das Listing des Digital Research-Assemblers zu Abb. 3.1

ASCII-Datei namens BELL.HEX erzeugt. Diese HEX-Datei wird mit dem Kommando

```
LOAD BELL
```

in die ausführbare Datei BELL.COM verwandelt. (LOAD ist ein zum CP/M-Betriebssystem gehörendes Programm.) Geben Sie nun

```
BELL
```

ein, um das Programm zu starten. Die Konsolglocke wird ertönen, und die Kontrolle wird an CP/M abgegeben.

Der Microsoft-Assembler erzeugt eine REL-Datei, die anders verarbeitet wird. Man kann zwar auch eine HEX-Datei aus einer Microsoft-REL-Datei machen, es ist aber einfacher, aus einer REL-Datei mit dem Linker L80 eine ausführbare Datei zu erstellen. Das Programm BELL.REL kan ausgeführt werden mit dem Kommando

```
L80 BELL/G
```

```
                TITLE Konsol-Glocke
                ;(Tagesdatum)
                    ;
0007            BEL     EQU     7       ;ASCII-Bell.
0005            BDOS    EQU     5       ;DOS-Eingang
0002            TYPEF   EQU     2       ;Zeichen auf die
                                        Konsole ausgeben
                    ;
0000'           START:
0000' 31 0100           LXI     SP,100H
0003' 0E 02             MVI     C,TYPEF
0005' 1E 07             MVI     E,BEL
0007' CD 0005           CALL    BDOS
000A' C3 0000           JMP     0
                    ;
                        END     START
```

Abb. 3.3: Das Listing des Microsoft-Assemblers zu Abb. 3.1

Dies Kommando generiert eine Binärdatei, die bei Adresse 100 hex beginnt, und führt sie aus. Nach der Ausführung wird das CP/M-Promptzeichen ausgegeben. Dann geben Sie

```
SAVE 1 BELL.COM
```

ein. (Wir sahen in Kapitel 2, daß der L80 uns sagt, wie viele Blöcke gespeichert werden müssen.) Dieses Kommando rettet das ausführbare Speicherabbild. Das Programm kann erneut mit Eingabe von BELL gestartet werden. In Kapitel 1 haben wir ein Programm erstellt, mit dem wir BELL noch einmal ausführen können, da das Speicherabbild noch unverändert ist.

Man kann mit dem L80 auch eine COM-Datei erstellen. Das Kommando

```
L80 BELL/N,BELL/E
```

erzeugt die COM-Datei BELL.COM und kehrt danach zu CP/M zurück. Ausgeführt wird das Programm dann durch Eingabe von BELL.

Bereiten wir nun die Änderung des BIOS vor.

DIE BIOS-EINSPRUNGTABELLE

In Kapitel 2 sahen wir, daß es eine Reihe von Vektoren am Beginn des BIOS gibt mit den Adressen der entsprechenden Routinen innerhalb des BIOS. Auch sahen wir, daß einige Versionen von CP/M eine Erweiterung des BIOS beinhalten, die man USER nennt. In diesem Fall umfaßt das BIOS die Diskettenroutinen, und der USER-Bereich die übrigen Routinen. Es ist also ein Satz von Vektoren am Beginn des BIOS und ein weiterer am Beginn des USER-Bereiches. Die Vektoren für die Diskettenoperationen verweisen auf das BIOS. Die übrigen Vektoren, die sich auf Konsol- und Druckeroperationen beziehen, zeigen im allgemeinen auf einen gleichartigen Vektorsatz am Anfang des USER-Bereiches.

Die Vektoren am Anfang des BIOS zeigt Abb. 3.4. Der erste Vektor heißt Kaltstarteingang. Er wird zum Starten von CP/M benutzt. Der zweite wird zur Beendigung von größeren Programmen benutzt; man nennt ihn Warmstartvektor. Die Vektoren für die vier logischen Geräte, Konsole, Leser, Stanzer und Drucker, erscheinen als nächstes. Diese vier Geräte werden mit den folgenden Namen bezeichnet:

```
CON:  Konsole  (Ein- und Ausgabe)
RDR:  Leser    (Eingabe)
PUN:  Stanzer  (Ausgabe)
LST:  Drucker  (Ausgabe)
```

Beachten Sie, daß die Namen für die logischen Geräte mit einem Doppelpunkt enden. Dies gilt auch für die Namen der Diskettenlaufwerke, wie A:, B: etc. Man kann dadurch einen Gerätenamen von einem Dateinamen unterscheiden. So ist z.B. PUN eine Datei, PUN: dagegen die logische Stanzeinheit.

```
BIOS      JMP   COLD     ;Kaltstart
BIOS+3    JMP   WARM     ;Warmstart
BIOS+6    JMP   CSTAT    ;Konsol-Status
BIOS+9    JMP   CONIN    ;Konsol-Eingabe
BIOS+12   JMP   CONOUT   ;Konsol-Ausgabe
BIOS+15   JMP   LIST     ;Drucker-Ausgabe
BIOS+18   JMP   PUNCH    ;Stanzer-Ausgabe
BIOS+21   JMP   READER   ;Extra Eingabe-Gerät
```

Abb. 3.4: Die ersten acht CP/M-BIOS-Vektoren

Untersuchung des BIOS mit dem Debugger

In Kapitel 2 haben wir die BIOS-Sprungvektoren mit dem Debugger gefunden. Falls Sie dies noch nicht getan haben, sollten Sie es jetzt nachholen.

Sie erinnern sich, daß wir den Debugger gestartet haben, und das Kommando

```
L0 (der Buchstabe L und eine Null)
```

eingegeben haben. Die Ausgabe ist dann etwa:

```
0000 JMP D303
0003 NOP
0004 NOP
0005 JMP AD00
. . . .
```

Der Sprung bei Adresse 0 verweist auf den Warmstarteingang im BIOS. D.h. das BIOS in diesem System beginnt bei Adresse D300 hex. Der Sprung bei Adresse 5 ist normalerweise der BDOS-Eingang; wir benutzten ihn, um die Konsolglocke in dem Programm BELL ertönen zu lassen. Hier jedoch ist die Adresse bei Adresse 6 vom Debugger geändert worden. D.h. die normale BDOS-Adresse in diesem System ist C506 hex, aber in diesem Beispiel hat der Debugger sie in AD00 hex geändert.

Wenn DDT (oder SID) gestartet wird, liest CP/M das Programm in den Speicher zum Beginn der TPA und springt dorthin. Der Debugger verschiebt sich dann selbst in den oberen Speicherbereich. Dadurch kann ein anderes Programm (das getestet wird) in die TPA geladen werden und unter Kontrolle des Debuggers laufen. Der Debugger jedoch muß die BDOS-Aufrufe des Programms abfangen. Daher ändert er die Adresse bei Adresse 6.

Nachdem wir die Lage des BIOS bestimmt haben, können wir die Sprungvektoren am Beginn des BIOS disassemblieren mit dem Debuggerkommando LD300. Die Ausgabe ist dann etwa:

```
D300  JMP  D380   (Kaltstart)
D303  JMP  D39F   (Warmstart)
D306  JMP  DA06   (Konsol-Status)
D309  JMP  DA09   (Konsol-Eingabe)
D30C  JMP  D4E6   (Konsol-Ausgabe)
D30F  JMP  DA0F   (Drucker-Ausgabe)
D312  JMP  DA12   (Stanzer-Ausgabe)
```

```
D315 JMP  DA15  (Reserve-Eingabegerät)
D318 JMP  D4CA  (Beginn der Disketten-Routinen)
D31B JMP  D499
D31E JMP  D4CC
```

In Kapitel 2 sahen wir, daß ein zweiter Satz von Sprungvektoren vorhanden sein könnte, in einem Extrabereich namens USER. Im obigen Beispiel beginnt der USER-Bereich bei Adresse DA00 hex. Wenn wir diesen Bereich mit dem Debuggerkommando LDA00 untersuchen, ist die Ausgabe etwa:

```
DA00 JMP  DA1B  (Kaltstart)
DA03 JMP  DA41  (Warmstart)
DA06 JMP  DA7E  (Konsol-Status)
DA09 JMP  DA96  (Konsol-Eingabe)
DA0C JMP  DAB6  (Konsol-Ausgabe)
DA0F JMP  DACC  (Drucker-Ausgabe)
DA12 JMP  DB8B  (Stanzer-Ausgabe)
DA15 JMP  DBD6  (Reserve-Eingabegerät)
```

In den folgenden Abschnitten interessieren wir uns für die Vektoren bei den Adressen DA0C und DA0F, die der Konsol- und Drucker-Ausgabe dienen.

BENUTZUNG DES DRUCKERS MIT DEM DEBUGGER

Manchmal ist es wünschenswert, eine gedruckte Liste von den Konsol-Ausgaben zu bekommen. Dies erreicht man durch Eingabe von Control-P in der Konsol-Eingabephase. Ein laufendes Programm jedoch kann den Drucker auf diese Weise nicht einschalten. Trotzdem ist es manchmal wünschenswert, daß ein Programm den Drucker anschaltet. Im nächsten Absatz werden wir ein kurzes Programm schreiben, das dies tut; zunächst werden wir dies direkt durchführen, und zwar mit dem Debugger.

Beachten Sie, daß der Vektor für die Drucker-Routine (bei Adresse DA0F in der obigen Liste) unmittelbar hinter dem Vektor für die Konsol-Ausgabe steht (bei Adresse DA0C). Wenn wir den Sprungbefehl zur Konsol-Ausgabe in einen CALL-Befehl ändern, aktivieren wir Konsol- und Drucker-Ausgabe gleichzeitig. Wir werden dies mit dem Debugger ändern. Dieses Vorgehen nennt man auch einen „patch" (Flicken). Sie müssen bei diesem Schritt sehr vorsichtig sein, da Sie direkt das BIOS ändern. Zwar ändern Sie nur ein Byte, es darf aber nicht ein falscher Wert oder eine falsche Adresse sein. Andernfalls wird CP/M nicht mehr auf Kommandos reagieren oder merkwürdige Sachen machen.

Benutzen Sie das Debuggerkommando A (Assemblieren), um an der Adresse DA0C (in diesem Beispiel) zu ändern:

```
ADA0C           (Sie geben dies ein)
DA0C CALL DAB6  (Sie geben „CALL DAB6" ein)
DA0F            (Sie geben RETURN ein)
```

In diesem Beispiel benutzen wir den Debugger, um den Drucker durch Änderung eines JMP-Befehls in einen CALL-Befehl zu aktivieren. Jedes laufende Programm (ausgenommen Microsoft-BASIC) kann diese Technik anwenden.

Alternativ können wir das Debuggerkommando S (für Setzen) benutzen, um den JMP-Befehl (C3 hex) in einen CALL-Befehl (CD hex) zu ändern. Die Kommandos hierfür sind:

```
SDA0C           (Sie geben dies ein)
DA0C C3 CD      (Sie geben „CD" ein)
DA0D B6.        (Sie geben den Punkt ein)
```

Von nun an wird der Drucker dieselbe Information anzeigen wie der Bildschirm. Zurückändern können wir das BIOS, indem wir den CALL-Befehl wieder in einen JMP-Befehl ändern. (Falls Sie etwas falsch gemacht haben und CP/M nicht mehr läuft, machen Sie einfach einen Kaltstart. Vielleicht müssen Sie Ihren Computer auch aus- und wieder einschalten, damit er wieder richtig arbeitet.) Wir wollen dies nun automatisieren.

EIN PROGRAMM ZUM AN- UND ABSCHALTEN DES DRUCKERS

Das An- und Abschalten des Druckers vereinfachen wir, indem wir das Ändern zwei Programmen überlassen. Da die beiden Programme sehr kurz sind, erstellen wir sie schneller mit dem Debugger als mit dem Assembler. Laden Sie den Debugger, und geben Sie das Kommando A100 ein, um ein Programm zu assemblieren, das bei 100 hex beginnt. Dann geben Sie die folgenden Befehle ein:

```
LHLD 1
LXI D,9
DAD D
MVI M,CD
RET
```

Dann drücken Sie noch einmal RETURN, um diesen Schritt zu beenden.

Nach Eingabe dieses kurzen Programms disassemblieren Sie es mit dem Kommando L100. Die Ausgabe ist dann:

```
0100 LHLD  0001
0103 LXI   D,0009
0106 DAD   D
0107 MVI   M,CD
0109 RET
```

Kehren Sie zurück zu CP/M mit Control-C. Speichern Sie das Programm:

```
SAVE 1 LISTON.COM
```

Bevor Sie dieses Programm starten, erstellen Sie das Programm, mit dem Sie die BIOS-Änderung wieder rückgängig machen können. Laden Sie LISTON mit dem Debugger:

```
DDT LISTON.COM
```

Ändern Sie den CALL-Befehl in Adresse 108 hex, den zweiten Operanden des MVI-Befehls, in einen JMP-Befehl (C3 hex). (Benutzen Sie das Kommando S108, um den Wert C3 einzugeben, oder geben Sie mit dem Kommando A107 den Befehl „MVI M,C3“ ein.) Kehren Sie zu CP/M zurück, und speichern Sie das zweite Programm mit

```
SAVE 1 LISTOFF.COM
```

Bei der Ausführung von LISTON lädt der erste Befehl die BIOS-Warmstart-Adresse in das HL-Register. (Zur Erinnerung: Adresse 0 enthält den JMP-Befehl, die Adressen 1 und 2 enthalten die Warmstart-Adresse.) Der zweite Befehl lädt eine 9 in das DE-Register, die Differenz zwischen dem Warmstarteingang und dem Konsol-Ausgabeeingang. Der dritte Befehl addiert die Register HL und DE und speichert die Summe in HL. Das HL-Register zeigt nun auf den Konsol-Ausgabevektor. Der vierte Befehl ersetzt den JMP-Befehl durch einen CALL-Befehl (CD hex). Der letzte Befehl bewirkt die Rückkehr zu CP/M.

Da Hexadezimal der Standardmodus des Debuggers ist, können wir hexadezimale Daten ohne das Suffix H eingeben. Im Gegensatz dazu ist der Standardmodus des Assemblers dezimal.

Um die Programme zu testen, schalten Sie Ihren Drucker ein, und geben Sie das CP/M-Kommando

```
LISTON
```

und danach

```
DIR
```

Das Inhaltsverzeichnis sollte sowohl auf dem Bildschirm als auch auf dem Drucker erscheinen. Dann geben Sie die Kommandos

```
LISTOFF
DIR
```

ein. Das Verzeichnis wird nur noch auf dem Bildschirm erscheinen.

Diese beiden kurzen Programme sind nützlicher für das Verständnis der Arbeitsweise von CP/M als für den tatsächlichen Gebrauch. Auch werden sie nicht immer erwartungsgemäß arbeiten. Insbesondere werden sie bei Microsoft-BASIC versagen. Wir werden nun das BIOS so ändern, daß man mit dem IOBYTE den Drucker aktivieren kann. (Lesen Sie in Kapitel 2 nach, wie man die BIOS- oder USER-Routinen findet und ändert.)

ANSCHALTEN DES DRUCKERS MIT DEM IOBYTE

Wir haben gesehen, daß das BIOS Vektoren für die Bedienung der vier logischen Geräte Konsole, Leser, Stanzer und Drucker benutzt. In CP/M ist eine Methode vorgesehen, diese vier logischen Geräte 16 physikalischen Geräten zuzuordnen. D.h. jedes der vier logischen Geräte kann vier verschiedenen physikalischen Geräten zugewiesen werden. Die momentane Zuordnung der logischen Geräte ist in einem Byte bei Adresse 3 verschlüsselt. Die beiden niedrigsten Bits zeigen die Zuordnung der Konsole, die nächsten beiden die des Lesers, die folgenden beiden die des Stanzers und die übrigen beiden die des Druckers.

Obwohl man mit dem IOBYTE die vier logischen Geräte 16 verschiedenen physikalischen Geräten zuordnen kann, ist es nicht nötig, dies für alle gleichzeitig zu tun. Jeder Teil kann einzeln zugefügt werden, wodurch sich der Vorgang vereinfacht. Das IOBYTE ist sehr nützlich, auch wenn die Konsole und der Drucker die beiden einzigen vorhandenen Geräte sind.

Lassen Sie uns mit einer einfachen Anwendung des IOBYTE beginnen – dem An- und Abschalten des Druckers. Wir definieren das niederwertige Bit des IOBYTE bei Adresse 3 als Druckerschalter. Wenn dieses Bit 1 ist, soll der Drucker die Konsol-Ausgabe ausgeben. Anderenfalls soll der

Drucker von der Konsol-Ausgabe unberührt bleiben. Man kann natürlich den Drucker immer noch mit Control-P an- oder abschalten. Weiterhin soll der Bildschirm die Konsol-Ausgabe anzeigen, ob der Drucker angeschaltet ist oder nicht.

Als erstes müssen wir sicherstellen, daß das IOBYTE richtig initialisiert ist. Suchen Sie in der BIOS-Liste nach dem ersten Sprungvektor. Dies wird der erste ausführbare Befehl nahe dem Programmanfang sein. Suchen Sie nun die angegebene Adresse auf, und folgen Sie den Befehlen bis zu einem RET-Befehl. Irgendwo in diesem Bereich wird eine Befehlsfolge sein, wie etwa:

```
COLD:
        MVI A,0
        STA 3
```

oder

```
COLD:
        MVI A,INITAL
        STA IOBYTE
```

Im zweiten Beispiel ist der Wert von INITAL mit 0 definiert, und IOBYTE hat den Wert 3. Wenn Sie eine solche Folge nicht finden können, fügen Sie sie mit dem Editor ein. Vergessen Sie nicht, die Symbole IOBYTE und INITAL zu definieren, wenn Sie die zweite Form wählen. Etwa:

```
INITAL   EQU  0
IOBYTE   EQU  3
```

Im nächsten Schritt wird die Konsol-Ausgaberoutine geändert. Suchen Sie in der BIOS-Assemblerliste den Konsol-Ausgabevektor (xx0C hex) und den Drucker-Ausgabevektor (xx0F hex). Es sind der fünfte und sechste Vektor in der Liste. Notieren Sie die als Operanden benutzten Labels. Sie könnten etwa so aussehen:

```
XX0C     JMP   CONOUT
XX0F     JMP   LIST
```

Gehen Sie zur Adresse CONOUT, und fügen Sie dort den folgenden Code ein:

```
CONOUT:                 ;
        LDA   IOBYTE    ; Eingabewert
        ANI   1         ; Maskieren
        CNZ   LIST      ; Ausgabe auf Drucker
CONO2:                  ; Terminalausgabe
```

Der erste Befehl lädt den Akkumulator mit dem Inhalt von Speicheradresse 3, der Adresse von IOBYTE:

```
LDA   IOBYTE
```

Der zweite Befehl verknüpft den Akkumulator und den Wert 1 durch ein logisches „UND“:

```
ANI   1
```

Die Operation UND (ANI) setzt alle Bits des Akkumulators außer dem niedrigsten (Bit 0) auf null. Der Befehl setzt auch das Zero-Flag entsprechend. D.h. das Zero-Flag wird gesetzt, wenn das niedrigste Bit null ist. Anderenfalls wird es gelöscht.

Der dritte Befehl weist die CPU an, das Unterprogramm LISTT aufzurufen, wenn das Zero-Flag nicht gesetzt ist (d.h. das niedrigste Bit nicht Null ist):

```
CNZ   LISTT
```

Vergessen Sie nicht, das Label CONO2 einzufügen. Wir brauchen es für ein späteres Programm in diesem Kapitel. Beachten Sie auch, daß der Sprung zur Druckroutine LISTT heißt. Dies ist nötig, um zwischen dem logischen und dem physischen Drucker zu unterscheiden. Suchen Sie das Label LIST. Steht in dieser Zeile auch ein Befehl, splitten Sie die Zeile so auf, daß das Label in einer Extrazeile steht. Fügen Sie das Label LISTT: direkt hinter dem Label LIST ein.

Fügen Sie in einer Kommentarzeile nahe dem Programmanfang das Tagesdatum ein.

Änderung des IOBYTE mit dem Debugger

Assemblieren Sie die neue Version, und laden Sie sie mit dem Debugger in den Speicher. Überprüfen Sie, ob das IOBYTE bei Adresse 3 den Wert 0 hat. Geben Sie das Debuggerkommando S3. Die Ausgabe wird

```
0003 X
```

sein, wobei X der Wert des IOBYTE ist. Ist der Wert 0, dann geben Sie einen Punkt ein, um das Kommando zu beenden. Andernfalls geben Sie 0 ein. Schalten Sie den Drucker ein, und ändern Sie mit dem Debuggerkommando S den Inhalt von Adresse 3 in 1:

```
S3        (Dies geben Sie ein)
0003 0 1  (Sie geben 1 ein)
0004 0 .  (Sie geben den Punkt ein)
```

Die letzte der obigen Zeilen sollte sowohl auf dem Bildschirm als auch auf dem Drucker erscheinen, da das IOBYTE jetzt 1 ist. Testen Sie andere Kommandos, wie

```
D0
```

Wiederum sollte die Drucker- und Bildschirm-Ausgabe gleich sein. Ändern Sie das IOBYTE mit dem S-Kommando wieder in 0. Der Drukker sollte nun nicht mehr die Bildschirmausgabe kopieren. Wenn Sie diese neue Version des BIOS permanent machen wollen, müssen Sie sie auf die Systemspuren kopieren. Natürlich sollte man den Drucker immer noch mit Control-P aktivieren können.

Änderung des IOBYTE mit BASIC

Wir werden diese Methode nun mit Microsoft-BASIC ausprobieren. Laden Sie BASIC, und schreiben Sie ein kurzes Programm, z.B. das aus Kapitel 1:

```
10 FOR K = 1 TO 9
20 PRINT K, 1/K, K*K
30 NEXT K
40 END
```

Starten Sie das Programm. Die Ergebnisse sollten auf der Konsole erscheinen. Schalten Sie den Drucker ein, und geben Sie die folgenden Kommandos:

```
POKE 3,1
RUN
```

Das BASIC-Kommando POKE ändert den Wert des IOBYTE in 1. Nach dem Start des Programms wird die Ausgabe sowohl auf dem Drucker als auch auf der Konsole erscheinen.

Wir hatten bereits festgestellt, daß Microsoft-BASIC das Anschalten des Druckers mit Control-P nicht erlaubt. Dafür haben wir nun eine andere Methode gefunden. Abgeschaltet wird der Drucker mit dem BASIC-Kommando

```
POKE 3,0
```

Kehren Sie mit dem SYSTEM-Kommando zu CP/M zurück.

Änderung des IOBYTE mit STAT

Wir haben gelernt, wie man mit dem Debugger oder mit BASIC das IOBYTE bei Adresse 3 ändert. Man kann das IOBYTE auch mit STAT ändern.

Wir haben gesehen, daß den vier logischen Geräten CON:, RDR:, PUN: und LST: jeweils zwei Bits des IOBYTE zugeordnet sind. Jedem dieser vier logischen Geräte kann eins von vier physikalischen Geräten durch Änderung des IOBYTE zugeordnet werden. STAT stellt hierfür 16 Namen zur Verfügung. Die 16 Namen finden Sie in Tabelle 3.1. Mit STAT kann man sich diese Tabelle anzeigen lassen:

```
STAT VAL:
```

Das IOBYTE kann von 0 in 1 geändert werden mit dem Kommando

```
STAT CON:=CRT:
```

Mit dem Kommando

```
STAT CON:=TTY:
```

wird STAT den Wert wieder von 1 in 0 ändern.

Die Namen der vier logischen Geräte wurden vor vielen Jahren gewählt, als man gewöhnlich Fernschreiber (TTY von Teletype) benutzte. Man sollte sie jetzt in zweckmäßigere Namen ändern. Z.B. könnte man TTY: in CRT: ändern, CRT: in LST:. Man kann dies leicht mit dem Debugger tun.

Laden Sie STAT in den Speicher mit dem Kommando

```
DDT STAT.COM
```

	Bits 00	01	10	11
CON:	TTY:	CRT:	BAT:	UC1:
RDR:	TTY:	PTR:	UR1:	UR2:
PUN:	TTY:	PTP:	UP1:	UP2:
LST:	TTY:	CRT:	LPT:	UL1:

Tabelle 3.1: Die Namen für die vier logischen Geräte

Sehen Sie sich den Anfang von STAT mit dem Kommando

```
D100
```

an. Die ASCII-Zeichen der Daten auf der rechten Seite des Bildschirms zeigen die Namen der vier logischen Geräte CON:, RDR:, PUN: und LST:, beginnend bei Adresse 139 hex. Die 16 Namen der physikalischen Geräte finden Sie ab Adresse 159 hex. Die Namen der ersten beiden physikalischen Geräte ändern Sie mit dem SID-Kommando

```
S159              (Sie geben dies ein)
159 54 "CRT:LST   (Ihre Eingabe beginnt mit ")
160 3A .          (Sie geben den Punkt ein)
```

Wenn Sie DDT benutzen, müssen Sie die hexadezimalen Werte der ASCII-Zeichen mit dem S-Kommando eingeben. Die ASCII-Zeichen und ihre hexadezimalen Werte sind:

```
ASCII  C  R  T  :  L  S  T
Hex    43 52 54 3A 4C 53 54
```

Sie geben das S-Kommando wie bei SID ein. Dann geben Sie die sieben Hex-Zahlen ein:

```
0159  54  43
015A  54  52
015B  59  54
015C  3A  3A
015D  43  4C
015E  52  53
015F  54  54
0160  3A  .      (Beenden Sie mit einem Punkt)
```

Kehren Sie zu CP/M zurück, und speichern Sie die richtige Anzahl von Blöcken in eine Datei names STAT2.COM. Versuchen Sie, das IOBYTE von 0 auf 1 zu setzen mit dem Kommando

```
STAT2 CON:=LST:
```

Die Drucker-Ausgabe sollte nun gleich der Konsol-Ausgabe sein. Schalten Sie den Drucker wieder ab mit dem Kommando

```
STAT2 CON:=CRT:
```

Mit dieser Methode kann man auch die Namen anderer Geräte in STAT ändern.

Nun werden wir einige neue Funktionen in die Druckerroutine des BIOS einbauen.

EINBAU EINER DRUCKER-BEREIT-ROUTINE

Computer kommunizieren mit ihrer Peripherie mit Hilfe von Ein-/Ausgabe-Registern oder Ports. Üblicherweise benutzt man bidirektionale Datenregister für den Transport der Information und ein weiteres bidirektionales Statusregister für die Abfrage der Bereitschaft. Das Statusregister wird automatisch auf „nicht-bereit" gesetzt, wenn die CPU ein Byte in das Datenregister schreibt.

Manchmal hat die CPU eine besondere Leitung für die Bedienung der Peripherie. Mit dieser Leitung kann die CPU unterbrochen werden, um eine Anfrage zu bedienen. Häufiger benutzt man bei der Kommunikation mit der Peripherie die „Looping"-Methode. Dabei prüft der Computer das Statusregister, um zu sehen, ob das Gerät bereit ist. Das Statusregister wird wiederholt geprüft, indem die hierfür benötigten Befehle wiederholt ausgeführt werden. Wenn das Statusregister anzeigt, daß das Gerät bereit ist, führt der Computer den Transfer durch und fährt dann mit seiner Arbeit fort.

Lassen Sie uns die „Looping"-Methode für eine Drucker-Ausgaberoutine betrachten. Die Befehle im BIOS könnten so aussehen:

```
LIST:
LISTT:
        IN      5
        ANI     1
        JZ      LISTT
```

In diesem Beispiel hat das Statusregister die Adresse 5, und das niedrigste Bit wird als Bereit-Anzeige benutzt. Der 8080-Befehl IN 5 liest den Statusport. Der folgende Befehl, ANI 1, führt eine logische Operation mit dem Akkumulator durch. Das Ergebnis ist null, wenn das Gerät nicht bereit ist. Daher bewirkt der dritte Befehl, JZ LISTT, einen Sprung zum Beginn der Schleife von drei Befehlen. Diese drei Befehle werden wiederholt so lange durchlaufen, bis das Gerät bereit ist. Wenn das Bereit-Bit anzeigt, daß das Gerät seine Arbeit beendet hat, wird die dem Befehl „JZ LISTT" folgende Instruktion ausgeführt. Der Computer sendet ein weiteres Byte zum Datenport und kehrt zurück. Da der Computer viel schneller ist als das Gerät, wird viel Zeit durch die Ausführung der obigen drei Befehle verschwendet.

Die „Looping"-Methode arbeitet zufriedenstellend, wenn der Drucker tatsächlich eingeschaltet ist. Unglücklicherweise zeigt meist das Statusbit Bereitschaft an, wenn der Drucker ausgeschaltet ist. Der Computer sendet dann Daten zum Drucker, und nichts passiert. Daher müssen wir zwei Dinge beachten – ob der Drucker eingeschaltet ist und ob das letzte Zeichen gedruckt ist. Das letztere haben wir betrachtet; wenden wir uns nun dem ersteren zu.

Manchmal gibt es eine einfache Lösung für dieses Problem. Wir haben uns nur eines der acht Bits des Statusregisters angesehen, nämlich das, das anzeigt, ob der Druckpuffer leer ist. Viele Computer benutzen ein weiteres Bit des Statusregisters zur Anzeige, ob das Gerät eingeschaltet ist. Dies Bit heißt „Data-Terminal-Ready" (DTR).

Wie man das Data-Terminal-Ready-Bit findet

Mit dem Assemblerprogramm in Abb. 3.5 können Sie herausfinden, ob Ihr Drucker-Statusregister ein DTR-Bit hat oder nicht. Beim standardmäßigen seriellen RS-232-Port findet man dieses Bit gewöhnlich auf Pin 20. Manchmal wird allerdings hierfür auch Pin 11 benutzt. Daher müssen Sie möglicherweise einen Draht des Druckerkabels umlöten.

Erstellen Sie das Quellprogramm nach Abb. 3.5. Suchen Sie in Ihrer BIOS- oder USER-Liste nach der Adresse des Drucker-Statusports, und ändern Sie entsprechend den Wert von PORT. Assemblieren Sie das Programm, und starten Sie es. Lassen Sie die ORG-Anweisung weg, wenn Sie den Microsoft-Assembler benutzen. Das Programm liest den Statusport und zeigt seinen Wert binär auf der Konsole an. Ist Ihr Drucker ausgeschaltet, dann schalten Sie ihn ein; ist er eingeschaltet, dann schalten Sie ihn aus. Falls sich eines der Bits ändert, wird der neue Wert auf dem Bildschirm angezeigt. Bei einigen Druckern kann es bis zu einer Minute dauern, bis sich das Bit nach dem Ausschalten ändert.

```
        TITLE 'Anzeige eines I/O Ports in binär'
        ;(Tagesdatum)
ORG     100H
;
PORT    EQU     5           ;Status-Port
BDOS    EQU     5
TYPEF   EQU     2           ;Konsole-Ausgabe
CSTATF  EQU     11          ;Konsole-Status
CR      EQU     13          ;Cursor Zeilenanfang
LF      EQU     10          ;Cursor nächste Zeile
;
START:
        LXI     SP,STACK
        IN      PORT        ;lesen
        MOV     H,A         ;retten
        CALL    BITS        ;Anzeige binär
NEXT:
        IN      PORT        ;nächste Eingabe
        MOV     L,A         ;retten
        CMP     H           ;Unterschied?
        JNZ     SHOW        ;ja
        PUSH    H
        MVI     C,CSTATF    ;Konsol-Status
        CALL    BDOS
        POP     H
        RRC                 ;teste Bit 0
        JC      0           ;Programmende
        JMP     NEXT
;
SHOW:
        CALL    BITS        ;Anzeige binär
        MOV     H,L         ;umschalten
        JMP     NEXT
;
BITS:                       ;konvertiere binär nach ASCII
        MOV     C,A
        MVI     B,8         ;8 Bits
BIT2:
        MOV     A,C
        ADD     A           ;shift links
```

Abb. 3.5: Programm zum Auffinden des DTR-Bits

```
        MOV     C,A
        MVI     A,0         ;null
        ACI     '0'         ;Carry + ASCII 0
        CALL    OUTT
        DCR     B           ;zählen
        JNZ     BIT2        ;8 mal
CRLF:                       ;neue Zeile
        MVI     A,CR
        CALL    OUTT
        MVI     A,LF
OUTT:                       ;Konsol-Ausgabe
        PUSH    H
        PUSH    B
        PUSH    PSW
        MVI     C,TYPEF     ;Konsol-Ausgabe
        MOV     E,A
        CALL    BDOS
        POP     PSW
        POP     B
        POP     H
        RET
        DS      12          ;Stack
STACK:
        END     START
```

Abb. 3.5: Programm zum Auffinden des DTR-Bits (Forts.)

Schalten Sie weiterhin Ihren Drucker ein und aus. Wenn Sie sehen, daß sich ein Bit ändert, notieren Sie, welches Bit sich geändert hat und seinen Wert (0 oder 1), wenn der Drucker aus ist. Nehmen wir z. B. folgendes an:

```
10110111    (Drucker an)
00110111    (Drucker aus)
```

In diesem Beispiel zeigt Bit 7 (das höchste Bit) an, daß der Drucker bereit ist (DTR), wenn es 1 ist. Das Bit wird auf 0 gesetzt, wenn der Drucker aus ist. Bei diesem Port zeigt Bit 0 an, ob der Druckpuffer leer ist. Wenn der Drucker an ist, aber besetzt, ist das Bitmuster

```
10110110    (Drucker an)
```

Ist der Drucker bereit, ein Zeichen zu empfangen, ist das Bitmuster

10110111 (Drucker an)

Durch Drücken irgendeiner Taste beenden Sie das Programm.

Sehen wir einmal, was das Programm macht. Wir beginnen mit den üblichen TITLE-, ORG- und EQU-Anweisungen. Die Adresse des Statusports sei hier einmal 5.

Die eigentlichen Befehle beginnen bei dem Label START. Der Stack ist hier am Ende des Programms und nicht bei 100 hex, wie in dem Programm nach Abb. 3.1. Das Statusregister wird in den Akkumulator gelesen und in das H-Register geladen. Der Wert wird mit dem Unterprogramm BITS auf der Konsole angezeigt.

Wieder wird der Port gelesen, aber diesmal wird sein Wert in das L-Register geladen. Die beiden Werte werden verglichen. Bei Ungleichheit wird der neue Wert mit dem Unterprogramm BITS ausgegeben und in das H-Register geladen. Sind die Werte gleich, wird nichts ausgegeben. Nur der Konsolstatus wird geprüft, um zu entscheiden, ob das Programm beendet werden soll. Falls nicht, wiederholt sich der Ablauf.

Das Unterprogramm BITS konvertiert eine Binärzahl im Akkumulator in eine ASCII-Zeichenfolge von acht Nullen oder Einsen und zeigt dann das Ergebnis auf der Konsole an. Das Byte wird zunächst in das C-Register geladen, und in das B-Register wird eine 8 geladen, die Anzahl der auszugebenden Zeichen.

Die Schleife beginnend bei BIT2 wird dann achtmal ausgeführt. Bei jedem Durchlauf wird der momentane Wert des Bytes mit dem Befehl ADD A zu sich selbst addiert. Dies ist eine logische Verschiebung nach links. Alle Bits des Akkumulators werden um eine Stelle verschoben. Das höchstwertige Bit wandert in das Carry-Flag; das niedrigstwertige Bit wird gelöscht. Der neue Wert wird für den nächsten Schritt in das C-Register geladen.

Das Carry-Flag wird auf 1 gesetzt, wenn das höchstwertige Bit 1 war. Es wird gelöscht, wenn das Bit 0 war. Wir werden eine Eins ausgeben, wenn das Carry-Flag gesetzt ist; andernfalls geben wir eine Null aus. Das erreichen wir durch Löschen des Akkumulators; dann addieren wir eine ASCII-Null und das Carry-Bit. Die Befehle hierfür sind:

```
MVI     A,0   ; Akkumulator löschen
ACI     '0'   ; Carry und ASCII-Null addieren
```

Gehen wir durch die beiden ersten Durchläufe mit einem Beispiel. Nehmen wir die Binärzahl 10101010 (AA hex) an. Wird diese Zahl zu sich

selbst addiert, ist das Ergebnis 01010100, und das Carry-Flag ist gesetzt. Unser Algorithmus zeigt eine Eins an. Die nächste Addition ergibt 10101000 und setzt das Carry-Flag auf 0. Das Unterprogramm gibt nun eine Null aus.

Diesen Algorithmus kann man sowohl mit einer 8080- als auch mit einer Z80-CPU benutzen, obgleich er mit einem Z80-Computer effektiver wird, indem man das C-Register direkt logisch verschiebt. Alle üblichen Algorithmen für die Konvertierung von Zahlen findet man in der Literatur.

Wenn Sie ein Drucker-Bereit-Bit gefunden haben, zeigt Ihnen der nächste Abschnitt, wie man einen DTR-Test in das BIOS einbaut.

Die Drucker-Bereit-Prüfung

Wie bereits oben erwähnt, gibt es nicht bei allen Computern ein DTR-Bit. Wenn Sie jedoch ein Drucker-Bereit-Bit entdeckt haben, können Sie einen Test in Ihr BIOS einbauen, der Ihnen mitteilt, daß gedruckt werden soll, aber Ihr Drucker nicht eingeschaltet ist. Dieser Test prüft die Drukker-Bereit-Anzeige. Ist der Drucker ausgeschaltet, wird Ihre Konsolglocke ertönen, und eine entsprechende Meldung wird ausgegeben. Ist der Drucker eingeschaltet, wird der nächste Befehl (gewöhnlich der Test, ob der Druckerpuffer leer ist) ausgeführt.

Angenommen, der Drucker-Statusport hat den Namen LSTATP, die Data-Terminal-Ready-Maske den Namen DTRMSK und die normale Drucker-Bereit-Maske den Namen LMSK. Die Ausgaberoutine für die physikalische Konsole ist CONO2, da wir zwischen der physikalischen und der logischen Konsol-Ausgabe unterscheiden wollen. Die Original-

```
LIST:                              ;logische Drucker-Ausgabe
LISTT:
        IN      LSTATP    ;teste Status
        ANI     LMSK      ;Maske für die Ausgabe
        JZ      LISTT     ;wiederholen bis bereit
        MOV     A,C
        OUT     LDATAP    ;Ausgabe
        RET
*PROGX*
```

Abb. 3.6: Original-Version einer typischen Drucker-Routine

Druckroutine könnte aussehen wie in Abb. 3.6, während die neue aussieht wie in Abb. 3.7.

```
CR      EQU     13              ;Cursor zu Zeilenanfang
LF      EQU     10              ;Cursor nächste Zeile
BEL     EQU     7               ;ASCII bell
;
LIST:
LISTT:
        IN      LSTATP          ;teste Status
        ANI     DTRMSK          ;Drucker an?
        JNZ     LIST2           ;ja
        PUSH    H               ;Drucker aus
        PUSH    B
        LXI     H,MESG          ;Adresse
        MVI     B,AROUND-MESG   ;Länge
LLOOP:
        MOV     C,M
        CALL    CONO2           ;Konsolausgabe
        INX     H               ;Zeiger
        DCR     B               ;zählen
        JNZ     LLOOP           ;weiter machen
        POP     B
        POP     H
        JMP     AROUND          ;Text überspringen
MESG:
        DB      BEL,CR,LF
        DB      ' DRUCKER ANSCHALTEN ',CR,LF
AROUND:
LIST3:
        IN      LSTATP
        ANI     DTRMSK          ;Drucker an?
        JZ      LIST3           ;nein
LIST2:
        IN      LSTATP          ;teste Status
        ANI     LMSK            ;Maske für Ausgabe
        JZ      LISTT           ;wiederholen bis bereit
        MOV     A,C
        OUT     LDATAP          ;Ausgabe
        RET
```

Abb. 3.7: Die geänderte Version der Drucker-Routine

Die ersten drei Zeilen definieren die Symbole CR (Carriage Return – Wagenrücklauf), LF (Line Feed – Zeilentransport) und BEL (Console Bell – Konsolglocke). Danach folgt der eigentliche Code. Der Drucker-Statusport (LSTATP) wird gelesen. Alle Bits außer dem DTR-Bit werden mit der Maske DTRMSK gelöscht. Ist dies Bit eins, wird die Nullanzeige gelöscht. Der Befehl

```
JNZ  LIST2
```

springt nach LIST2, der Original-Druckroutine.

Ist das DTR-Bit jedoch null, zeigt es an, daß der Drucker ausgeschaltet ist. In diesem Fall ertönt die Konsolglocke, und die Meldung

```
DRUCKER ANSCHALTEN
```

erscheint auf der Konsole. Wieder wird der Statusport beobachtet, beginnend bei der Marke LIST3. In dieser Schleife (drei Befehle) wird gewartet, bis der Drucker eingeschaltet ist. Erst dann fährt das Programm mit der Druckerausgabe fort.

Fügen Sie diese neue Passage in Ihr BIOS ein. Assemblieren Sie es, und laden Sie es mit dem Debugger. Schalten Sie Ihren Drucker an mit Control-P, und geben Sie das Kommando DIR ein. Während der Drucker arbeitet, schalten Sie ihn aus. Die Konsolglocke wird ertönen, und die Meldung

```
DRUCKER ANSCHALTEN
```

wird erscheinen. Wird der Drucker wieder eingeschaltet, sollte die Druckausgabe an der Stelle fortgesetzt werden, wo sie unterbrochen wurde. Diese Routine ist auch bei Programmen wie WordStar und BASIC anwendbar (natürlich werden dort andere Kommandos benutzt, um den Drucker anzuschalten). (Die richtige Fortsetzung ist nur gewährleistet, wenn der Drucker nicht über einen eigenen Druckpuffer verfügt, der beim Ausschalten gelöscht wird. Anmerkung des Übersetzers.)

STEUERUNG DER LISTENAUSGABE MIT DEM IOBYTE

An früherer Stelle in diesem Kapitel haben wir das IOBYTE in die Konsol-Ausgaberoutine einbezogen. Dafür benutzten wir die niederwertigen zwei Bits des IOBYTE. Jetzt werden wir einige neue Funktionen zur logischen Drucker-Ausgabe hinzufügen, wobei wir die beiden höchstwertigen Bits benutzen.

Eine dieser Funktionen ist leicht zu implementieren. Diese Routine ist immer nützlich, wenn man ein Programm mit langer Ausgabe testet, die Ausgabe selbst jedoch nicht interessiert. Man nennt dies auch „Bit Bukket" (Bit-Eimer). Darüber hinaus wird sie es uns ermöglichen, die Listenausgabe zum Drucker zu senden, wie gewöhnlich, zur Konsole oder in einen separaten Speicherbereich.

Wir definieren für das IOBYTE den Wert 0 für die normale Drucker-Ausgabe. Der Wert 40 hex sendet die Ausgabe zur Konsole, und der Wert 80 hex läßt die Ausgabe verfallen - d.h. die Ausgabe verschwindet. Mit dem Wert C0 hex wird hier die Ausgabeliste in einen separaten Speicherbereich gesendet, den man „Cache" (Versteck) nennt. Diese Routine werden wir allerdings zunächst nicht implementieren. Es folgt die Zuordnung; sie sollte als Kommentar in das BIOS-Quellprogramm aufgenommen werden.

IOBYTE	*Funktion*
00	Drucker-Ausgabe
40	Konsol-Ausgabe
80	Bit Bucket
C0	Cache-Speicher

Die Drucker-Ausgaberoutine beginnt mit den folgenden Befehlen:

```
LIST:                   ; logisch
LISTT:                  ; physikalisch
        IN      LSTATP  ; Statustest
        ...     ...
```

LIST bezeichnet die logische Ausgabe, und LISTT bezieht sich physikalisch auf den Drucker. Zwischen diesen beiden Marken werden wir nun Befehle einfügen.

Wie bei der Konsol-Ausgaberoutine müssen wir das IOBYTE untersuchen. Die neuen Befehle stehen zwischen den Marken LIST und LISTT. Wir lesen das IOBYTE. Da wir uns nur für die beiden höchstwertigen Bits interessieren, maskieren wir sie mit dem Befehl „ANI 0C0H". Dieser Befehl löscht die unteren sechs Bits. Ist das Ergebnis 0, senden wir die Ausgabe zum Drucker. Ist das Ergebnis 40 hex, geht die Ausgabe zur Konsole. Ist das Ergebnis 80 hex, kehren wir einfach zurück. Und ist schließlich das Ergebnis C0 hex, speichern wir die Ausgabe im „Cache". Zunächst werden wir diese Funktion nicht implementieren und kehren daher einfach zurück. Das Quellprogramm hierfür ist in Abb. 3.8.

```
LIST:                                   ;logische Drucker-Ausgabe
        LDA     IOBYTE
        ANI     0C0H        ;Maske für Bits 6,7
        JZ      LISTT       ;Drucker-Ausgabe
        CPI     40H
        JZ      CONO2       ;Konsole Ausgabe
        CPI     80H
        RZ                  ;Bit-Eimer
;
;Cache-Routine hier einfügen
;
        RET                 ;(vorläufig)
;
LISTT:                      ;physikalische Drucker-Ausgabe
        ...         ...
```

Abb. 3.8: Einbeziehung des IOBYTE in die Drucksteuerung

Beachten Sie, daß die Listausgabe bei CONO2 fortgesetzt wird, wenn der IOBYTE-Wert 40 hex ist, und nicht bei CONOUT, der logischen Konsol-Ausgaberoutine. Dadurch wird sichergestellt, daß die Ausgabe, die für die Konsole bestimmt ist, nicht zum Drucker zurückgeht, wenn das unterste Bit des IOBYTE 1 ist.

Assemblieren Sie die neuen Befehle in Ihrem BIOS oder USER, laden und testen Sie die neue Version mit dem Debugger. Setzen Sie das IOBYTE mit dem Debugger auf 40 hex. Schalten Sie den Drucker an mit Control-P. Jedes Zeichen sollte nun zweimal auf der Konsole erscheinen, da sowohl die logische Konsol-Ausgabe als auch die logische Drucker-Ausgabe physikalisch zur Konsole gesendet werden. Schalten Sie den Drucker mit einem weiteren Control-P wieder ab. Sind Sie mit Ihrer neuen Version zufrieden, dann speichern Sie sie mit SAVEUSER oder mit SYSGEN auf die Systemspuren einer Diskette.

Wir werden nun eine Routine einfügen, mit der wir Druckausgaben in den Speicher senden.

SPEICHERUNG DER LISTENAUSGABE IN EINEM PUFFERSPEICHER

Manchmal ist es wünschenswert, die Druckausgabe in einem Pufferspeicher oder „Cache" zu speichern, anstatt sie zu drucken. Man kann ihn dann in einer Diskettendatei speichern, um ihn zu editieren oder in einer

Veröffentlichung zu verwenden. Alle Computerausgaben in diesem Buch wurden übrigens auf diese Weise erstellt.

Ausgabe in einen „Cache" wird mit zwei Zeigern durchgeführt. Der erste Zeiger gibt an, wo das nächste Byte gespeichert werden soll. Dieser Zeiger wird zu Beginn auf den Pufferanfang eingestellt und jedesmal erhöht, nachdem ein Byte in den Puffer geschrieben wurde. Am Ende des Programms wird 1A hex, die Datei-Ende-Kennung, an das Listenende gesetzt, der zweite Zeiger wird auf das Ende, der erste auf den Beginn der Liste eingestellt. Die beiden Zeiger stehen unmittelbar vor dem Pufferspeicher; sie sind beide zwei Bytes groß.

Für diesen Puffer müssen wir einen Bereich im Speicher finden, der niemals vom Betriebssystem CP/M benutzt wird. Andernfalls könnte der „Cache" versehentlich überschrieben werden. Es gibt mehrere Wege, dies zu tun. Ein North Star Horizon, z.B., benutzt den Bereich von E800 bis EBFF hex als Disketten-Controller-Speicher. Da CP/M einen zusammenhängenden Speicherblock benötigt, ist die höchste CP/M-Adresse für diesen Computer E7FF hex. Daher ist der Bereich von F000 bis FFFF hex frei. Eine andere Möglichkeit besteht darin, ein kleineres CP/M-System mit MOVCPM zu generieren. Der Bereich oberhalb von CP/M kann dann für den Speicherpuffer benutzt werden.

Im vorigen Abschnitt haben wir einen IOBYTE-Wert von C0 hex vorgesehen für die Anzeige, daß Druckausgabe in einen Speicherpuffer geschrieben werden soll. Wir werden nun die hierfür nötige Routine schreiben. Als Speicher wählen wir den Bereich von F000 bis FFFF hex (die obersten 4K Bytes). Die Zeiger stehen bei F000 und F002 hex. Der Puffer selbst beginnt bei F004 hex.

Es gibt noch eine Schwierigkeit, die wir beachten müssen. Der Puffer wird überlaufen, wenn zu viele Bytes in ihn eingetragen werden. Der Zeiger wird dann aus dem Puffer herauslaufen, größer als FFFF hex in diesem Fall. Wird jedoch FFFF hex, die größte 16-Bit-Zahl, um eins erhöht, ist das Ergebnis null. Der Zeiger zeigt dann auf den Anfang des Speichers und nicht auf das Ende. (Man nennt dies „Wrap Around".) Wie wir in Kapitel 1 gesehen haben, hält CP/M einige wichtige Informationen am Beginn des Speichers. Daher müssen wir ein „Wrap Around" verhindern, um diese wichtigen Informationen nicht zu verändern.

Wir werden den Zeiger auf den Anfang des Puffers setzen und die Konsolglocke ertönen lassen, falls „Wrap Around" droht. Dadurch schützen wir das CP/M-System. Natürlich wird die ursprüngliche Information im Puffer dadurch zerstört, aber es ist nicht sehr wahrscheinlich, daß dies auftritt. Sie werden sehen, daß ein 4-KByte-Puffer für die meisten Anwendungen ausreicht.

Am Ende des Programms können wir mit dem Debugger die Information im „Cache" in die TPA ab 100 hex umspeichern. Wir beenden mit Control-C und speichern die Information in einer Diskettendatei. In Kapitel 7 werden wir ein Programm schreiben, das den Pufferspeicher direkt in eine Datei schreibt. Dieses Programm findet die Dateigröße anhand der Zeiger heraus.

Um den Pufferspeicher zu implementieren, brauchen wir zwei Folgen von Befehlen. Die erste Folge zum Initialisieren der Pointer und Setzen des Dateiende-Kennzeichens wird in die Warm- und die Kaltstartroutine des BIOS oder USER eingefügt. Mit der zweiten Folge werden die Bytes in den Puffer eingetragen und der Hauptzeiger erhöht. Diese Folge steht in der Drucker-Ausgaberoutine. Wir beginnen mit der Folge, die die Zeiger initialisiert.

Initialisierung der Zeiger für den Pufferspeicher

In diesem Abschnitt ändern wir die Warm- und die Kaltstartroutine von BIOS oder USER und fügen die Befehle für die Initialisierung der Pufferzeiger und das Schreiben der Dateiendekennung ein. Als erstes definieren wir vier Symbole – die Namen der beiden Zeiger, den Namen des Puffers und die Dateiendekennung. Plazieren Sie diese vier Zeilen nahe dem Anfang des Quellprogramms:

```
MPOINT  EQU  0F000H     ;Zeiger zum Anfang
MMAX    EQU  MPOINT+2   ;Zeiger zum Ende
MBUFF   EQU  MMAX+2     ;Pufferbeginn
EOF     EQU  1AH        ;Dateiendekennung
```

Suchen Sie den Warmstartvektor Ihres BIOS oder USER. Dies erkennen Sie am zweiten Sprungvektor. Verfolgen Sie die Warmstart-Befehle, bis Sie den abschließenden RET-Befehl finden. Fügen Sie die Befehle nach Abb. 3.9 direkt davor ein.

Sehen wir uns an, wie dieser Teil funktioniert. Als erstes prüfen wir, ob die Druckerausgabe in den Pufferspeicher gelangen soll. Diese Information ist in den beiden höchsten Bits des IOBYTE verschlüsselt. Der erste neue Befehl lädt das IOBYTE in den Akkumulator:

```
LDA   IOBYTE
```

Alle außer den beiden höchsten Bits werden gelöscht mit dem Befehl

```
ANI   0C0H
```

```
WARM:
        ...
        LDA     IOBYTE
        ANI     0C0H    ;Maske für Drucker
        CPI     0C0H    ;Speicher?
        RNZ             ;nein
        PUSH    H       ;retten Register
        PUSH    D
        LXI     D,MBUFF ;Pufferstart
        LHLD    MPOINT  ;Zeiger
        MOV     A,L     ;teste unteres Byte
        CMP     E       ;Zeiger gelöscht?
        JNZ     MEM3    ;nein
        MOV     A,H     ;teste oberes Byte
        CMP     D       ;gelöscht?
        JZ      MEM4    ;ja
MEM3:                   ;löschen Zeiger
        MVI     M,EOF   ;Dateiende-Kennzeichen
        SHLD    MMAX    ;retten letzte Adresse
        LXI     H,MBUFF ;Puffer-Beginn
        SHLD    MPOINT  ;retten Zeiger
MEM4:
        POP     D       ;laden
        POP     H
        RET
```

Abb. 3.9: Das Setzen der Zeiger

Wenn das Ergebnis nicht C0 hex ist, beenden wir den Warmstartvorgang mit RNZ.

Wenn das Ergebnis jedoch C0 hex ist, soll die Ausgabe in den Pufferspeicher gehen. Die Register HL und DE werden mit PUSH-Befehlen gerettet. Dann prüfen wir, ob der Zeiger bereits auf den Pufferanfang zeigt. Wenn ja, ist die Arbeit beendet. Die Register HL und DE werden mit POP-Befehlen wiederhergestellt, und ein RET-Befehl wird ausgeführt.

Ist der Zeiger aber nicht zurückgesetzt, zeigt er zum Pufferende. Hierhin wird eine Dateiendekennung (1A hex) geschrieben. Die Pufferadresse wird im zweiten Zeiger gespeichert (MMAX), und der Hauptzeiger (MPOINT) wird auf den Pufferanfang gesetzt. Die Register werden wiederhergestellt, und ein RET-Befehl wird ausgeführt.

Da wir auch bei einem Kaltstart die Pufferzeiger initialisieren müssen, suchen wir den Kaltstarteingang. Der erste Vektor am Beginn des BIOS oder USER zeigt auf ihn. In einem früheren Abschnitt dieses Kapitels haben wir hier zwei Befehle zum Initialisieren des IOBYTE eingefügt. Fügen Sie die beiden folgenden Befehle unmittelbar danach ein:

```
COLD:
        ...
        LXI     H,MBUFF
        SHLD    MPOINT
```

Jetzt werden wir die übrigen Befehle einfügen.

Die Befehle für Speicherung einer Liste im Puffer

Da wir nun die Befehle für die Initialisierung der Pufferzeiger eingefügt haben, können wir den Code für das eigentliche Speichern der Daten im Puffer einbauen. Die neuen Befehle, wie in Abb. 3.10 gezeigt, werden zwischen die Befehle RZ und RET im Druckausgabeteil nach Abb. 3.8 eingesetzt.

Dieser Abschnitt besteht aus zwei Teilen. Im ersten Teil werden die Bytes in den Puffer gespeichert und der Pufferzeiger erhöht. Im zweiten Teil wird auf Pufferüberlauf geprüft. Wir beginnen mit dem Retten des Inhalts des HL-Registers durch einen PUSH-Befehl. Der Hauptzeiger wird geladen und für die Adressierung des Bytes im Puffer benutzt. Der Zeiger wird erhöht und anschließend auf Überlauf geprüft.

Falls kein Überlauf erfolgte, wird der Zeiger gespeichert und ein RET-Befehl ausgeführt. Ist der Zeiger jedoch null geworden, wird er auf den Beginn des Puffers gesetzt, und die Konsolglocke ertönt.

Bauen Sie die restlichen Befehle für den Speicherpuffer in das BIOS ein. Assemblieren Sie die neue Version, und testen Sie sie. Laden Sie das Programm mit dem Debugger.

Es ist von größter Wichtigkeit, daß der Hauptzeiger vor Benutzung des „Cache" richtig gesetzt wird. Andernfalls speichert CP/M die Daten an der falschen Stelle, was zu unvorhersehbaren Ergebnissen führt. Die beiden Befehle, die wir in die Kaltstartroutine eingefügt haben, initialisieren den Hauptzeiger, wann immer Sie CP/M starten. Wir wollen jedoch die Routine testen, bevor sie auf die Systemspuren geschrieben wird. Daher müssen wir dieses eine Mal die Zeiger selbst initialisieren.

```
;
;Drucker Ausgabe in einen Speicherpuffer
;
        PUSH    H           ;retten
        LHLD    MPOINT      ;Zeiger
        MOV     M,C         ;Byte in Speicher
        INX     H           ;erhöhen Zeiger
        MOV     A,H         ;testen ob
        ORA     L           ;Nulldurchgang
        JNZ     MEM2        ;weiter
;
;Puffer ist voll
;Konsol-Glocke als Warnung
;
        PUSH    D
        MVI     C,BEL
        CALL    CONO2       ;Glocke
        POP     D
        LXI     H,MBUFF     ;Beginn
MEM2:                       ;ändern Zeiger
        SHLD    MPOINTER    ;retten
        POP     H
        RET
```

Abb. 3.10: Das Ablegen der Druckausgabe im Speicher

Setzen Sie mit dem S-Kommando des Debuggers den Hauptzeiger auf F004 hex. Die Anweisungen dazu sind:

```
SF000           (Sie geben dies ein)
F000 XX 4       (Sie geben 4 ein)
F001 XX F0      (Sie geben F0 ein)
F002 XX .       (Sie geben den Punkt ein)
```

Setzen Sie das IOBYTE auf C0 hex, wiederum mit dem S-Kommando:

```
S3              (Sie geben dies ein)
0003 XX C0      (Sie geben C0 ein)
0004 XX .       (Sie geben den Punkt ein)
```

Führen Sie mit Control-C einen Warmstart aus. Sie sind nun auf der CP/M-Systemebene. Schalten Sie den Drucker mit Control-P an, und geben Sie das Kommando DIR. Auf dem Drucker sollte keine Ausgabe erscheinen, da wir die Ausgabe in den Puffer umleiten. Führen Sie erneut mit Control-P einen Warmstart aus. Dadurch wird der Drucker wieder abgeschaltet, und die Zeiger werden zurückgesetzt.

Laden Sie den Debugger, und inspizieren Sie den Beginn des Puffers mit dem D-Kommando:

```
DF000,F03F
```

Die ASCII-Zeichenfolge des vorangegangenen DIR-Kommandos sollte auf der rechten Seite des Bildschirms erscheinen. Sehen Sie sich den zweiten Zeiger bei Adresse F002 und F003 an. Dieser Zeiger verweist auf das Ende des Textes. Das entsprechende Byte sollte die Dateiendekennung 1A hex enthalten.

Mit dem Debuggerkommando M können Sie die Information im Pufferspeicher nun nach 100 hex umspeichern. Führen Sie einen Warmstart aus, und speichern Sie die Information in einer Diskettendatei. Sie sollten jetzt mit SAVEUSER oder SYSGEN die momentane BIOS- oder USER-Version auf den Systemspuren einer Diskette speichern. Schalten Sie Ihren Computer aus und wieder ein, und führen Sie einen Kaltstart aus. Überprüfen Sie mit dem Debugger, ob der Hauptzeiger richtig initialisiert ist.

Die Assemblerliste einer Reihe von USER-Routinen zeigt Abb. 3.11. Diese Liste beinhaltet alle in diesem Kapitel beschriebenen Funktionen. Es läuft in einer Lifeboat CP/M Version 2.2 auf einem 56K-Byte North Star System.

```
        TITLE 'Beispiel für ein BIOS/USER-Programm'
        ;
        ;(Tagesdatum)
        ;
DA00    ORG     0DA00H
        ;
0003 =  IOBYTE  EQU  3
0003 =  CSTATP  EQU  3              ;Konsole-Status
0002 =  CDATAP  EQU  CSTATP-1       ;Konsole-Daten
```

Abb. 3.11: Die USER-Routinen für die in diesem Kapitel beschriebenen Funktionen

```
0001 =  COMSK   EQU   1              ;Konsole-Ausgabe-Maske
0002 =  CIMSK   EQU   2              ;Konsole-Eingabe-Maske
0005 =  LSTATP  EQU   5              ;Drucker-Status
0004 =  LDATAP  EQU   LSTATP-1       ;Drucker-Daten
0001 =  LMSK    EQU   1              ;Drucker-Ausgabe-Maske
0080 =  DTRMSK  EQU   80H            ;Drucker-bereit-Maske
F000 =  MPOINT  EQU   0F000H         ;Zeiger auf Anfang
F002 =  MMAX    EQU   MPOINT+2       ;Zeiger auf Ende
F004 =  MBUFF   EQU   MMAX+2         ;Puffer-Beginn
        ;
000D =  CR      EQU   13             ;Return
000A =  LF      EQU   10             ;Line Feed
0007 =  BEL     EQU   7              ;ASCII bell
001A =  EOF     EQU   1AH            ;Dateiende-Kennzeichen
        ;
        START
DA00 C399DA     JMP   COLD           ;Kaltstart
DA03 C3A5DA     JMP   WARM           ;Warmstart
DA06 C3CDDA     JMP   CSTAT          ;Konsole-Status
DA09 C3D5DA     JMP   CONIN          ;Konsole-Eingabe
DA0C C318DA     JMP   CONOUT         ;Konsole-Ausgabe
DA0F C32BDA     JMP   LIST           ;Drucker-Ausgabe
DA12 C3E0DA     JMP   PUNCH          ;Stanzer-Ausgabe
DA15 C3E1DA     JMP   READER         ;alternatives
                                     Eingabe-Gerät
        ;
        CONOUT:                      ;Konsole-Ausgabe
DA18 3A0300     LDA   IOBYTE         ;Wert laden
DA1B E601       ANI   1              ;Maske für Bit 0
DA1D C455DA     CNZ   LISTT          ;Drucker-Ausgabe
        CONO2:                       ;normale Konsole-Ausgabe
DA20 DB03       IN    CSTATP         ;Status
DA22 E601       ANI   COMSK          ;Maske für Ausgabe
DA24 CA20DA     JZ    CONO2          ;weiter bis bereit
DA27 79         MOV   A,C            ;Daten Byte
DA28 D302       OUT   CDATAP         ;Ausgabe
DA2A C9         RET
        ;
        LIST:                        ;logische Drucker-Ausgabe
```

Abb. 3.11: Die USER-Routinen für die in diesem Kapitel beschriebenen Funktionen (Forts.)

```
DA2B 3A0300      LDA   IOBYTE
DA2E E6C0        ANI   0C0H        ;Maske für Bits 6,7
DA30 CA55DA      JZ    LISTT       ;Drucker-Ausgabe
DA33 FE40        CPI   40H
DA35 CA20DA      JZ    CONO2       ;Konsole-Ausgabe
DA38 FE80        CPI   80H
DA3A C8          RZ                ;Bit-Eimer
                 ;
                 ;Drucker-Ausgabe in einen Speicherpuffer
                 ;
DA3B E5          PUSH  H           ;retten
DA3C 2A00F0      LHLD  MPOINT      ;Zeiger
DA3F 71          MOV   M,C         ;Byte in Speicher
DA40 23          INX   H           ;erhöhen Zeiger
DA41 7C          MOV   A,H         ;test ob
DA42 B5          ORA   L           ;Nulldurchgang
DA43 C250DA      JNZ   MEM2        ;weiter
                 ;
                 ;Puffer ist voll
                 ;Konsolglocke als Warnung
                 ;
DA46 D5          PUSH  D
DA47 0E07        MVI   C,BEL
DA49 CD20DA      CALL  CONO2       ;Glocke
DA4C D1          POP   D
DA4D 2104F0      LXI   H,MBUFF     ;Beginn
        MEM2:                      ;ändern Zeiger
DA50 2200F0      SHLD  MPOINT      ;retten
DA53 E1          POP   H
DA54 C9          RET
                 ;
                 LISTT:            ;physikalische Drucker-
                                   Ausgabe
DA55 DB05        IN    LSTATP      ;teste Status
DA57 E680        ANI   DTRMSK      ;Drucker an?
DA59 C28EDA      JNZ   LIST2       ;ja
DA5C E5          PUSH  H           ;Drucker aus
DA5D C5          PUSH  B
DA5E 2171DA      LXI   H,MESG      ;Adresse
```

Abb. 3.11: Die USER-Routinen für die in diesem Kapitel beschriebenen Funktionen (Forts.)

```
DA61 0616          MVI   B,AROUND-
                         MESG          ;Länge
        LLOOP:
DA63 4E            MOV   C,M
DA64 CD20DA        CALL  CONO2         ;Konsol-Ausgabe
DA67 23            INX   H             ;Zeiger
DA68 05            DCR   B             ;zählen
DA69 C263DA        JNZ   LLOOP         ;weiter
DA6C C1            POP   B
DA6D E1            POP   H
DA6E C387DA        JMP   AROUND        ;Text überspringen
        MESG:
DA71 070D0A        DB    BEL,CR,LF
DA74 205455        DB    ' TURN PRINTER ON ',CR,LF
        AROUND:
        LIST3:
DA87 DB05          IN    LSTATP
DA89 E680          ANI   DTRMSK        ;Drucker an?
DA8B CA87DA        JZ    LIST3         ;nein
        LIST2:
DA8E DB05          IN    LSTATP        ;teste Status
DA90 E601          ANI   LMSK          ;Maske für Ausgabe
DA92 CA55DA        JZ    LISTT         ;weiter bis bereit
DA95 79            MOV   A,C
DA96 D304          OUT   LDATAP        ;Ausgabe
DA98 C9            RET
        ;
        COLD:                          ;Kaltstart-Eingang
DA99 3E00          MVI   A,0
DA9B 320300        STA   IOBYTE        ;löschen
DA9E 2104F0        LXI   H,MBUFF
DAA1 2200F0        SHLD  MPOINT        ;löschen
DAA4 C9            RET
        ;
        WARM:                          ;Warmstart-Eingang
DAA5 3A0300        LDA   IOBYTE
DAA8 E6C0          ANI   0C0H          ;Maske für Drucker
DAAA FEC0          CPI   0C0H          ;Speicher?
DAAC C0            RNZ                 ;nein
```

Abb. 3.11: Die USER-Routinen für die in diesem Kapitel beschriebenen Funktionen (Forts.)

```
DAAD E5          PUSH  H          ;retten Register
DAAE D5          PUSH  D
DAAF 1104F0      LXI   D,MBUFF    ;Puffer-Beginn
DAB2 2A00F0      LHLD  MPOINT     ;Zeiger
DAB5 7D          MOV   A,L        ;teste unteres Byte
DAB6 BB          CMP   E          ;Zeiger gelöscht?
DAB7 C2BFDA      JNZ   MEM3       ;no
DABA 7C          MOV   A,H        ;teste oberes Byte
DABB BA          CMP   D          ;gelöscht?
DABC CACADA      JZ    MEM4       ;ja
        MEM3:                     ;löschen Zeiger
DABF 361A        MVI   M,EOF      ;markiere Pufferende
DAC1 2202F0      SHLD  MMAX       ;retten letzte Adresse
DAC4 2104F0      LXI   H,MBUFF    ;Puffer-Beginn
DAC7 2200F0      SHLD  MPOINT     ;retten Zeiger
        MEM4:
DACA D1          POP   D          ;laden
DACB E1          POP   H
DACC C9          RET              ;original ret.
        ;
        ;notwendige Routinen, die im Text nicht diskutiert sind:
        ;
        CSTAT:                    ;Konsol-Eingabe-Status
DACD DB03        IN    CSTATP     ;lies Status
DACF E602        ANI   CIMSK      ;Maske für Eingabe
DAD1 C8          RZ               ;nicht bereit
DAD2 3EFF        MVI   A,0FFH
DAD4 C9          RET              ;bereit
        CONIN:
DAD5 CDCDDA      CALL  CSTAT
DAD8 CAD5DA      JZ    CONIN      ;nicht bereit
DADB DB02        IN    CDATAP     ;Byte einlesen
DADD E67F        ANI   7FH        ;Maske parity
DADF C9          RET
        PUNCH:
DAE0 C9          RET
        READER:
DAE1 C9          RET
        ;
DAE2             END
```

Abb. 3.11: Die USER-Routinen für die in diesem Kapitel beschriebenen Funktionen (Forts.)

ZUSAMMENFASSUNG

In diesem Kapitel haben wir detailliert die CP/M BIOS- und USER-Routinen erforscht. Wir haben mehrere nützliche Funktionen entwickelt und implementiert, um Fähigkeiten Ihres CP/M-Systems zu erweitern, einschließlich einer Routine zum An- und Abschalten des Druckers, einer Drucker-Bereit-Routine und einer Routine zur Umleitung der Druckausgabe in einen Speicherpuffer. Darüber hinaus könnten Sie weitere Funktionen einbauen, wie z.B. Umleiten der logischen Stanzer-Ausgabe zu einem Telefonmodem oder Empfang der Konsol-Eingabe von der Drukkertastatur. Das alles überlassen wir Ihnen zur Übung.

Kapitel 4

Das Einrichten einer Makro-Bibliothek

EINFÜHRUNG

In diesem Kapitel werden wir das Konzept der Makrobefehle, auch Makros genannt, einführen. Wir werden mehrere wirksame Makros entwickeln, die weiterhin in diesem Buch benutzt werden. Wir beginnen mit allgemeinen Makros für die Pflege der Versionsnummer und zum Retten und Restaurieren des Stackpointers. Dann werden wir Makros zum Umspeichern von Daten, zum Füllen des Speichers mit einer Konstanten, zum Vergleichen, zur Umwandlung von Klein- und Großbuchstaben, für eine 16-Bit-Subtraktion und zur Konvertierung eines nichteindeutigen in einen eindeutigen Dateinamen schreiben.

MAKROS

Ein Makrobefehl, kurz Makro genannt, ist eine Assembleranweisung, die eine Gruppe von Anweisungen, Befehlen oder Makros definiert. Ein Makro besteht aus zwei Teilen – der Definition und einem oder mehreren Befehlen. Der Name des Makros ist verknüpft mit einer Gruppe von Befehlen, die es definiert. Immer, wenn der Name des Makros in einem Computerprogramm erscheint, setzt der Assembler hierfür die entsprechenden Befehle ein. Das nennt man Makroexpansion. Die folgende Befehlssequenz kann man mit dem Makro SAVE definieren:

```
PUSH  H
PUSH  D
PUSH  B
PUSH  PSW
```

Dadurch werden immer, wenn der Name SAVE im Programm erscheint, die entsprechenden vier Befehle eingesetzt. Das gegenteilige Makro namens UNSAVE kann die umgekehrte Operation ausführen:

```
POP   PSW
POP   B
```

```
POP  D
POP  H
```

Die Makrodefinition wird an den Anfang des Programms oder in eine separate Datei, die man Makrobibliothek nennt, gestellt. Die erste Zeile des Makros definiert den Makronamen. Der mittlere Teil, der die Befehle enthält, wird gewöhnlich Makrorumpf genannt. Die letzte Zeile beendet das Makro mit der Anweisung ENDM. Diese Anweisung am Ende der Definition dürfen Sie nicht vergessen. Fehlt sie, wird der Rest Ihres Programms fälschlicherweise als Teil eines sehr großen Makros angesehen. Die meisten Assembler werden dadurch völlig verwirrt und reagieren mit den absonderlichsten Fehlermeldungen.

Die Makrodefinitionen für die obigen Beispiele würden etwa so aussehen:

```
SAVE MACRO
     PUSH  H
     PUSH  D
     PUSH  B
     PUSH  PSW
     ENDM

UNSAVE MACRO
       POP  PSW
       POP  B
       POP  D
       POP  H
       ENDM
```

Makro-Parameter

Durch die Anwendung von Parametern werden Makros noch vielseitiger. Nehmen wir z. B. an, wir wollen unter Verwendung des Akkumulators als Arbeitsregister die Inhalte der Register H und L vertauschen. Ein Makro, das dies tut, könnte so aussehen:

```
INTER MACRO
      PUSH   PSW
      MOV    A,H
      MOV    H,L
      MOV    L,A
      POP    PSW
      ENDM
```

Immer, wenn nun der Makroname INTER im Programm erscheint, würde der Assembler folgende fünf Befehle einsetzen:

```
PUSH    PSW
MOV     A,H
MOV     H,L
MOV     L,A
POP     PSW
```

Beachten Sie, daß dieses Makro immer Befehle zum Austausch der Register H und L generiert. Ändern wir jedoch das Makro, indem wir zwei Parameter einfügen, wird es vielseitiger. Das folgende Makro ähnelt INTER bis auf die formalen Parameter REG1? und REG2? in der ersten Zeile. (Die Fragezeichen in den Parameternamen werden als normale Zeichen angesehen.)

```
INTER2 MACRO   REG1?,REG2?
       PUSH    PSW
       MOV     A,REG1?
       MOV     REG1?,REG2?
       MOV     REG2?,A
       POP     PSW
       ENDM
```

Für die formalen Parameter setzt der Assembler die aktuellen Werte ein. Die Anweisung

```
INTER2 H,L
```

wird in dieselben fünf Befehle umgesetzt wie in dem obigen Makro. Die Anweisung

```
INTER2 D,E
```

jedoch wird die folgenden Befehle generieren:

```
PUSH    PSW
MOV     A,D
MOV     D,E
MOV     E,A
POP     PSW
```

Makros und bedingtes Assemblieren

Anweisungen für bedingtes Assemblieren erhöhen weiterhin die Wirksamkeit von Makros. Die beiden folgenden Anweisungen können z. B. benutzt werden, um die Anwesenheit des wahlfreien Parameters PARAM? zu testen:

```
IF   NUL PARAM?
...
ENDIF
```

Der Ausdruck NUL PARAM? ist wahr, wenn der Parameter nicht angegeben wurde; sonst ist er falsch. Man kann auch den gegenteiligen Ausdruck

```
IF   NOT NUL PARAM?
```

benutzen, um die Abfrage umzukehren; d. h. der Ausdruck ist wahr, wenn der Parameter angegeben wurde.

Der Microsoft-Assembler akzeptiert auch die alternativen Formen

```
IFDEF   PARAM?
...
ENDIF
```

und

```
IFNDEF   PARAM?
...
ENDIF
```

für IF NOT NUL und IF NUL. Die Ausdrücke IFDEF und IFNDEF bedeuten „wenn definiert" und „wenn nicht definiert".

In manchen Programmen wollen wir bei Beendigung einen RET-Befehl ausführen. Bei anderer Gelegenheit wollen wir jedoch zu einer bestimmten Adresse verzweigen. Betrachten Sie das folgende Fragment des Makros EXIT, das wir demnächst entwickeln werden:

```
EXIT MACRO   WHERE?
    ...
    ...
    IF   NUL WHERE?
    RET
```

```
        ELSE
        JMP   WHERE?
        ENDIF
        ...
        ENDM
```

Der Parameter WHERE? in diesem Beispiel ist wahlfrei. Angenommen, das Makro EXIT wird ohne Parameter aufgerufen:

```
EXIT
```

Ein einfacher RET-Befehl wird in diesem Fall generiert, da der Ausdruck IF NUL WHERE? wahr ist. Wird jedoch ein Parameter angegeben, dann wird ein Sprung zum Parameter generiert. Der Makroaufruf

```
EXIT BOOT
```

wird also den Befehl

```
JMP BOOT
```

erzeugen.

Bevor wir mit unserer Makrobibliothek beginnen, wollen wir noch einen Blick auf die Generierung von Z80-Befehlen unter Benutzung von Makros und eines 8080-Assemblers werfen.

DAS GENERIEREN VON Z80-BEFEHLEN MIT EINEM 8080-ASSEMBLER

Die Z80-CPU kann alle 8080-Befehle ausführen; daher wird auch gewöhnlich ein 8080-Assembler für die Erzeugung von Assemblerprogrammen für einen Z80-Computer benutzt. Der Digital Research-Makroassembler MAC benutzt die mnemonischen Befehle für den INTEL 8080. Der Microsoft-Assembler MACRO-80 kann sowohl die INTEL 8080-, als auch die Zilog Z80-Befehle übersetzen. In diesem Buch werden wir primär die 8080-Befehle verwenden. Daher sind beide Assembler anwendbar.

Es gibt jedoch einige mächtige Z80-Befehle, die bei der Assemblerprogrammierung nützlich sind. Ein 8080-Assembler kann diese Befehle mit Makros erzeugen. Der Digital Research-Assembler beinhaltet tatsächlich eine Gruppe von Makros für diesen Zweck.

Angenommen, wir wollen eine Zahl von einer anderen subtrahieren. Dies kann man dadurch machen, daß man das Zweierkomplement der ersten zur zweiten Zahl addiert. Es gibt einen Z80-Befehl, der dies tut; sein Name ist NEG.

Im Befehlssatz des 8080 ist ein solcher Befehl nicht enthalten, jedoch kann man die Operation mit zwei 8080-Befehlen durchführen. Das Zweierkomplement ist das um 1 erhöhte Einerkomplement. Da es einen 8080-Befehl für das Einerkomplement und einen für die Erhöhung gibt, können wir beide in einem Makro vereinen. Die Makrodefinition folgt:

```
NEG  MACRO          ;Zweierkomplement
     CMA            ;Einerkomplement
     INR    A
     ENDM
```

Immer, wenn das Zweierkomplement benötigt wird, setzt man das Makro

```
NEG
```

in das Programm. Der 8080-Assembler erzeugt hierfür die entsprechenden Befehle:

```
     CMA
     INR    A
```

Beachten Sie, daß der zweite Kommentar mit zwei Semikolons beginnt, statt mit normalerweise einem:

```
     CMA            ;;Einerkomplement
```

Dies hat eine besondere Bedeutung in Makrodefinitionen. Beginnt ein Kommentar in einer Makrodefinition mit einem einzelnen Semikolon, wird der Kommentar in jeder Generierung wiedergegeben. Beginnt der Kommentar jedoch mit einem doppelten Semikolon, wird er nicht wiedergegeben. Da die erste und die letzte Zeile eines Makros nicht wiedergegeben werden, darf ein Kommentar hier mit einem Semikolon beginnen.

Beachten Sie wieder, wie ein Makro durch Anwendung von Parametern vielseitiger wird. Angenommen, wir ändern die Definition des vorigen Makros, wie folgt:

```
NEG   MACRO   REG?          ;Zweierkomplement
;;
      IF      NOT NUL REG?
      PUSH    PSW           ;;A retten
      MOV     A,REG?        ;;Register laden
      ENDIF
      CMA                   ;;Einerkomplement
      INR     A
      IF      NOT NUL REG?
      MOV     REG?,A        ;;Wert zurückschreiben
      POP     PSW           ;;A restaurieren
      ENDIF
      ENDM
```

Der Aufruf NEG wird dieselben zwei Befehle generieren wie in der vorigen Version, da kein Parameter beim Aufruf angegeben wurde. Wird jedoch ein Parameter angegeben, ist das Ergebnis anders. Der Aufruf

```
NEG   C
```

z. B. hat den Parameter C. Diesmal wird der Assembler folgenden Code erzeugen:

```
PUSH  PSW
MOV   A,C
CMA
INR   A
MOV   C,A
POP   PSW
```

D. h. die einzelne Makroanweisung NEG C produziert sechs Befehle statt zwei. Während der Generierung wurde der formale Parameter REG? durch den aktuellen Parameter C ersetzt. Die bedingte Passage

```
IF    NOT NUL REG?
...
ENDIF
```

erzeugt nur Befehle, wenn beim Aufruf ein Parameter angegeben wird. Andernfalls wird der Abschnitt zwischen IF und ENDIF übergangen.

DER 8080/Z80-SCHALTER

Obwohl die Z80-Computer sehr populär sind, sind auch viele 8080- und 8085-Computer in Gebrauch. Es gibt auch eine kombinierte 8085/8088-CPU-Karte, die sowohl eine 8085- als auch eine 8088-CPU enthält. (Der 8085 kann alle 8080-Befehle, aber keine Z80-spezifischen Befehle ausführen.) Daher kann es nötig sein, einerseits 8080-Befehle, zu anderer Gelegenheit die wirksameren Z80-Befehle zu benutzen. Dies kann man leicht einrichten mit Makros und bedingter Assemblierung.

Man kann am Beginn des Programms einen Z80-Schalter definieren. Die Anweisung

```
Z80M  EQU     TRUE          ;Z80-Modus-Schalter
```

wird für die Anzeige benutzt, daß Z80-Code gewünscht wird. Sonst benutzt man die Anweisung

```
Z80M  EQU     FALSE         ;Z80-Modus-Schalter
```

(Die Symbole TRUE und FALSE müssen natürlich separat definiert werden.) Ein Makro kann dann abhängig von der Definition des Schalters Z80M entweder Z80- oder 8080-Code generieren.

Betrachten wir als Beispiel den unbedingten relativen Sprung. Manchmal will man ein Programm an einer anderen Stelle fortsetzen. Dafür braucht man einen „unbedingten" Sprungbefehl. Beim Z80 haben wir die Wahl zwischen einem „relativen" unbedingten Sprung zu einer Adresse nicht weit entfernt von der momentanen Position und einem „absoluten" unbedingten Sprung zu einer beliebigen Adresse. Der relative Sprung wird normalerweise vorgezogen, da der Befehl kürzer ist als der absolute Sprungbefehl. Die 8080-CPU hat aber keinen solchen relativen Sprungbefehl. Wir wollen also vielleicht einen relativen Sprung bei einem Z80, aber einen absoluten Sprung bei einem 8080.

Wir können ein Makro mit bedingter Assemblierung schreiben, das den Z80-verträglichen Befehl für die eine Anwendung, den 8080-verträglichen Befehl für die andere Anwendung generiert. Wir definieren ein relatives Sprungmakro, wie folgt:

```
JR    MACRO   ADDR?
      IF      Z80M
      DB      18H,ADDR?-$-1
      ELSE
      JMP     ADDR?
      ENDIF
      ENDM
```

Ist der Z80-Schalter an (TRUE), generiert der Makroaufruf

```
JR      DONE
```

die beiden Bytes für den gewünschten Z80-Code:

```
DB      18H,DONE-$-1
```

Andernfalls wird der 3 Byte lange 8080-Befehl

```
JMP     DONE
```

generiert.

Ein weiteres Beispiel ist der Z80-Befehl DJNZ. Dieser Befehl vermindert das B-Register um 1 und springt dann zum Operanden, falls das Zero-Flag gelöscht ist. (Zur Erinnerung: Ein Flag ist „gelöscht" oder „falsch", wenn es null ist, sonst ist es „gesetzt" oder „wahr".) Das Makro kann dann etwa so aussehen:

```
DJNZ MACRO  ADDR?
     IF     Z80M
     DB     10H,ADDR?-$-1
     ELSE
     DCR    B
     JNZ    ADDR?
     ENDIF
     ENDM
```

Für die Z80-Version ergibt der Aufruf

```
DJNZ    LOOP
```

die Generierung

```
DB      10H,LOOP-$-1
```

Im 8080-Modus jedoch werden die Zeilen

```
DCR     B
JNZ     LOOP
```

produziert.

Der resultierende generierte Code ist fest. Er wird bei jeder Ausführung immer dasselbe tun. Ganz anders ist es bei dem Pascal- oder BASIC-Ausdruck

```
IF A=B THEN ...
```

Bei dieser BASIC-Anweisung wird eine Gruppe von Befehlen ausgeführt, wenn der Ausdruck wahr ist. Eine andere Gruppe jedoch kann ausgeführt werden, wenn er falsch ist.

Bevor wir mit der Makrobibliothek beginnen, wollen wir kurz das Makrokonzept zusammenfassen. Ein Makroassembler analysiert das Quellprogramm, indem er es mehrfach liest. Jedes Lesen nennt man „Pass" (Durchgang). In einem Pass konvertiert der Teil des Assemblers, der die Makros verarbeitet, die Makroaufrufe in die gewünschten Befehle. Wir haben z. B. gesehen, daß das Makro NEG zu den folgenden Befehlen führt:

```
CMA
INR     A
```

Im nächsten Durchgang analysiert der Assembler die vom Makroprozessor erzeugten Befehle, als ob sie in das Original-Quellprogramm eingeführt wären. Der sich ergebende Code ist derselbe, ob Makros benutzt wurden oder nicht.

DAS EINRICHTEN DER MAKROBIBLIOTHEK

In diesem und den folgenden Kapiteln werden wir eine Diskettendatei mit nützlichen Makros erstellen. Diese „Makrobibliothek" wird in vielen Programmen, die wir entwickeln, benutzt. Wenn man Makros in jedes Programm kopiert, wird viel Platz verschwendet. Es ist daher viel bequemer, alle Makros in einer separaten Makrobibliothek zu speichern. Auf diese können wir dann in einem Programm zurückgreifen.

Ein anderer Vorteil einer Makrobibliothek ist, daß etwaige Programmänderungen vereinfacht werden. Angenommen, Sie müssen ein Makro, das in vielen Programmen benutzt wird, ändern. Wenn dieses Makro in viele Programme kopiert worden wäre, müßten Sie jedes dieser Programme ändern. Erscheint das Makro jedoch nur einmal in der Makrobibliothek, muß es nur hier geändert werden.

Wir wollen unsere Makrobibliothek nun mit einem Kopf und einigen nützlichen Symbolen beginnen.

Häufig benutzte Konstanten

Mehrere Werte werden wir in fast all unseren Programmen brauchen; z. B. Zeichen wie Wagenrücklauf, Zeilentransport und Leerstelle. Es ist bequemer, sich symbolisch auf diese Werte zu beziehen, als die entsprechenden Werte dezimal oder hexadezimal anzugeben. Man könnte zwar einen Satz von Symbolen am Beginn eines jeden Programms angeben, einfacher jedoch ist es, diese Definitionen in der Makrobibliothek zu speichern. Wen Sie den Digital Research-Assembler benutzen, erstellen Sie mit dem Editor eine Datei mit dem Namen

```
CPMMAC.LIB
```

Benutzen Sie den Microsoft-Assembler, dann nennen Sie die Datei

```
CPMMAC.MAC
```

Jedes neue Makro werden wir dann in diese Datei eintragen. Jedes Assemblerprogramm, das sich auf diese Datei bezieht, wird nahe dem Programmanfang die folgende Anweisung enthalten:

```
MACLIB CPMMAC
```

Die Anweisung MACLIB weist den Assembler an, erforderliche Makrodefinitionen in der Datei CPMMAC zu suchen.

Achten Sie darauf, daß der Digital Research-Assembler den Dateityp ASM für Assemblerquellprogramme und LIB für Makrobibliothek erfordert, während der Microsoft-Assembler für beides den Typ MAC erwartet.

Schreiben Sie die Information nach Abb. 4.1 in die Datei CPMMAC.LIB (oder CPMMAC.MAC, wenn Sie den Microsoft-Assembler benutzen). Die Datei beginnt mit einer Kurzbeschreibung in der ersten Zeile und dem Tagesdatum in der zweiten Zeile. Ändern Sie dieses Datum bei jeder Änderung dieser Datei. Die dritte Angabe in der Bibliothek ist ein Inhaltsverzeichnis der Makros in dieser Datei. Bis jetzt gibt es noch keine Makros; am Ende des Buches werden jedoch etwa 40 Makros in dieser Bibliothek enthalten sein. Daher sollten wir alles sorgfältig kommentieren. Danach stehen die symbolischen Konstanten.

Wir werden nun unser erstes Makro in die Bibliothek eintragen. Dieses Makro codiert die Versionsnummer in jedes Programm, das wir schreiben.

Ein Makro zur Codierung der Versionsnummer

In Abb. 3.11 haben wir nahe dem Programmanfang das Erstellungsdatum geschrieben, um eine neue von einer älteren Version unterscheiden zu können. Dieses Datum wird jedoch nicht in die binäre Form des Programms aufgenommen. Nach der Übersetzung eines Programms in die Binärform ist es schwer herauszufinden, wann es erstellt wurde. Gibt es zwei Programme mit gleichen Namen, dann ist es nicht mehr möglich, die neueste Fassung zu bestimmen. Daher werden wir von nun an in jedes geschriebene Programm eine Versionsnummer codieren. Diese Information werden wir in ASCII verschlüsseln, damit man sie leichter entziffern kann. Um es einfach zu machen, codieren wir das Datum um den Programmnamen. Wir können dann leicht das Programm und sein letztes Datum identifizieren. Um das zu erreichen, schreiben wir ein Inline-Makro namens VERSN.

Die Zeilen eines Computerprogramms werden normalerweise sequentiell ausgeführt, eine nach der anderen. Daher stellt man den Hauptteil eines Programms an den Anfang, die Unterprogramme dagegen an das Ende. Z. B.

```
MAIN:
        ...
        CALL    SUB1
        CALL    SUB2
        ...
SUB1:
        ...
        RET
SUB2:
        ...
        RET
```

Nach dieser Methode kann das Hauptprogramm mit seinen Unterprogrammaufrufen zuerst geschrieben werden. Die Unterprogramme folgen dann dem Hauptprogramm.

Eine andere Methode ist, die Unterprogramme in Reihe mit dem Hauptprogramm zu plazieren. In diesem Fall benötigen wir einen Sprungbefehl, um das Hindernis zu umgehen. Z B.:

```
MAIN:
        ...
        CALL    SUB1
        JMP     AROUND
```

```
SUB1:
        ...
        RET
AROUND:
        ...
        CALL    SUB2
        JMP     OVER
SUB2:
        ...
        RET
OVER:
```

Diese Methode erscheint zwar weniger strukturiert als die vorige, sie hat jedoch einen wichtigen Vorteil – das Unterprogramm ist da in das Hauptprogramm eingefügt worden (inline), wo es gebraucht wird. Darüber hinaus kann diese Methode leichter mit Makros implementiert werden. In diesem Buch bezeichnen wir diese Art von Makro als „Inline-Makro".

Unser erstes Makro, in Abb. 4.2 gezeigt, heißt VERSN (für Versionsnummer). Dieses Inline-Makro steht nahe dem Programmanfang. Der Aufruf

```
VERSN  '9.23.82.FIRST'
```

```
;;Makro Bibliothek für CP/M-System-Routinen
;;(Tagesdatum)
;;
;;Makros in dieser Bibliothek:
;;(Tragen Sie hier jeden Makronamen ein)
;
;
EOF     EQU   1AH     ;Dateiende
ESC     EQU   1BH     ;Escape
CR      EQU   13      ;Cursor zu Zeilenanfang
LF      EQU   10      ;Cursor nächste Zeile
TAB     EQU   9       ;Control-I
BLANK   EQU   32      ;Leerzeichen
PERIOD  EQU   46      ;Dezimalpunkt
COMMA   EQU   44
;;(hier fügen Sie die Makros ein)
```

Abb. 4.1: Der Anfang einer Makrobibliothek: häufig benutzte Symbole

```
VERSN       MACRO   NUM
;;(Tagesdatum)
;;Inline-Makro zum Einbetten der Versionsnummer.
;;NUM wird in Apostrophe eingeschlossen
;;
;;Anwendung: VERSN 'XX.XX.XX.NAME'
;;
            LOCAL   AROUND
            JMP     AROUND
            DB      'Ver ',NUM
AROUND:                         ;;VERSN
            ENDM
```

Abb. 4.2: Das Makro VERSN zur Codierung der Versionsnummer

erzeugt drei Zeilen:

```
            JMP     ??0001
            DB      'Ver','9.23.82.FIRST'
??0001:
```

Mit diesem Makro kann man Informationen, wie Datum und Programmname, direkt in den Binärcode einbetten. Die Datenanweisung „Ver 9.23.82.FIRST" ist in das Programm eingebettet, und mit einem Sprungbefehl wird der Ausdruck umgangen. Die Marke AROUND wird als lokale Variable im Makro deklariert und hat daher nur im Makro eine Bedeutung. Der Digital Research-Assembler ordnet das Symbol ??0001 der ersten lokalen Variablen (AROUND in diesem Beispiel) zu. Andere Makroassembler können andere Symbole benutzen.

Bei mehr als einmaligem Aufruf dieses Makros wird jedesmal eine andere Marke erzeugt. Das Wort AROUND erscheint nicht in der Assemblerliste. Die Marke AROUND kann daher anderswo im Programm oder als Variable in einem anderen Makro benutzt werden, ohne daß es zu einem Fehler wegen Doppeldefinition kommt. Das Symbol NUM ist ein formaler Parameter. Auch er kann ohne Schwierigkeiten außerhalb des Makros benutzt werden.

Die zweite durch das Makro VERSN generierte Zeile definiert die in das Programm einzubettenden Daten. (Die Assembleranweisung DB steht für „define byte" – definiere Byte.) Der Operand besteht in diesem Bei-

```
TITLE   'TESTVER zum Testen des Makros VERSN'
;
;Mar. 3, 82
;
BOOT    EQU     0           ;Warmstart
TPA     EQU     100H        ;Beginn des Anwenderbereichs
;
        MACLIB  CPMMAC
ORG     TPA                 ;nicht bei der Microsoft-Version
;
START:
        VERSN   '3.3.82.FIRST'
        JMP     BOOT
;
        END     START
```

Abb. 4.3: Ein Programm, um das Makro VERSN zu testen

spiel aus einer in Apostrophe eingeschlossenen alphanumerischen Zeichenfolge. Hier können auch Bytewerte angegeben werden. Die dritte durch das Makro VERSN generierte Zeile ist die Marke ??0001, das Ziel des Sprungbefehls.

Erstellen Sie nun eine weitere Datei namens TESTVERS.ASM. Damit wollen wir unser erstes Makro testen. Geben Sie die Daten nach Abb. 4.3 ein. Dieses Programm bezieht sich auf unsere Makrobibliothek. Tragen Sie das Tagesdatum am Beginn des Programms und als Parameter des Makros VERSN ein.

Für den Microsoft-Assembler müssen Sie noch einige Änderungen vornehmen. Erstens entfernen Sie die Apostrophe in der TITLE-Anweisung. Stellen Sie zweitens sicher, daß Sie MACLIB mit Großbuchstaben geschrieben haben. Drittens entfernen Sie die ORG-Anweisung. Fügen Sie viertens die Anweisung „.XLIST“ vor und die Anweisung „.LIST“ hinter der MACLIB-Zeile ein. Hierdurch erreichen Sie, daß der Microsoft-Assembler die Makrobibliothek nicht druckt.

Assemblieren Sie das Programm, und vergleichen Sie die Assemblerliste mit der in Abb. 4.4 (Assembler betrachten Klein- und Großbuchstaben als gleich. Der Digital Research-Assembler wandelt kleine Buchstaben jedoch in große um.) Der erste Befehl in der Assemblerliste folgt der

Marke START. Es ist ein Sprung über die Codierung von Programmname und Datum. Beachten Sie, daß die ersten beiden Zeilen ein Pluszeichen zwischen der Adresse und dem Code haben. Dadurch kennzeichnet der Digital Research-Assembler durch Makros generierte Zeilen.

Laden Sie die übersetzte Datei in den Speicher, und untersuchen Sie sie mit dem Debugger. Für die Digital Research Version tun Sie dies mit dem Kommando

```
SID TESTVER.HEX
```

Zeigen Sie den ersten Teil des Speichers mit dem D-Kommando an:

```
D100,11F
```

Die Anzeige wird sein:

```
0100: C3 13 01 56 65 72 20 33 2E 33 2E 38 32 2E 46 49 ...Ver 3.3.82.FI
0110: 52 53 54 C3 00 00 00 00 00 00 00 00 00 00 00 00 RST.............
```

Die Anzeige besteht aus drei Teilen. Die erste Zahl einer Zeile ist die Adresse (100 hex in der ersten Zeile hier). Der zweite Teil gibt die Inhalte

```
                TITLE   'TESTVER zum Test des Makros VERSN'
                ;
                ;Mar. 3, 82
                ;
0000 =          BOOT    EQU     0       ;Warmstart
0100 =          TPA     EQU     100H    ;Beginn des Anwenderbereichs
                ;
                        MACLIB  CPMMAC
0100            ORG     TPA             ;nicht bei der Microsoft-Version
                ;
                START:
                        VERSN   '3.3.82.FIRST'
0100+C31301             JMP     ??0001
0103+5665722033         DB      'Ver','3.3.82.FIRST'
0113 C30000             JMP     BOOT
                ;
0116                    END     START
```

Abb. 4.4: Das Assemblerlisting zu Abb. 4.3

von 16 Bytes des Speichers hexadezimal aus. Der dritte Teil zeigt die ASCII-Zeichen dieser 16 Bytes. Nicht druckbare Zeichen erscheinen dabei als Punkte. Sie können das Programm mit G100 starten. Dieses einfache Programm tut jedoch überhaupt nichts; wir haben es nur geschrieben, um die Übersetzung zu sehen.

Makros zum Retten und Wiederherstellen des übergebenen Stacks

Wenn ein Programm mit dem Betriebssystem CP/M ausgeführt wird, wird es von der Diskette in den Anwenderprogrammbereich (TPA) ab Adresse 100 hex geladen. CP/M springt nach 100 hex. Bei Beendigung des Programms gibt es zwei verschiedene Wege, um zu CP/M zurückzugelangen. Der einfachste Weg ist es, mit einem Sprung nach Adresse 0 einen Warmstart auszuführen, wie wir es in Abb. 4.3 getan haben.

Eine andere Methode, zu CP/M zurückzugelangen, ist, den übergebenen Stackpointer zu retten und für Programmzwecke einen neuen Stack anzulegen. Bei Programmende wird der alte Stackpointer wiederhergestellt und ein RET-Befehl ausgeführt. Diese Methode zur Beendigung ist schneller und daher der ersten vorzuziehen, da CCP und BDOS nicht erneut von der Diskette geladen werden. Wir werden in den meisten Programmen dieses Buches diesen Weg gehen.

Manchmal ist ein Programm jedoch so groß, daß der CCP überschrieben wird. In diesem Fall muß ein Programm sich mit Warmstart beenden. CCP und BDOS werden dann erneut von der Systemdiskette geladen.

Mit einer Z80-CPU ist es leicht, den Stackpointer zu retten und wiederherzustellen. Die Z80-Befehle sind:

```
START:
        LD      (OLDSTK),SP
        LD      SP,STACK
        ...
DONE:
        LD      SP,(OLDSTK)
        RET
```

Die beiden ersten Befehle stehen am Beginn des Programms. Der Stackpointer wird in den Speicher OLDSTK gerettet. Der neue Stack wird bei der Adresse STACK eingerichtet. Die beiden anderen Befehle stehen am (logischen) Ende des Programms. Der erste Befehl stellt den Original-Stackpointer wieder her. Der letzte Befehl bewirkt die Rückkehr zu CP/M.

Die 8080-CPU hat keine Befehle zum direkten Retten und Wiederherstellen des Stackpointers. Daher ist die 8080-Version etwas komplizierter. Gewöhnlich lädt man den übergebenen Stackpointer in das HL-Register und schreibt es in den Speicher. Die Befehle hierfür sind:

```
START:
        LXI     H,0             ;löschen
        DAD     SP              ;addieren Pointer
        SHLD    OLDSTK          ;retten
        LXI     SP,STACK        ;neuer Stack
```

Bei Beendigung des Programms wird der Original-Stackpointer aus dem Speicher in das HL-Register geladen und dann in den Stackpointer gebracht. Der RET-Befehl wird ausgeführt. Das sieht so aus:

```
DONE:
        LHLD    OLDSTK          ;Original-Stackpointer
        SPHL
        RET
```

Sowohl beim Z80 als auch beim 8080 müssen wir Platz für die Speicherung des Original-Stackpointers und für den neuen Stack bereitstellen. Die folgenden Zeilen fügen wir dafür ein:

```
OLDSTK: DS      2               ;übergebener Stack
        DS      34
STACK:
```

In den meisten Programmen dieses Buches werden wir den übergebenen Stackpointer retten und wiederherstellen. Das werden wir zweckmäßigerweise mit zwei Makros durchführen. Das Makro am Beginn des Programms nennen wir ENTER, das am Ende nennen wir EXIT. (Wir müssen darauf achten, daß die von uns gewählten Namen keine für den Assembler reservierten Namen sind, wie z. B. END.)

Fügen Sie die beiden Makros nach Abb. 4.5 in die Makrobibliothek (CPMMAC) ein. Wenn Sie sie vor VERSN einfügen, sind die drei Makros in alphabetischer Reihenfolge. Vergessen Sie nicht, die Namen ENTER und EXIT in das Verzeichnis am Beginn der Bibliothek aufzunehmen.

Den ENTER-Makroaufruf wird man direkt hinter die Marke START schreiben; den EXIT-Markoaufruf an das (logische) Ende des Programms. Beachten Sie, daß das Makro ENTER keine, das Makro EXIT

hingegen zwei Parameter hat. Innerhalb des Makros EXIT gibt es auch zwei bedingte Blöcke. Dadurch kann man verschiedene Befehlsgruppen mit demselben Makro generieren.

```
ENTER     MACRO
;;(Tagesdatum)
;;Inline-Makro zum Retten des übergebenen Stack-Pointers
;;
          LXI       H,0             ;löschen
          DAD       SP              ;addiere Pointer
          SHLD      OLDSTK          ;retten
          LXI       SP,STACK
                                    ;;ENTER
          ENDM
;
EXIT      MACRO     WHERE?,SPACE?
;;
;;Inline-Makro zum Retten des übergebenen Stack-Pointers
;;und Sprung zu Adresse WHERE?
;;Fehlt WHERE?, wird ein RET-Befehl ausgeführt.
;;SPACE? definiert den Stackbereich; Standard ist 34.
;;
          LHLD      OLDSTK
          SPHL
          IF        NUL  WHERE?
          RET
          ELSE
          JMP       WHERE?
          ENDIF
;
OLDSTK:   DS        2               ;übergebener Stack-Pointer
          IF        NUL  SPACE?
          DS        34
          ELSE
          DS        SPACE?
          ENDIF
STACK:    EQU       $               ;nicht bei Microsoft
                                    ;;EXIT
          ENDM
```

Abb. 4.5: Die Makros ENTER und EXIT

Wenn beim Aufruf des Makros EXIT keine Parameter angegeben werden, sind die beiden Parameter WHERE? und SPACE? leer (NUL). Die bedingten Ausdrücke

```
        IF      NUL WHERE?
```

und

```
        IF      NUL SPACE?
```

sind wahr, und der erste Teil des bedingten Blocks wird bis ELSE assembliert; die Befehle zwischen ELSE und ENDIF dagegen nicht. Im resultierenden Code wird der Stackpointer wiederhergestellt, ein RET-Befehl ausgeführt und ein Stack von 34 Bytes eingerichtet.

Beachten Sie, daß der Stack am Ende des Programms liegt. Es mag logischer erscheinen, ihn an den Anfang zu verlegen. Dadurch wird jedoch das Objektprogramm viel größer, da der Stackbereich mit dem Programm gespeichert werden müßte.

Erstellen Sie eine Kopie des Programms in Abb. 4.3, und ändern Sie es ab, wie in Abb. 4.6 dargestellt. Für den Microsoft-Assembler machen Sie

```
TITLE   'TESENT zum Testen der Makros ENTER und EXIT'
;
;Mar. 3, 82
;
BOOT    EQU     0       ;Warmstart
TPA     EQU     100H    ;Anwenderprogrammbereich
;
;Digital Research-Version
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   '3.3.82.SECOND'
        EXIT
;
        END     START
```

Abb. 4.6: Ein Programm, um die Makros ENTER und EXIT zu testen

dieselben Änderungen wie für die Version in Abb. 4.3. Außerdem müssen Sie noch den Teil „EQU $“ hinter der Marke STACK entfernen. Für den Microsoft-Assembler sieht das Ende des EXIT-Makros dann so aus:

```
STACK:
                                        ;;EXIT
        ENDM
```

```
                TITLE   'TESENT zum Testen der Makros ENTER und EXIT'
                ;
                ;Mar. 3, 82
                ;
0000 =          BOOT    EQU     0       ;Warmstart
0100 =          TPA     EQU     100H    ;Anwenderprogramm-
                                        bereich
                ;
                ;Digital Research-Version
                        MACLIB  CPMMAC
                ;
0100            ORG     TPA
                ;
                START:
                        ENTER
0100+210000             LXI     H,0     ;löschen
0103+39                 DAD     SP      ;addiere Pointer
0104+222301             SHLD    OLDSTK  ;retten
0107+314701             LXI     SP,STACK
                        VERSN   '3.3.82.SECOND'
010A+C31E01             JMP     ??0001
010D+5665722033 DB              'Ver ','3.3.82.SECOND'
                        EXIT
011E+2A2301             LHLD    OLDSTK
0121+F9                 SPHL
0122+C9                 RET
0123+           OLDSTK: DS      2       ;übergebener
                                        Stack-Pointer
0125+                   DS      34
0147+=          STACK:  EQU     $       ;nicht bei Microsoft
                ;
0147                    END     START
```

Abb. 4.7: Das Assemblerlisting zu Abb. 4.6

```
                        EXIT    BOOT,20
011E+2A2501             LHLD    OLDSTK
0121+F9                 SPHL
0122+C30000             JMP     BOOT
0125+           OLDSTK: DS      2       ;übergebener
                                        Stack-Pointer
0127+                   DS      20
013B+=          STACK:
                        EQU     $
```

Abb. 4.8: Die Verwendung von Parametern im Makro EXIT

Beachten Sie, daß der Aufruf des EXIT-Makros keine Parameter hat. Assemblieren Sie das Programm, und vergleichen Sie die Assemblerliste mit Abb. 4.7. Das Programm kann man ausführen, es wird allerdings nichts tun.

Benutzung von Parametern im Makro EXIT

Im vorigen Programm hatte der Aufruf des Makros EXIT keine Parameter. Doch stellen Sie sich das Programmfragment aus Abb. 4.8 vor. In diesem Fall hat der EXIT-Aufruf zwei Parameter:

```
EXIT    BOOT,20
```

Bei der Übersetzung wird dem formalen Parameter WHERE? das Label BOOT zugeordnet, dem formalen Parameter SPACE? der Wert 20. Daher sind die Ausdrücke

```
IF      NUL WHERE?
```

und

```
IF      NUL SPACE?
```

falsch, wodurch der assemblierte Code einen Sprung nach BOOT sowie einen 20 Byte langen Stack erhält.

Natürlich sind auch andere Kombinationen möglich. Die Anweisung

```
EXIT    ,20
```

hat nur den zweiten Parameter. Das Komma vor der 20 zeigt an, daß der erste Parameter ausgelassen wude und daher NUL ist. Das Makro wird einen RET-Befehl und einen 20-Byte-Stack bewirken.

EIN MAKRO ZUM UMSPEICHERN VON INFORMATIONEN

Von Zeit zu Zeit ist es nötig, Daten von einer Stelle zu einer anderen umzuspeichern. Dies nennt man „Block Move" (Blockverschiebung). Dafür werden wir nun ein Makro schreiben. Sowohl die 8080- als auch die Z80-CPU haben Register, die beim Umspeichern als Pointer dienen können. Darüber hinaus hat der Z80 einen Befehl, der direkt einen Block umspeichern kann. Man kann also die Blockverschiebung sehr vereinfachen, wenn das Programm für einen Z80 geschrieben wird. Hier wollen wir jedoch den 8080 betrachten.

Fügen Sie das MOVE-Makro nach Abb. 4.9 zu Ihrer Makrobibliothek hinzu. Fügen Sie es in der alphabetischen Reihenfolge zwischen den Makros EXIT und VERSN ein. Vergessen Sie nicht, den Namen in das Inhaltsverzeichnis einzutragen. Dieses Verzeichnis sollte nun die folgenden Makros anzeigen:

```
ENTER
EXIT
MOVE
VERSN
```

Die Organisation des Makros MOVE ist typisch für viele der Makros, die wir im Verlauf dieses Buches schreiben werden. Es gibt eine Initialisierung, einen Unterprogrammaufruf, einen Sprung um das Unterprogramm herum und das Unterprogramm selbst.

Lassen Sie uns die Einzelheiten des MOVE-Makros untersuchen. Es hat drei formale Parameter: FROM, TO und BYTES. Entsprechend den Namen ist FROM die Adresse des Sendefeldes, TO die des Empfangsfeldes und BYTES die Anzahl der zu speichernden Bytes. Zu Beginn werden mit PUSH-Befehlen die CPU-Register gerettet. Das HL-Register wird mit der Sende-, das DE-Register mit der Empfangsadresse und das BC-Register mit der Anzahl geladen. (Das X im Befehl deutet auf Doppelregister. Der Operand H bedeutet HL, usw.)

Der Hauptteil des Makros ruft das Unterprogramm MOVE2? für die eigentliche Umspeicherung auf. Der Befehl MOV A,M lädt ein Byte des Sendefeldes in den Akkumulator. Der Befehl STAX D speichert es dann in das Empfangsfeld. Die Pointer HL und DE werden dann um 1 erhöht, und die Anzahl im BC-Register wird um 1 vermindert. Dies wird wiederholt, bis die Zahl im BC-Register 0 wird.

```
MOVE        MACRO   FROM,TO,BYTES
;;(Tagesdatum)
;;Inline-Makro zum Umspeichern von Text
;;
            LOCAL   AROUND
            PUSH    H
            PUSH    D
            PUSH    B
            LXI     H,FROM
            LXI     D,TO
            LXI     B,BYTES
            CALL    MOVE2?
            POP     B
            POP     D
            POP     H
            JMP     AROUND
;
MOVE2?:
            MOV     A,M         ;lesen
            STAX    D           ;speichern
            INX     H           ;von
            INX     D           ;nach
            DCX     B           ;Zeichenzahl
            MOV     A,C
            ORA     B
            JNZ     MOVE2?      ;weiter
            RET
AROUND:                         ;;MOVE
ENDM
```

Abb. 4.9: Das Makro MOVE, Version 1

Ein Doppelregister auf 0 zu prüfen ist schwieriger als ein einfaches Register, da die CPU-Flags durch die Doppelregistererhöhung bzw. -verminderung nicht verändert werden. Daher geht dies nicht mit den Befehlen

```
DCX     B
JNZ     MOVE2?
```

Das Makro macht diesen Test dadurch, daß eine Hälfte des Registers in den Akkumulator geladen und durch OR mit der anderen Hälfte verknüpft wird. Am Ende geht es beim Hauptteil des Makros weiter.

Bei der ersten Expandierung des Makros MOVE wird die Subroutine MOVE2? generiert, unmittelbar hinter dem Hauptteil des Makros. Es gibt also einen Sprungbefehl, der diese Subroutine überspringt. Dafür wird die Marke AROUND benutzt. Der Unterprogrammname MOVE2? ist übrigens nicht als lokale Variable deklariert; es ist also eine globale Variable und kann auch von anderen Programmteilen aufgerufen werden.

```
TITLE   'TESTMOVE zum Testen des Makros MOVE'
;
;Dec. 16, 81
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
BOOT    EQU     0           ;Warmstart
BDOS    EQU     5           ;BDOS-Eingang
TPA     EQU     100H        ;Anwenderbereich
;
MVFLAG  SET     FALSE
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        VERSN   '12.16.81.TESTMOVE.1'
        MOVE    TEXT, NEWTEX, TEXEND-TEXT
        JMP     BOOT
;
        DB      '<===='
TEXT:
        DB      'A test of macro MOVE'
TEXEND:
;
ORG     400H
;
NEWTEX: DS      1
;
END     START
```

Abb. 4.10: Ein Programm zum Testen der Version 1 des Makros MOVE

Erstellen Sie eine Datei namens MOVE1.ASM, und tragen Sie das Programm nach Abb. 4.10 ein. Mit diesem Programm werden wir das Makro MOVE testen.

Unser Testprogramm beginnt mit dem Makro VERSN, gefolgt von dem Makro MOVE. Es schließt ab mit einem Sprung nach BOOT und einem Pfeil, der auf diesen Befehl zeigt. Der Ausgangstext beginnt bei TEXT und endet bei TEXEND. Das Empfangsfeld beginnt bei NEWTEX.

```
                    TITLE   'TESTMOVE zum Testen des Makros MOVE'
                    ;
                    ;Dec. 16, 81
                    ;
0000 =      FALSE   EQU     0
FFFF =      TRUE    EQU     NOT  FALSE
            ;
0000 =      BOOT    EQU     0           ;Warmstart
0005 =      BDOS    EQU     5           ;BDOS-Eingang
0100 =      TPA     EQU     100H        ;Anwenderbereich
            ;
                    MACLIB  CPMMAC
            ;
0100        ORG     TPA
            ;
            START:
                    VERSN   '12.16.81.TESTMOVE.1'
0100+C31A01         JMP     ??0001
0103+5665722031     DB      'Ver','12.16.81.TESTMOVE.1'
                    MOVE    TEXT, NEXTEX, TEXEND-TEXT
011A+E5             PUSH    H
011B+D5             PUSH    D
011C+C5             PUSH    B
011D+214201         LXI     H,TEXT
0120+110004         LXI     D,NEWTEX
0123+011400         LXI     B,TEXEND-TEXT
0126+CD2F01         CALL    MOVE2?
0129+C1             POP     B
012A+D1             POP     D
012B+E1             POP     H
012C+C33A01         JMP     ??0002
```

Abb. 4.11: Die Assemblerliste zu Abbildung 4.10

```
012F+7E                 MOV     A,M         ;lesen
0130+12                 STAX    D           ;speichern
0131+23                 INX     H           ;von
0132+13                 INX     D           ;nach
0133+0B                 DCX     B           ;Zeichenzahl
0134+79                 MOV     A,C
0135+B0                 ORA     B
0136+C22F01             JNZ     MOVE2?      ;weiter
0139+C9                 RET
013A                    C30000  JMP         BOOT
                ;
013D 3C3D3D3D3D         DB      '<===='
                TEXT:
0142 4120746573         DB      'A test of Makro MOVE'
                TEXEND:
                ;
0400 ORG                400H
                ;
0400            NEWTEX: DS      1
                ;
0401                    END     START
```

Abb. 4.11: Die Assemblerliste zu Abbildung 4.10 (Forts.)

Assemblieren Sie das Programm, und vergleichen Sie die Assemblerliste mit Abb. 4.11. Notieren Sie die Adresse 13A hex des letzten Sprungbefehls. (Die Adresse kann abweichen, wenn Sie das Datum anders codiert haben.) Wir brauchen diese Adresse für den nächsten Schritt. Laden Sie die Datei mit dem Debuggerkommando

```
SID MOVE1.HEX
```

Lassen Sie sich den Anfang des Speichers mit D100,15F anzeigen. Das Ergebnis ist dann etwa:

```
0100: C3 1A 01 56 65 72 20 31 32 2E 31 36 2E 38 31 2E ...Ver 12.16.81.
0110: 54 45 53 54 4D 4F 56 45 2E 31 E5 D5 C5 21 42 01 TESTMOVE.1...!B.
0120: 11 00 04 01 14 00 CD 2F 01 C1 D1 E1 C3 3A 01 7E ......./.....:.~
0130: 12 23 13 0B 79 B0 C2 2F 01 C9 C3 00 00 3C 3D 3D .#..y../.....<==
0140: 3D 3D 41 20 74 65 73 74 20 6F 66 20 6D 61 63 72 ==A test of macr
0150: 6F 20 4D 4F 56 45 00 00 00 00 00 00 00 00 00 00 o MOVE..........
```

Der Text, der aus dem Makro VERSN entstand (nahe dem Anfang), ist in der ASCII-Ausgabe gut lesbar. In der vierten Zeile zeigt der Pfeil auf den beendenden Sprungbefehl bei Adresse 13A hex. Starten Sie das Programm mit G100,13A. Dadurch wird die Ausführung bei Adresse 100 hex begonnen und bei Adresse 13A hex gestoppt. Der Debugger setzt einen Stop bei Adresse 13A. Er ändert hier den Sprungbefehl in einen RST 7. Der Debugger antwortet mit

```
*013A
```

und zeigt damit die Unterbrechung des Programms an.

Geben Sie das Debuggerkommando D400,41F, um das Empfangsfeld anzuzeigen. Das sieht dann so aus:

```
0400: 41 20 74 65 73 74 20 6F 66 20 6D 61 63 72 6F 20 A test of macro
0410: 4D 4F 56 45 00 00 00 00 00 00 00 00 00 00 00 00 MOVE............
```

Das Programm hat den Text „A test of macro MOVE" in das Empfangsfeld übertragen.

Wenn Sie den Test wiederholen wollen, löschen Sie vorher das Zielfeld mit dem Debuggerkommando

```
F400,41F,0
```

Dann starten Sie erneut mit G100,13A.

Das Makro MOVE, Version 2

Bei unserer zweiten Version des MOVE-Makros werden wir eine neue Technik anwenden, die auf weitere Makros in diesem Buch anwendbar sein wird. Wie wir oben sahen, besteht unser Makro aus vier Teilen – einer Initialisierung, einem Unterprogrammaufruf, einem Sprung über das Unterprogramm und dem Unterprogramm selbst. So sieht es bei der ersten Expandierung aus. Bei nachfolgenden Aufrufen werden nur noch die beiden ersten Teile benötigt. Weitere Aufrufe beziehen sich auf das Unterprogramm, das beim ersten Mal generiert wurde.

Wir werden ein besonderes Symbol benutzen, um zu entscheiden, ob das Makro schon einmal aufgerufen wurde oder nicht. Hierfür gibt es einige Gründe. Beim ersten Aufruf des Makros wird das Unterprogramm MOVE2? generiert. Beim zweiten Mal wird es noch einmal generiert. D. h. bei jedem Makroaufruf wird eine Kopie der Subroutine generiert. Das erzeugt unnötig viel Code. Außerdem würde der Assembler einen

Phase-Fehler melden, weil dem globalen Symbol MOVE2? ein neuer Wert zugewiesen wird.

Wir brauchen eine Methode, mit der wir beim ersten Aufruf des MOVE-Makros die Subroutine MOVE2? generieren, nicht jedoch bei nachfolgenden Aufrufen. Es gibt hierzu mehrere Wege, aber wir wählen einen, der bei allen Assemblern angewendet werden kann.

```
MOVE        MACRO   FROM,TO,BYTES
;;(Tagesdatum)
;;Inline-Makro zum Umspeichern von Text
;;
            LOCAL   AROUND
            PUSH    H
            PUSH    D
            PUSH    B
            LXI     H,FROM
            LXI     D,TO
            LXI     B,BYTES
            CALL    MOVE2?
            POP     B
            POP     D
            POP     H
            IF      NOT  MVFLAG
            JMP     AROUND
;
MOVE2?:
            MOV     A,M             ;lesen
            STAX    D               ;speichern
            INX     H               ;von
            INX     D               ;nach
            DCX     B               ;Zeichenzahl
            MOV     A,C
            ORA     B
            JNZ     MOVE2?          ;weiter
            RET
MVFLAG      SET     TRUE            ;;eine Kopie
            ENDIF                   ;;not MVFLAG
AROUND:                             ;;MOVE
            ENDM
```

Abb. 4.12: Das Makro MOVE, Version 2

Wir definieren das Symbol MVFLAG, das anzeigt, ob das Unterprogramm bereits generiert wurde. Das Symbol kann die Werte wahr oder falsch haben. So ein Symbol nennt man Flag. Dieses Flag wird initialisiert mit der Anweisung

```
MVFLAG SET     FALSE
```

Das Flag wird mit einer SET-Anweisung definiert, nicht mit EQU, damit man seinen Wert während einer Assemblierung ändern kann. (EQU-Symbole kann man nicht ändern.) Der beste Platz für diese Anweisung wäre der Anfang der Makrobibliothek, aber der Digital Research-Assembler erlaubt dies nicht. Daher stellen wir diese Anweisung an den Beginn des Programms, das das Makro benutzt.

Ändern Sie das Move-Makro in die in Abb. 4.12 gezeigte Form ab. Unmittelbar vor dem JMP AROUND sehen Sie einen bedingten Ausdruck, mit dem der Zustand des Flags MVFLAG getestet wird. Beim ersten Aufruf ist der Ausdruck NOT MVFLAG wahr, und die Anweisungen bis zur ENDIF-Anweisung werden assembliert. Diese Befehle stellen das Unterprogramm MOVE2? dar. Da ist auch noch eine wichtige Anweisung vor dem ENDIF. Mit ihr wird der Zustand des Flags geändert:

```
MVFLAG SET TRUE
```

Wenn das Makro MOVE das nächste Mal im selben Programm aufgerufen wird, ist das Flag wahr, und der Ausdruck NOT MVFLAG ist falsch. Daher wird der Assembler das Unterprogramm MOVE2? nicht ein zweites Mal übersetzen, und der Übersprung ist auch nicht mehr nötig.

Machen Sie eine Kopie des Programms aus Abb. 4.10, und ändern Sie es nach Abb. 4.13 ab. Geben Sie der neuen Version den Namen MOVE 2.ASM.

```
TITLE   'TESTMOVE zum Testen des Makros MOVE'
;
;Dec.16, 81
;
FALSE   EQU     0
TRUE    EQU     NOT     FALSE
;
```

Abb. 4.13: Ein Programm zum Testen der Version 2 des Makros MOVE

```
BOOT    EQU     0           ;Warmstart
BDOS    EQU     5           ;BDOS-Eingang
TPA     EQU     100H        ;Anwenderbereich
;
MVFLAG  SET     FALSE       ;Blockverschiebung
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        VERSN   '12.16.81.TESTMOVE.2'
        MOVE    TEXT, NEWTEX, TEXT2-TEXT
        MOVE    TEXT2, NEWTEX+TEXT2-TEXT, TEXEND-TEXT2
        JMP     BOOT
;
        DB      '<====='
TEXT:
        DB      'A test of Makro MOVE'
TEXT2:
        DB      '. A second MOVE.'
TEXEND:
;
ORG     400H
;
NEWTEX: DS      1
;
        END     START
```

Abb. 4.13: Ein Programm zum Testen der Version 2 des Makros MOVE (Forts.)

Assemblieren Sie das neue Testprogramm, und vergleichen Sie den letzten Teil der Liste mit Abb. 4.14. Sie sehen, daß der erste Aufruf des Makros MOVE bei Adresse 11A hex die Subroutine MOVE2? generiert. Aus dem JMP AROUND wird (beim Digital Research-Assembler) JMP ??0002, da es eine lokale Variable ist. Der zweite Aufruf des Makros MOVE bei Adresse 13A hex generiert die Subroutine nicht noch einmal, sondern einen Aufruf der beim ersten Mal erzeugten Subroutine.

Laden Sie das Programm mit dem Debuggerkommando

```
SID MOVE2.HEX
```

```
                MOVE    TEXT,NEWTEX,TEXT2-TEXT
011A+E5         PUSH    H
011B+D5         PUSH    D
011C+C5         PUSH    B
011D+215401     LXI     H,TEXT
0120+110004     LXI     D,NEWTEX
0123+011400     LXI     B,TEXT2-TEXT
0126+CD2F01     CALL    MOVE2?
0129+C1         POP     B
012A+D1         POP     D
012B+E1         POP     H
012C+C33A01     JMP     ??0002
012F+7E         MOV     A,M         ;lesen
0130+12         STAX    D           ;speichern
0131+23         INX     H           ;von
0132+13         INX     D           ;nach
0133+0B         DCX     B           ;Zeichenzahl
0134+79         MOV     A,C
0135+B0         ORA     B
0136+C22F01     JNZ     MOVE2?      ;weiter
0139+C9         RET
                MOVE    TEXT2,NEWTEX+TEXT2-TEXT,
                        TEXEND-TEXT2
013A+E5         PUSH    H
013B+D5         PUSH    D
013C+C5         PUSH    B
013D+216801     LXI     H,TEXT2
0140+111404     LXI     D,NEWTEX+TEXT2-TEXT
0143+011000     LXI     B,TEXEND-TEXT2
0146+CD2F01     CALL    MOVE2?
0149+C1         POP     B
014A+D1         POP     D
014B+E1         POP     H
014C C30000     JMP     BOOT
                ;
014F 3C3D3D3D3D DB      '<===='
            TEXT:
0154 4120746573 DB      'A test of Makro MOVE'
            TEXT2:
```

Abb. 4.14: Teil-Assemblerliste zu Abb. 4.13

```
0168 2E20412073     DB          '.A second MOVE.'
             TEXEND:
             ;
0400         ORG     400H
             ;
0400         NEWTEX: DS          1
             ;
0401                 END         START
```

Abb. 4.14: Teil-Asemblerliste zu Abb. 4.13 (Forts.)

Zeigen Sie das Programm an mit D100,17F. Sie erhalten:

```
0100: C3 1A 01 56 65 72 20 31 32 2E 31 36 2E 38 31 2E ...Ver 12.16.81.
0110: 54 45 53 54 4D 4F 56 45 2E 32 E5 D5 C5 21 54 01 TESTMOVE.2...!T.
0120: 11 00 04 01 14 00 CD 2F 01 C1 D1 E1 C3 3A 01 7E ......./.....:.~
0130: 12 23 13 0B 79 B0 C2 2F 01 C9 E5 D5 C5 21 68 01 .#..y../.....!h.
0140: 11 14 04 01 10 00 CD 2F 01 C1 D1 E1 C3 00 00 3C ......./.......<
0150: 3D 3D 3D 3D 41 20 74 65 73 74 20 6F 66 20 6D 61 ====A test of ma
0160: 63 72 6F 20 4D 4F 56 45 2E 20 41 20 73 65 63 6F cro MOVE. A seco
0170: 6E 64 20 4D 4F 56 45 2E 00 00 00 00 00 00 00 00 nd MOVE.........
```

Wie bei der ersten Version können wir die ASCII-Zeichen am Beginn des Programms sehen, sowie den Pfeil, jedoch zeigt er jetzt auf einen Sprungbefehl nach Adresse 14C hex. Löschen Sie das Empfangsfeld mit

```
F400,42F,0
```

und starten Sie das Programm mit

```
G100,14C
```

Dies setzt einen Stop bei Adresse 14C hex, dem letzten Befehl. Der Debugger antwortet mit

```
*014C
```

Geben Sie das Empfangsfeld mit dem Kommando D400,42F aus. Sie sehen, daß die beiden MOVE-Aufrufe den folgenden zusammenhängenden Text erzeugt haben:

```
0400: 41 20 74 65 73 74 20 6F 66 20 6D 61 63 72 6F 20 A test of macro
0410: 4D 4F 56 45 2E 20 41 20 73 65 63 6F 6E 64 20 4D MOVE. A second M
0420: 4F 56 45 2E 00 00 00 00 00 00 00 00 00 00 00 00 OVE.............
```

Makro MOVE, Version 3

Manchmal muß man einen Text in einem Feld speichern. Da der Text noch nicht existiert, sind die beiden ersten Versionen des MOVE-Makros dafür nicht geeignet. In unsere dritte Version bauen wir deshalb eine neue Funktion ein. Sie erlaubt einen Text als ersten Parameter anstelle einer Speicheradresse. Wir können also schreiben:

```
MOVE    "THIRD",FCB2+1
```

um den Text „THIRD" in den Speicher zu schreiben, der ein Byte nach dem Symbol FCB2 beginnt. Sie sehen, daß der dritte Parameter und das Komma in diesem Beispiel wegfallen. Der Assembler wird die benötigte Länge selbst berechnen. Mit dieser Methode signalisieren wir, daß der erste Parameter ein Text ist und keine Adresse.

Der Textparameter kann auch aus Variablen und Konstanten bestehen, wenn der ganze Parameter in spitze Klammern eingeschlossen ist. Z. B. werden durch den Ausdruck

```
MOVE    <2,"FIFTH">,FCB1
```

sechs Bytes in den Speicher ab Adresse FCB1 (5C hex) geladen. Das erste Byte enthält eine binäre Zwei; danach folgt die Zeichenfolge FIFTH. Natürlich kan man auch Symbole wie EOF (Dateiende), CR (Carriage Return) oder LF (Line Feed) als Parameter benutzen. Das Komma trennt die Konstante 2 vom Text FIFTH.

Ändern Sie das Makro nach Abb. 4.15 ab. Die dritte Version rettet als erstes die Register (wie die ersten Versionen auch). Dann sehen wir eine neue Funktion. Wenn der Assembler den Ausdruck

```
IF      NOT NUL TO
LXI     D,TO
ENDIF
```

findet, prüft er, ob der zweite Parameter, die Adresse des Empfangsfelds, im Aufruf angegeben wurde. Wird dieser Parameter weggelassen, wird angenommen, daß diese Adresse vor dem Aufruf in das DE-Register geladen wurde. Der Ausdruck IF NOT NUL TO ist dann falsch. Ist der Parameter andererseits jedoch angegeben, dann ist dieser Ausdruck wahr, und der Befehl LXI D,TO wird generiert.

In der vorigen Version mußte das Empfangsfeld als Parameter angegeben werden. Manchmal ist diese Adresse zur Zeit der Übersetzung nicht bekannt. Bei dieser neuen Version können wir die Adresse des Empfangsfeldes aus dem Speicher laden oder als Ergebnis einer Berechnung im Programm erhalten. Angenommen, die Adresse steht im Speicher bei DEST. Dann werden die folgenden Befehle 20 Bytes beginnend bei der Adresse FROM in den Speicher, dessen Adresse in DEST steht, speichern:

```
PUSH    H
LHLD    DEST
XCHG
POP     H
MOVE    FROM,,20
```

Der nächste Teil des MOVE-Makros prüft, ob der dritte Parameter, die Anzahl der Bytes, angegeben wurde. Ist er weggelassen worden, wird eine Zeichenkette gespeichert. Die Befehle zwischen IF NUL BYTES und ELSE werden generiert. Der Assembler erzeugt den Code zum Speichern der Zeichen des ersten Parameters, angesprochen durch MESG, in das Zielfeld. Dieses Symbol findet man nahe dem Ende des Makros. MESG ist als lokales Symbol definiert. Daher kann es in jedem Makroaufruf verwendet werden.

Der alternative Teil zwischen ELSE und ENDIF wird generiert, wenn der dritte Parameter angegeben wird. Der erste Parameter kann auch weggelassen werden. Der Aufruf

```
MOVE    ,,20
```

speichert 20 Bytes von der Adresse in HL zu der Adresse in DE.

Das Makro wird fortgesetzt mit dem Aufruf der Subroutine MOVE2? und dem Wiederherstellen der Register. Der JMP AROUND ist eingebettet in einen bedingten Block, der zwei Dinge prüft: den Status von MVFLAG und,ob der dritte Parameter, BYTES, angegeben ist.

```
IF      NOT MVFLAG OR NUL BYTES
JMP     AROUND
ENDIF
```

Wenn NOT MVFLAG wahr ist, werden die Subroutine und der Übersprung benötigt. Wenn ein Text gespeichert werden soll (der dritte Parameter ist weggelassen), müssen wir den Text überspringen. In den anderen Fällen werden die Subroutine und der Übersprung nicht generiert.

```
MOVE        MACRO   FROM,TO,BYTES
;;(Tagesdatum)
;;Inline Makro zu Umspeichern von Text
;;
            LOCAL   AROUND,         MESG
            PUSH    H
            PUSH    D
            PUSH    B
            IF      NOT NUL TO
            LXI     D,TO
            ENDIF
            IF      NUL BYTES       ;;Textkonstante
                                      speichern
            LXI     H,MESG          ;;Test
            LXI     B,AROUND-MESG
            ELSE                    ;;keine Konstante
            IF      NOT NUL FROM
            LXI     H,FROM
            ENDIF
            LXI     B,BYTES
            ENDIF
            CALL    MOVE2?
            POP     B
            POP     D
            POP     H
            IF      NOT MVFLAG OR NUL BYTES
            JMP     AROUND
            ENDIF
;
            IF      NOT MVFLAG
MOVE2?:
            MOV     A,M             ;lesen
            STAX    D               ;speichern
            INX     H               ;von
            INX     D               ;nach
            DCX     B               ;Zeichenzahl
            MOV     A,C
            ORA     B
            JNZ     MOVE2?          ;weiter
            RET
```

Abb. 4.15: Das Makro MOVE, Version 3

```
;
MVFLAG      SET     TRUE            ;;eine Kopie
            ENDIF                   ;;not MVFLAG
            IF      NUL  BYTES
MESG:
            DB      FROM            ;;Textkonstante
            ENDIF
AROUND:                             ;;MOVE
            ENDM
```

Abb. 4.15: Das Makro MOVE, Version 3 (Forts.)

Beachten Sie, daß die beiden Ausdrücke der logischen Operation OR, NOT MVFLAG und NUL BYTES in der angegebenen Reihenfolge erscheinen. Sie dürfen nicht vertauscht werden, da der Assembler sie sonst anders interpretiert. Das kommt durch die Reihenfolge, in der die Operatoren NUL und OR ausgewertet werden. Der Ausdruck

```
NUL BYTES OR NOT MVFLAG
```

wird als

```
NUL (BYTES OR NOT MVFLAG)
```

interpretiert. Das ist aber nicht dasselbe wie

```
(NUL BYTES) OR NOT MVFLAG
```

und das war das gewünschte Ergebnis.

Um die dritte Version des MOVE-Makros zu testen, erstellen Sie eine neue Datei namens MOVE3.ASM, und kopieren Sie die Datei MOVE 2.ASM in sie. Ändern Sie sie nach Abb. 4.16 ab. Übersetzen Sie das Programm, und laden Sie es mit dem Debugger in den Speicher. Zeigen Sie den Anfang mit dem Kommando D100,1AF. Sie sehen den abschließenden Sprungbefehl bei Adresse 17C hex:

```
0100: C3 1A 01 56 65 72 20 31 32 2E 31 36 2E 38 31 2E ...Ver 12.16.82.
0110: 54 45 53 54 4D 4F 56 45 2E 33 E5 D5 C5 11 75 00 TESTMOVE.3....u.
0120: 21 3A 01 01 03 00 CD 2F 01 C1 D1 E1 C3 3D 01 7E !:...../.....=.~
0130: 12 23 13 0B 79 B0 C2 2F 01 C9 24 24 24 E5 D5 C5 .#..y../..$$$...
0140: 11 5C 00 21 52 01 01 06 00 CD 2F 01 C1 D1 E1 C3 .\.!R...../.....
0150: 58 01 02 46 49 46 54 48 E5 D5 C5 11 00 04 21 84 X..FIFTH......!.
0160: 01 01 14 00 CD 2F 01 C1 D1 E1 E5 D5 C5 11 14 04 ...../..........
```

```
0170: 21 98 01 01 10 00 CD 2F 01 C1 D1 E1 C3 00 00 3C !....../.......µ
0180: 3D 3D 3D 3D 41 20 74 65 73 74 20 6F 66 20 6D 61 ====A test of ma
0190: 63 72 6F 20 4D 4F 56 45 2E 20 41 20 73 65 63 6F cro MOVE. A seco
01A0: 6E 64 20 4D 4F 56 45 2E 00 00 00 00 00 00 00 00 nd MOVE.........
```

```
TITLE   'TESTMOVE zum Testen des Makros MOVE'
;
;Dec. 16, 81
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
BOOT    EQU     0               ;Warmstart
BDOS    EQU     5               ;BDOS-Eingang
TPA     EQU     100H            ;Anwenderbereich
FCB1    EQU     5CH             ;Eingabe FCB
FCB2    EQU     6CH             ;2. Parameter
;
MVFLAG  SET     FALSE           ;Blockverschiebung
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        VERSN   '12.16.81.TESTMOVE.3'
        MOVE    '$$$', FCB2+9
        MOVE    <2,'FIFTH'>,FCB1
        MOVE    TEXT, NEWTEX, TEXT2-TEXT
        MOVE    TEXT2, NEWTEX+TEXT2-TEXT, TEXEND-TEXT2
        JMP     BOOT
;
        DB      '<===='
TEXT:
        DB      'A test of Makro MOVE'
TEXT2:
        DB      '. A second MOVE.'
TEXEND:
;
ORG     400H
```

Abb. 4.16: Ein Programm zum Testen der Version 3 des Makros MOVE

```
;
NEWTEX: DS      1
;
        END     START
```

Abb. 4.16: Ein Programm zum Texten der Version 3 des Makros MOVE (Forts.)

Starten Sie die dritte Version mit dem Kommando G100,17C. Zeigen Sie den Bereich von 50 bis 7F hex mit dem Kommando D50,7F an. Sie sehen, daß die drei Dollarzeichen nach 75 hex gebracht wurden, und eine binäre Zwei nach 5C hex gefolgt von dem Text „FIFTH":

```
0050: 00 00 00 00 00 00 00 00 00 00 00 00 02 46 49 46 .............FIF
0060: 54 48 00 00 00 00 00 00 00 00 00 00 00 00 00 00 TH..............
0070: 00 00 00 00 00 24 24 24 00 00 00 00 00 00 00 00 .....$$$........
```

Zur Überprüfung kann man sich den Bereich ab Adresse 400 hex ansehen, um sich von der Richtigkeit des MOVE zu überzeugen. Alle drei Versionen des MOVE-Makros werden „inline" generiert, d. h. der Code wird dort erzeugt, wo er benötigt wird; einschließlich dem Übersprungbefehl und dem Unterprogramm beim ersten Aufruf.

EIN MAKRO ZUM FÜLLEN DES SPEICHERS MIT EINER KONSTANTEN

Mit dem gerade entwickelten MOVE-Makro können wir eine Zeichenkette in einen Speicher laden. Als Beispiel haben wir drei Dollarzeichen in den zweiten Dateikontrollblock (FCB) geladen mit der Anweisung

```
MOVE    '$$$',FCB2+9
```

Um jedoch einen großen Speicherbereich mit einer Konstanten zu füllen, ist dieses Makro nicht geeignet. Dafür werden wir das Makro FILL entwickeln. Mit diesem Makro können wir einen beliebig großen Bereich mit einer bestimmten Konstanten füllen. Wie das Makro MOVE wird dieses Makro „inline" codiert.

Bauen Sie das Makro FILL, wie in Abb. 4.17 gezeigt, in Ihre Makrobibliothek ein. Ebenfalls fügen Sie den Namen FILL in das Verzeichnis am Beginn der Makrobibliothek ein. Dies ist das zweite Makro, das ein Flag benutzt. Wir werden noch viele andere Makros einfügen, die ein Flag

benutzen, daher führen wir eine Spalte im Verzeichnis für die Aufnahme dieser Flags ein. Das Verzeichnis sieht nun also so aus:

```
;;Makros in dieser Bibliothek          Flags
;;ENTER MACRO                          (keine)
;;EXIT  MACRO  SPACE?                  (keine)
;;FILL  MACRO  ADDR,BYTES,CHAR         (FLFLAG)
;;MOVE  MACRO  FROM,TO,BYTES           (MVFLAG)
;;VERSN MACRO  NUM                     (keine)
```

```
FILL        MACRO   ADDR, BYTES, CHAR
;;(Tagesdatum)
;;Inline-Makro zum Füllen von bytes Adressen
;;mit dem Zeichen CHAR, Start bei ADDR
;;Anwendung: FILL FCB+1,BLANK,8
;;          FILL FCB+9,'?',3
;;
            LOCAL   AROUND
            PUSH    H
            PUSH    B
            IF      NOT NUL ADDR
            LXI     H,ADDR
            ENDIF
            MVI     C,BYTES
            MVI     A,CHAR
            CALL    FILL2?
            POP     B
            POP     H
            IF      NOT FLFLAG
            JMP     AROUND
FILL2?:
            MOV     M,A             ;speichern
            INX     H               ;Pointer
            DCR     C               ;Zahl
            JNZ     FILL2?          ;weiter
            RET
FLFLAG      SET     TRUE
            ENDIF
AROUND:                             ;;FILL
            ENDM
```

Abb. 4.17: Das Makro FILL zum Füllen eines Speicherblocks mit einem Byte

Wie Sie sehen, ist der erste Parameter des Makros FILL die Adresse des zu füllenden Bereiches. Wegen des bedingten Ausdrucks

```
IF      NOT NUL ADDR
LXI     H,ADDR
ENDIF
```

darf der erste Parameter weggelassen werden. Der zweite Parameter, die Anzahl der Bytes, wird in das C-Register geladen. Da dies ein 8-Bit-Register ist, ist die Größe des Bereiches auf 256 beschränkt. (Mit der Anzahl null werden 256 Bytes gefüllt.) Reicht dies nicht aus, kann das Makro wiederholt aufgerufen werden. Alternativ kann man in diesem Makro das BC-Register anstelle des C-Registers benutzen. Dies werden wir in Kapitel 8 tun.

Machen Sie eine Kopie des Testprogramms aus Abb. 4.16, und nennen Sie es TESTFILL.ASM. Ändern Sie es nach Abb. 4.18. Setzen Sie das Flag FLFLAG nahe dem Programmanfang auf FALSE. Dieses Flag dient demselben Zweck wie das Flag MVFLAG im vorigen Makro. Das Flag wird zunächst auf FALSE gesetzt, damit beim ersten Aufruf das Unterprogramm FILL2? generiert wird. Um weitere Kopien dieser Subroutine durch nachfolgende Aufrufe zu vermeiden, wird es dann auf TRUE gesetzt.

Assemblieren Sie das Programm, und laden Sie es mit dem Debugger. Geben Sie das Programm aus mit dem Kommando

```
D100,16F
```

In der Liste sehen Sie wieder den schon bekannten Pfeil, der auf einen Sprungbefehl bei Adresse 156 hex zeigt. Beachten Sie, daß die beiden Makros ENTER und EXIT in dieser Version enthalten sind. Das FILL-Makro wird hier dreimal benutzt. Beim ersten Mal werden Dollarzeichen in den zweiten FCB gespeichert. Dasselbe hatte das erste MOVE-Makro im vorigen Programm getan. Der nächste Aufruf FILL setzt 40 Bytes auf Blank, und der letzte Aufruf setzt 40 Bytes auf binär Null.

```
0100: 21 00 00 39 22 63 01 31 87 01 C3 22 01 56 65 72 !..9"c.1...".Ver
0110: 20 31 32 2E 32 34 2E 38 31 2E 54 45 53 54 46 49  12.24.81.TESTFI
0120: 4C 4C E5 C5 21 75 00 0E 03 3E 24 CD 33 01 C1 E1 LL..!u...°$.3...
0130: C3 3A 01 77 23 0D C2 33 01 C9 E5 C5 21 00 08 0E .:.w#..3....!...
0140: 40 3E 20 CD 33 01 C1 E1 E5 C5 21 40 08 0E 40 3E §> .3.....!§..§>
0150: 00 CD 33 01 C1 E1 C3 5E 01 3C 3D 3D 3D 3D 2A 63 ..3....´.<====*c
0160: 01 F9 C9 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Setzen Sie den Bereich ab 800 hex auf A5 hex mit dem Debuggerkommando F800,8FF,A5. Starten Sie das neue Programm mit dem Debuggerkommando G100,156. Zeigen Sie dann mit dem Kommando D50,7F den Dateisteuerblock an, und achten Sie auf die drei Dollarzeichen:

```
0050: 00 00 00 00 00 00 00 00 00 00 00 28 00 20 20 20 ...........(.
0060: 20 20 20 20 20 20 20 20 00 00 00 00 16 00 00 00         ........
0070: 00 00 00 00 00 24 24 24 00 00 00 00 00 00 00 00 .....$$$........
```

```
            TITLE 'TESTFILL zum Testen des Makros FILL'
            ;
            ;Dec. 24, 81
            ;
FALSE       EQU     0
TRUE        EQU     NOT  FALSE
;
BOOT        EQU     0               ;Warmstart
BDOS        EQU     5               ;BDOS-Eingang
TPA         EQU     100H            ;Anwenderbereich
FCB1        EQU     5CH             ;Eingabe FCB
FCB2        EQU     6CH             ;2. Parameter
;
FLFLAG      SET     FALSE           ;FILL-Flag
;
            MACLIB  CPMMAC
;
ORG         TPA
;
START:
            ENTER
            VERSN   '12.24.81.TESTFILL'
            FILL    FCB2+9, 3, '$'
            FILL    800H, 40H, BLANK
            FILL    800H+40H, 40H, 0
            JMP     DONE
            DB      '<===='
DONE:
            EXIT
;
            END     START
```

Abb. 4.18: Ein Programm zum Testen des Makros FILL

Eine abschließende Ausgabe des Bereiches ab 800 hex zeigt das Ergebnis des zweiten und dritten Makroaufrufes. Geben Sie D800,88F ein. Die Ausgabe sollte etwa so sein:

```
0800: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0810: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0820: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0830: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0840: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0850: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0860: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0870: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0880: A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 ................
```

Gehen Sie mit Control-C zu CP/M. Wir werden nun ein Makro schreiben, mit dem man zwei Speicherbereiche vergleicht.

EIN MAKRO ZUM VERGLEICH ZWEIER BEREICHE

Oft will man wissen, ob ein Speicherbereich denselben Inhalt hat wie ein anderer. In Kapitel 6, z. B., schreiben wir ein Programm, mit dem man eine ASCII-Datei auf dem Bildschirm anzeigt. Eine binäre COM-Datei kann man so nicht ausgeben; daher wollen wir den Typ der Datei, deren Name eingegeben wurde, mit der Zeichenfolge COM vergleichen. Das Programm kann sich dann beenden, wenn der Typ COM ist.

Als zweites Beispiel nehmen wir an, ein Programm benötigt einen Dateinamen, aber der Anwender gibt einen nicht eindeutigen Namen wie

```
SORT.*
```

ein. Der CCP ersetzt den Stern durch drei Fragezeichen. Das Programm erwartet eine einzige Datei, aber der CCP übergibt

```
SORT.???
```

In diesem Fall hat das Programm es vielleicht mit mehreren Dateien statt mit einer zu tun. Um dies zu überprüfen, vergleichen wir den Dateinamen mit einer Folge von Fragezeichen.

Das Inline-Makro COMPAR in Abb. 4.19 kann man für einen Vergleich zweier Bereiche von bis zu 256 Bytes benutzen. Will man zwei Bereiche vergleichen, gibt man die Adressen der Bereiche als ersten und zweiten Parameter an, die Anzahl der Bytes als dritten. Die maximale Länge ist 256, da das C-Register zum Zählen benutzt wird. Tragen Sie das Makro in alphabetischer Reihenfolge ein.

```
COMPAR  MACRO  FIRST, SECOND, BYTES
;;(Tagesdatum)
;;Inline-Makro zum Vergleich zweier Bereiche
;;Das Z-Flag ist gesetzt, wenn beide Bereiche gleich sind.
;;Der erste und der zweite Parameter können Adressen
;;sein, der dritte ist die Anzahl der Bytes.
;;Der erste Parameter darf eine Textkonstante sein.
;;In diesem Fall fehlt der dritte Parameter.
;;Jeder der Parameter darf fehlen.
;;Register A wird verändert.
;;
;;Anwendung: COMPAR  FCB1, FCB2, 12
;;          COMPAR  '???', FCB1+9
;;          COMPAR  , , 5
;;
          LOCAL   MESG, AROUND
          PUSH    H
          PUSH    D
          PUSH    B
          IF      NUL  BYTES
          LXI     H,MESG              ;Textkonstante
          MVI     C,AROUND-MESG       ;Länge
          ELSE
          IF      NOT  NUL  FIRST
          LXI     H,FIRST
          ENDIF
          IF      NOT  NUL  BYTES
          MVI     C,BYTES
          ENDIF
          ENDIF   ;keine              Länge
          IF      NOT  NUL  SECOND
          LXI     D,SECOND
          ENDIF
          CALL    COMP2?
          POP     B
          POP     D
          POP     H
          IF      NOT  CMFLAG  OR   NUL  BYTES
          JMP     AROUND
          ENDIF
```

Abb. 4.19: Das Makro COMPAR für einen binären Vergleich

```
        IF      NOT  CMFLAG     ;eine Kopie
COMP2?:                         ;Vergleichsroutine
        LDAX    D               ;lesen
        CMP     M               ;gleich?
        RNZ                     ;nein
        INX     H
        INX     D               ;Zeiger
        DCR     C               ;zählen
        JNZ     COMP2?          ;weiter
RET
CMFLAG  SET     TRUE            ;nur eine
        ENDIF
        IF      NUL             BYTES
MESG:   DB      FIRST           ;;Text
        ENDIF
AROUND:                         ;;COMPAR
        ENDM
```

Abb. 4.19: Das Makro COMPAR für einen binären Vergleich (Forts.)

Die bedingten Teile

```
IF      NOT NUL FIRST
```

und

```
IF      NOT NUL SECOND
```

erlauben das Weglassen eines oder beider ersten Parameter. Werden die Parameter weggelassen, dann müssen die Register vor dem Aufruf geladen werden. Der Aufruf kann dann so aussehen:

```
COMPAR  ,,8
```

Wenn Sie herausfinden wollen, ob der erste und zweite Parameter einer CP/M-Kommandozeile gleich sind, tun Sie das mit:

```
COMPAR  FCB1,FXB2,12
```

Hiermit vergleichen Sie die ersten 12 Bytes der beiden Dateisteuerblöcke. Das Makro setzt das Zero-Flag, wenn die beiden Bereiche gleich sind; sonst wird es gelöscht.

Wenn Sie einen Bereich mit einer Zeichenkette vergleichen wollen, lassen Sie den dritten Parameter weg. Der erste Parameter ist dann der Text selbst. Der Assembler bestimmt die Länge des Bereiches aus der Länge des ersten Parameters. Der Makroaufruf

```
COMPAR  '???',FCB1+9
```

setzt das Zero-Flag, wenn in den drei Bytes ab FCB1+9 Fragezeichen stehen. (FCB1+9 ist der Dateityp des ersten Parameters einer CP/M-Kommandozeile.)

Ein ASCII-Vergleich

Jedes Byte umfaßt zwar acht Bits, der ASCII-Zeichensatz nutzt jedoch nur die niederwertigen 7 Bits (0–6) aus. Da das höchste Bit, Bit 7, nicht benutzt wird, kann es andere Informationen aufnehmen. Man kann also ein 8-Bit-Byte aufteilen in ein 1-Bit-Flag gefolgt von einem 7-Bit-ASCII-Zeichen. So ein Byte hat zwei Aufgaben. CP/M benutzt diese Eigenschaft, um Dateischutz zu ermöglichen und dadurch die Wahrscheinlichkeit des ungewollten Löschens einer Datei zu reduzieren. Ein CP/M-Dateiname besteht aus dem eigentlichen Namen und einer Erweiterung von drei Zeichen. Diese Erweiterung wird gern benutzt, um die gespeicherte Information zu klassifizieren (FOR für FORTRAN, BAS für BASIC etc.). Ist das höchstwertige Bit des ersten Zeichens der Erweiterung gesetzt, betrachtet CP/M die Datei als schreibgeschützt. Ist es dagegen nicht gesetzt, kann man die Datei ändern oder löschen. Die übrigen 7 Bits enthalten das ASCII-Zeichen.

Angenommen, wir wollen sicherstellen, daß eine gegebene Datei den Typ COM hat. Wir würden dann das Makro so aufrufen:

```
COMPAR  'COM',FCB1+9
```

Dieser Vergleich ist falsch, wenn die Datei schreibgeschützt ist. Die ASCII-Verschlüsselung des Buchstaben C ist

```
100 0011
```

oder auch

```
0100 0011
```

wenn das höchstwertige Bit gelöscht ist. Ist die Datei jedoch geschützt, ist das höchstwertige Bit gesetzt. Die Verschlüsselung ist dann

```
1100 0011
```

Wir brauchen also eine andere Version des Vergleichsmakros, um nur jeweils die niederen 7 Bits eines Bytes zu vergleichen. Das Makro in Abb. 4.20 kann hierfür benutzt werden. Tragen Sie es in Ihre Bibliothek ein.

```
COMPRA   MACRO   FIRST,SECOND,BYTES
;;(Tagesdatum)
;;ASCII-Version (höchstes Bit wird gelöscht).
;;Inline-Makro zum Vergleich zweier Bereiche.
;;Das Z-Flag ist gesetzt, wenn beide Bereiche gleich sind.
;;Der erste und der zweite Parameter können Adressen
;;sein, der dritte ist die Anzahl der Bytes.
;;Der erste Parameter darf eine Textkonstante sein.
;;In diesem Fall fehlt der dritte Parameter.
;;Jeder der Parameter darf fehlen.
;;Register A wird verändert.
;;
;;Anwendung:  COMPRA  FCB1,FCB2,11
;;          COMPRA  'COM',FCB1+9
;;          COMPRA  ,FCB1+1,11
;;
            LOCAL   MESG, AROUND
            PUSH    H
            PUSH    D
            PUSH    B
            IF      NUL  BYTES
            LXI     H, MESG             ;Textkonstante
            MVI     C,AROUND-MESG       ;Länge
            ELSE
            IF      NOT NUL FIRST
            LXI     H,FIRST
            ENDIF
            IF      NOT NUL C
            MVI     C,BYTES
            ENDIF
            ENDIF                       ;keine Länge
            IF      NOT NUL SECOND
            LXI     D,SECOND
            ENDIF
            CALL    COMP2?
            POP     B
```

Abb. 4.20: Das Makro COMPRA für einen ASCII-Vergleich

```
            POP     D
            POP     H
            IF      NOT CMFLAG OR NUL BYTES
            JMP     AROUND
            ENDIF
            IF      NOT CMFLAG      ;eine Kopie
COMP2?:                             ;Vergleichsroutine
            LDAX    D               ;lesen
            ANI     7FH             ;Bit 7 löschen
            PUSH    B
            MOV     C,A
            MOV     A,M
            ANI     7FH
            CMP     C               ;gleich?
            POP     B
            RNZ                     ;nein
            INX     H
            INX     D               ;Zeiger
            DCR     C               ;und zählen
            JNZ     COMP2?          ;weiter
RET
CMFLAG      SET     TRUE            ;nur eine
            ENDIF
            IF      NUL  BYTES
MESG:       DB      FIRST           ;;Text
            ENDIF
AROUND:                             ;;COMPRA
            ENDM
```

Abb. 4.20: Das Makro COMPAR für einen ASCII-Vergleich (Forts.)

EIN MAKRO ZUM ÄNDERN VON KLEINBUCHSTABEN IN GROSSBUCHSTABEN

Kleinbuchstaben in CP/M-Kommandozeilen werden automatisch in Großbuchstaben verwandelt. Will man jedoch von einem Programm etwas einlesen, geschieht dies nicht. Angenommen, ein Programm gibt folgende Zeile aus:

```
ALLE DATEIEN LOESCHEN
```

Es reicht nicht aus, die Antwort mit der Anweisung

```
CP      'J'
```

zu testen, denn die Eingabe kann ein Klein- oder ein Großbuchstabe sein. Natürlich kann man beide Möglichkeiten mit weiteren Befehlen testen:

```
CP      'J'
JZ      ...
CP      'j'
JZ      ...
```

Eine bessere Methode bietet das Makro in Abb. 4.21, mit dem kleine in große Buchstaben umgewandelt werden. Das Makro wird direkt vor dem Vergleich aufgerufen:

```
UCASE
CP      'J'
JZ      ...
```

```
UCASE   MACRO   REG
;;(Tagesdatum)
;;Inline-Makro, um ein Zeichen in einem beliebigen
;;Register in einen Großbuchstaben umzuwandeln.
;;Fehlt der Parameter, wird Register A angenommen.
;;
;;Anwendung:  UCASE
;;          UCASE   C
;;
            LOCAL   NOTUP?
            IF      NOT NUL REG
            PUSH    PSW         ;retten
            MOV     A,REG       ;Zeichen laden
            ENDIF
            CPI     'Z'+7       ;Großbuchstabe?
            JC      NOTUP?      ;nein
            ANI     5FH         ;umwandeln
NOTUP?:
            IF      NOT NUL REG
            MOV     REG,A       ;im Register speichern
            POP     PSW         ;wieder laden
ENDIF
                                ;;UCASE
            ENDM
```

Abb. 4.21: Das Makro UCASE, das einen Kleinbuchstaben in einen Großbuchstaben umwandelt

Zum besseren Verständnis schauen wir uns den ASCII-Code näher an. Das große Y unterscheidet sich von dem kleinen y nur durch ein Bit:

```
Y    0101 1001 (groß)
y    0111 1001 (klein)
```

Bei den übrigen Buchstaben sieht es entsprechend aus. Aus dem Beispiel sehen wir, daß man durch Löschen von Bit 5 aus einem Kleinbuchstaben einen Großbuchstaben macht. Dafür benutzen wir die logische Operation AND mit dem Wert 5F hex.

```
     y     0111 1001 (klein)
AND  5F    0101 1111
     ---------------
     Y     0101 1001 (groß)
```

Diese Methode funktioniert für Kleinbuchstaben. Auch für Großbuchstaben kann man sie benutzen:

```
     Y     0101 1001 (groß)
AND  5F    0101 1111
     ---------------
     Y     0101 1001 (groß)
```

D. h. angewendet auf kleine oder große Buchstaben ergibt diese Methode Großbuchstaben. Beachten Sie, daß man diese Technik nur für Buchstaben verwenden darf.

Betrachten wir, was passiert, wenn wir das logische AND mit dem Wert 5F hex auf die Ziffer 8 anwenden. Die Bitmuster sind:

```
     8     0011 1000 (Ziffer 8)
AND  5F    0101 1111
     ---------------
     0001 1000 (Control-X)
```

Wir haben die Ziffer 8 in das Zeichen Control-X verwandelt. Wir müssen also darauf achten, daß wir die Konvertierung nur auf Buchstaben anwenden. (Es gibt einige Sonderzeichen wie Klammern, die im Bereich der Kleinbuchstaben liegen. Das sollte jedoch kein Problem sein.)

Das Makro enthält die folgenden Befehle:

```
CPI     'Z'+7
JC      ...
```

Mit dem CPI-Befehl stellt man fest, ob das Zeichen ein Kleinbuchstabe ist. Der Wert des Buchstaben „a" ist um sieben größer als der des Großbuchstaben „Z". Daher übergeht der JC-Befehl das logische AND. (Wenn wir es für wichtig halten, können wir einen zweiten Test einfügen für Zeichen mit Werten größer als „z". Dadurch wäre sichergestellt, daß nur die Buchstaben von „a" bis „z" konvertiert werden. Es gibt jedoch nur wenige Zeichen im ASCII-Zeichensatz oberhalb von „z".)

Eine weitere Möglichkeit bietet der zweite optionale Parameter des Makros UCASE. Fehlt der Parameter, wird das Zeichen im A-Register erwartet. Wird jedoch ein Register als Parameter angegeben, dann wird mit zusätzlichen Befehlen das Zeichen in diesem Register bearbeitet. Der Makroaufruf

```
UCASE   C
```

z. B. erzeugt zusätzlich die Befehle

```
PUSH    PSW
MOV     A,C
```

zu Beginn der Makrogenerierung und die Befehle

```
MOV     C,A
POP     PSW
```

am Ende.

Wir werden das Makro UCASE gewöhnlich in Programmen benutzen, in denen Eingaben des Anwenders verlangt werden.

EIN MAKRO ZUR UMWANDLUNG EINES NICHT EINDEUTIGEN DATEINAMENS IN EINEN EINDEUTIGEN

In Kapitel 7 werden wir ein Programm zum Umbenennen von Dateien schreiben. In diesem Programm sind nicht eindeutige Namen zugelassen, und der alte Dateiname sowie der neue werden ausgegeben.

Geben wir das Kommando

```
RENAME SORT.PAS *.BAK
```

soll dasselbe geschehen wie mit dem Kommando

```
RENAME SORT.PAS SORT.BAK
```

D. h. der Dateiname *.BAK muß in SORT.BAK umgewandelt werden. Diese Konvertierung machen wir in zwei Schritten.

CP/M verwandelt den ersten Parameter geringfügig und stellt ihn in den Dateisteuerblock bei 5C hex. Dieser Speicher heißt in diesem Buch FCB1

```
AMBIG   MACRO   OLD, NEW
;;(Tagesdatum)
;;Inline-Makro, um einen nicht-eindeutigen Dateinamen
;;bei FCB NEW in einen dem bei FCB OLD entsprechenden umzuwandeln.
;;
;;Anwendung:  AMBIG    FCB1, FCB2
;;
            PUSH    H
            PUSH    D
            PUSH    B
            LXI     H,NEW+1
            LXI     D,OLD+1
            MVI     C,11        ;Anzahl der Zeichen
AMB2?:
            MVI     A,'?'
            CMP     M           ;Fragezeichen?
            JNZ     AMB3?       ;nein
;
;Kopiere ein Zeichen
;
            LDAX    D           ;lies Zeichen in OLD
            MOV     M,A         ;speichere in NEW
AMB3?:
            INX     H           ;neu
            INX     D           ;alt
            DCR     C           ;Zähler
            JNZ     AMB2?
            POP     B
            POP     D
            POP     H
                                ;;AMBIG
            ENDM
```

Abb. 4.22: Das Makro AMBIG, das einen nicht eindeutigen Dateinamen in einen eindeutigen umwandelt

(oder meist nur FCB). CP/M entfernt den Punkt zwischen dem Namen und der Namenserweiterung. Der Name wird ggfs. mit Leerzeichen auf acht Zeichen verlängert und ab 50 hex gespeichert. Die Erweiterung steht unmittelbar hinter dem Namen.

Der zweite Parameter wird im Bereich ab 6C hex gespeichert. Diesen Bereich nennen wir FCB2. CP/M ersetzt den Stern durch acht Fragezeichen und speichert sie ab 60 hex. Die Erweiterung steht wiederum hinter dem Namen. Aus dem zweiten Parameter wird dann ????????.BAK.

Die Fragezeichen des zweiten Dateinamens müssen in unserem Programm in die vier Buchstaben „SORT" und vier Blanks gewandelt werden.

Mit dem Makro AMBIG in Abb. 4.22 kann man einen nicht eindeutigen Dateinamen in einen eindeutigen verwandeln. In diesem Beispiel ist die Adresse des eindeutigen Namens der erste Parameter (OLD) und die Adresse des nicht eindeutigen der zweite Parameter (NEW). Die Namen werden zeichenweise bearbeitet. Immer, wenn im nicht eindeutigen Namen ein Fragezeichen steht, wird es durch das entsprechende Zeichen des eindeutigen Namens ersetzt. Das erste Fragezeichen durch S, das zweite durch O, etc. Tragen Sie dieses Makro in Ihre Bibliothek ein.

Das Makro beginnt mit dem Retten der Register HL, DE und BC. HL und DE werden dann mit den Werten der Parameter NEW und OLD geladen. Eine 11, die Länge des Namens (8+3), wird in Register C geladen.

In das A-Register wird das Fragezeichen geladen. Dann wird mit dem Befehl

```
CPM     M
```

jedes Zeichen des neuen Namens mit dem Fragezeichen verglichen. Wird ein Fragezeichen entdeckt, dann wird es durch das entsprechende Zeichen des alten Namens ersetzt. Die Befehle dazu sind:

```
LDAX    D
MOV     M,A
```

Nach jedem Vergleich wird der Zähler in Register C vermindert. Ist er null geworden, dann ist die Arbeit beendet. Die Registerinhalte werden mit POP wiederhergestellt.

EIN MAKRO ZUM VERSCHIEBEN DER HÖHERWERTIGEN VIER BITS IN DIE VIER NIEDERWERTIGEN BITS

Zahlen werden in einem Computer auf drei Arten verschlüsselt:

ASCII, binär oder binär-dezimal (BCD – binary-coded decimal). ASCII-Ziffern erfordern 7 Bits, weshalb in einem Byte nur eine ASCII-Ziffer gespeichert werden kann. In der binären Art können Zahlen von 0 bis 255 (1 kleiner als 2 hoch 8) in einem Byte gespeichert werden. Im BCD-Code wird jede Ziffer in vier Bits gespeichert. Daher kann ein Byte BCD-Zahlen von 0 bis 99 aufnehmen.

```
UPPER   MACRO   REG
;;(Tagesdatum)
;;Makro, um die oberen 4 Bits des
;;Akkumulators in die unteren 4 Bits zu laden. Die
;;neuen oberen 4 Bits werden gelöscht.
;;Mit diesem Makro wird die linke Ziffer
;;einer gepackten BCD-Zahl isoliert.
;;
;;Anwendung:  UPPER
;;              OUTHEX   ;drucken
;;
              IF       NOT NUL REG
              PUSH     PSW       ;A retten
              MOV      A,REG     ;laden in das A-Register
              ENDIF
              RAR                ;in die untere
              RAR                ;Hälfte schieben
              RAR
              RAR
              ANI      0FH       ;obere Hälfte löschen
              IF       NOT NUL REG
              MOV      REG,A     ;im Register speichern
              POP      PSW       ;wiederherstellen
              ENDIF
                                 ;;UPPER
              ENDM
```

Abb. 4.23: Das Makro UPPER, das die oberen 4 Bits eines Bytes in die unteren 4 Bits verschiebt

Die BCD-Methode bedeutet nichts anderes als hexadezimal, außer daß die Ziffern A bis F nicht benutzt werden. Daher kann eine Routine zur Umwandlung von binär in hexadezimal auch für BCD-Zahlen benutzt werden. Im nächsten Kapitel werden wir ein Makro für die Konvertierung einer binären Zahl in zwei hexadezimale Ziffern schreiben.

Es gibt jedoch Fälle, in denen wir nur an dem linken Teil (oder Nibble) einer BCD- oder Hexadezimalzahl interessiert sind (z. B. im Makro OUTHEX in Kapitel 5). Wir schreiben also ein Makro, um diese obere Hälfte eines Bytes zu erhalten. Das Makro UPPER in Abb. 4.23 rotiert die oberen vier Bits in die unteren vier Bits (durch vierfache Anwendung des RAR-Befehls), und löscht die oberen vier Bits durch logisches AND mit dem Wert OF hex. Ist der optionale Parameter angegeben, dann wird die Operation auf das als Parameter angegebene Register (inkl. Speicher) angewendet. Tragen Sie das Makro in die Bibliothek ein und den Namen in das Verzeichnis.

EIN MAKRO FÜR EINE 16-BIT-SUBTRAKTION

Sowohl der 8080 als auch der Z80 können eine 8-Bit-Addition und eine 8-Bit-Subtraktion mit oder ohne Carry ausführen. Darüber hinaus kann der Z80 eine 16-Bit-Subtraktion mit Carry ausführen. Es ist wichtig, zu beachten, daß der Z80 diese Subtraktion immer mit Carry ausführt.

```
SBC             MACRO
;;(Tagesdatum)
;;Inline-Makro, um DE von HL zu subtrahieren.
;;Das Ergebnis steht in HL. Dies ist etwa der
;;Z80-Befehl SBC HL,DE .
;;
;; Anwendung:   SBC oder SBC HL,DE
;;
                MOV     A,L
                SUB     E
                MOV     L,A
                MOV     A,H
                SBB     D
                MOV     H,A
                                        ;;SBC
                ENDM
```

Abb. 4.24: Das Makro SBC für eine 16-Bit-Subtraktion ohne Carry

Daher müssen wir das Carry-Flag immer vor der Subtraktion löschen, da es natürlich das Ergebnis beeinflußt.

Das letzte Makro in diesem Kapitel sehen Sie in Abb. 4.24. Es kann für eine 16-Bit-Subtraktion ohne Berücksichtigung des Carry-Flags benutzt werden. Wir werden dieses Makro SBC in mehreren Programmen benutzen, um die Differenz zweier Adressen zu berechnen.

Die 8080-Version der Doppelregister-Subtraktion berechnet die Differenz von dem Wert in HL und dem in DE. Das Ergebnis steht dann in HL. Der Zustand des Carry-Flags geht nicht in die Berechnung ein, entspricht jedoch am Ende dem Ergebnis. D. h., wenn der alte Wert in DE größer war als der in HL, wird das Carry-Flag am Ende der Operation gesetzt. Das Makro entspricht den Z80-Befehlen

```
OR      A
SBC     HL,DE
```

ZUSAMMENFASSUNG

In diesem Kapitel haben wir den Wert der Makros erkannt und haben mehrere elementare Makros entwickelt. In späteren Kapiteln werden wir sie in weiteren Programmen verwenden. All diese Makros haben eines gemeinsam – sie benutzen keine BDOS-Aufrufe. In den nächsten drei Kapiteln werden wir uns Makros mit BDOS-Aufrufen ansehen und Programme mit diesen Makros schreiben.

Das Verzeichnis unserer Makrobibliothek sollte jetzt so aussehen:

```
;;Makros in dieser Bibliothek                    Flags
;;AMBIG    MACRO OLD,NEW                         (keine)
;;COMPAR   MACRO FIRST,SECOND,BYTES              CMFLAG
;;COMPRA   MACRO FIRST,SECOND,BYTES              CMFLAG
;;ENTER    MACRO                                 (keine)
;;EXIT     MACRO SPACE?                          (keine)
;;FILL     MACRO ADDR,BYTES,CHAR                 FLFLAG
;;MOVE     MACRO FROM,TO,BYTES                   MVFLAG
;;SBC      MACRO                                 (keine)
;;UCASE    MACRO REG                             (keine)
;;UPPER    MACRO REG                             (keine)
;;VERSN    MACRO NUM                             (keine)
```

Kapitel 5

Die Benutzung von BDOS für einfache Aufgaben

EINFÜHRUNG

In diesem Kapitel lernen wir, wie man Konsol-Ein-/Ausgaben sowie Druckausgaben unter Benutzung des CP/M Basis-Disketten-Betriebssystems (BDOS) macht. Wir entwickeln eine Reihe nützlicher Makros, um dies zu erleichtern. Unter anderem werden wir Makros schreiben für die Konvertierung von Binärzahlen in dezimale und hexadezimale sowie die Konvertierung von hexadezimalen Zahlen in binäre. Schließlich werden wir diese Makros in Programmen benutzen, um mehr über die Organisation von CP/M zu erfahren. Das Programm CPU bestimmt, ob ein 8080 oder ein Z80 vorliegt; IOBYTE gibt das IOBYTE aus und ändert die CP/M IOBYTE-Funktion, die wir in Kapitel 3 entworfen haben; GO verzweigt zu einer absoluten Adresse im Speicher; PAGE bewirkt einen oder mehrere Seitenvorschübe auf dem Drucker.

BDOS-AUFRUFE

Wie wir in Kapitel 1 gesehen haben, unterteilt CP/M den Speicher in mehrere verschiedene Bereiche. Den obersten Bereich nennt man „Full Disk-Operating System" (FDOS); er ist wiederum in zwei Bereiche unterteilt. Das „Basic Input-Output System" (BIOS) belegt den oberen Teil des FDOS, das „Basic Disk-Operating System" (BDOS) den unteren. In Kapitel 3 haben wir den Aufbau des BIOS untersucht und einige neue Funktionen eingebaut. Betrachten wir nun das BDOS.

Das BIOS enthält die Primitivroutinen für den Betrieb der Konsole, des Druckers und der Diskettenlaufwerke. Diese Routinen müssen den tatsächlich an den Computer angeschlossenen Geräten angepaßt werden. Verschiedene Rechner haben verschiedene Versionen des BIOS. Man könnte CP/M-Programme unter Benutzung des BIOS schreiben, einfacher jedoch ist es, das BDOS als Bindeglied zum BIOS zu benutzen. Alle Konsol-, Drucker- und Disketten-Operationen können mit dem BDOS durchgeführt werden unter Verwendung einer festen Adresse im Spei-

cher. Da das BDOS geräteunabhängig ist, können Programme des einen Computers auch auf einem anderen laufen, auch wenn die Hardware und das BIOS verschieden sind.

Die Benutzung des BDOS für den Datenaustausch ist nicht nur vielseitiger, sie ist auch bequemer. Sie erinnern sich, daß die ersten drei Bytes des Speichers, ab Adresse 0, einen Sprungbefehl zum Warmstart des BIOS enthalten. Das nächste Byte ist das IOBYTE bei Adresse 3. Das nächste Byte, bei Adresse 4, zeigt zwei Dinge an: das momentane Laufwerk und den momentanen Benutzer. Die nächsten drei Bytes, ab Adresse 5, enthalten einen Sprungbefehl zum BDOS. Diese Adresse wird immer dann aufgerufen, wenn Konsol-, Drucker- oder Disketten-Ein-/Ausgaben erforderlich sind. Beachten Sie den Unterschied zwischen diesem einzelnen Sprungbefehl in das BDOS und dem mehrfachen Sprungvektor am Beginn des BIOS. Das BIOS hat einen eigenen Eingang für jede einzelne Operation.

Wir wenden uns nun einfachen BDOS-Operationen zu.

Die Nicht-Disketten-Funktionsnummern

Wenn ein Programm periphere Geräte mit dem BDOS ansprechen will, ruft es das BDOS über die Adresse 5 auf. Das C-Register des 8080 oder Z80 muß dafür eine Funktionsnummer enthalten, die der gewünschten

Funktion Nummer (in C)	Operation	Übergebener Wert	Erhaltener Wert
1	Konsoleingabe		Zeichen in A
2	Konsolausgabe	Zeichen in E	
3	Lesereingabe		Zeichen in A
4	Stanzerausgabe	Zeichen in E	
5	Druckerausgabe	Zeichen in E	
6	Direkte Konsol-Ein-Ausgabe	FF (Eingabe) Zeichen (Ausgabe)	0 = nicht bereit oder Zeichen in A
7	Lies IOBYTE		Byte in A
8	Speichere IOBYTE	in E	
9	Drucke Puffer	Adresse in DE	
10	Lies Puffer	Adresse in DE	
11	Lies Konsolstatus		Byte in A
12	Lies CP/M-Version		Byte in A und L

Tabelle 5.1: Die Nicht-Disketten-Funktion des BDOS

Operation entspricht. Die vom Programm zu sendende Information steht dann im E-Register, wenn es eine Byte-Größe ist oder im DE-Register bei einer Zwei-Byte-Größe. Die zurückerhaltene Information steht normalerweise im A-Register bei einem Byte oder im HL-Register bei zwei Bytes.

Mit dem BDOS kann man viele verschiedene Funktionen aufrufen. Diese Funktionen kann man in zwei Gruppen unterteilen. Eine Gruppe behandelt Konsole, Drucker, Leser und Stanzer. Die andere Gruppe führt Diskettenoperationen aus, die wir im nächsten Kapitel behandeln. Hier wollen wir die Nicht-Disketten-Funktionen untersuchen. Tabelle 5.1 zeigt die ersten 12 BDOS-Funktionen. Sie umfassen die vier logischen Geräte – Konsole, Drucker, Leser und Stanzer – und die Behandlung des IOBYTE und der CP/M-Versionsnummer. Wir werden diese Funktionen im Laufe dieses Kapitels erläutern. Wir wollen nun ein allgemeines Makro zum Aufruf des BDOS betrachten.

EIN MAKRO ZUM AUFRUF DES BDOS

BDOS-Aufrufe arbeiten alle gleich. Adresse 5 wird mit einer Funktionsnummer im Register C aufgerufen. Information wird gesendet im E- oder DE-Register und empfangen im A- oder HL-Register. Da die Inhalte der CPU-Register durch BDOS-Aufrufe verändert werden, rettet man sie gewöhnlich vor dem BDOS-Aufruf in den Stack. Nach Beendigung des BDOS-Aufrufes lädt man sie dann wieder. Jedoch ist Vorsicht geboten: Wenn wir das A-Register mit PUSH PSW retten und mit POP PSW wieder laden, verlieren wir die Information, die uns BDOS im A-Register geliefert hat. Daher sollte man das A-Register bei Eingabeaufrufen nicht retten.

Das Makro in Abb. 5.1 wird von einigen anderen Makros, die wir schreiben, benutzt werden. Tragen Sie es in alphabetischer Reihenfolge in die Makrobibliothek ein, und vergessen Sie nicht den Eintrag im Verzeichnis.

Unsere Makros sind normalerweise für direkten Inline-Gebrauch konzipiert. Wird ein Text oder ein Unterprogramm generiert, muß es übersprungen werden. Das Makro SYSF wird jedoch immer als Unterprogramm benutzt. Wir benötigen hier keinen Übersprung, da das Makro normalerweise von einem anderen Makro aufgerufen wird, das seinerseits den Sprungbefehl enthält. Nur wenn man das Makro SYSF direkt aufruft, muß man einen Sprungbefehl vorsehen.

Das Makro SYSF hat zwei Parameter. Der erste Parameter ist die Funktionsnummer, die in das C-Register geladen wird. Der zweite Parameter ist wahlfrei. Wird er benutzt, wird vor dem Aufruf einer BDOS-Funktion

der Wert des A-Registers in das E-Register umgeladen und das A-Register mit PUSH PSW gerettet. Die BDOS-Adresse wird aufgerufen, um die gewünschte Funktion auszuführen. Nach der Rückkehr vom BDOS wird das A-Register mit POP PSW wieder geladen, wenn es vorher gerettet worden war. Die übrigen Register werden wieder geladen, und die Kontrolle wird an das aufrufende Programm zurückgegeben.

EIN MAKRO ZUM LESEN EINES ZEICHENS VON DER KONSOLE

Die beiden ersten BDOS-Funktionen sind sehr wichtig. Funktion 1 benutzt man, um ein einzelnes Zeichen von der Konsole zu lesen, und Funktion 2, um ein einzelnes Zeichen zur Konsole auszugeben. Diese beiden Funktionen sind nicht komplementär. Wird nämlich ein Zeichen mit der Konsole eingegeben, wird es zusätzlich auf dem Bildschirm ausgegeben. Funktion 3 ist der Funktion 1 ähnlich, nur wird hier ein Zeichen – statt von der Konsole – vom logischen Leser erhalten.

Wir erhalten ein Zeichen von der Konsole, indem wir die Funktionsnummer 1 in das Register C laden und Adresse 5 aufrufen. Die 8080-Befehle hierzu sind:

```
MVI  C,1
CALL 5
```

Diese Befehle können wir bei entsprechender Parametrisierung mit dem Makro SYSF generieren. Bei Ausführung dieser Befehle ruft BDOS den BIOS-Vektor auf, der die Konsol-Eingabe ausführt. Das nächste auf der Konsole eingegebene Zeichen wird vom BIOS gelesen. Die Kontrolle wird danach über BDOS dem Anwenderprogramm wiedergegeben. Das Zeichen steht dann im A-Register zur Verfügung.

Gelegentlich will man auch den Konsolstatus prüfen, um zu sehen, ob eine Taste gedrückt wurde oder nicht. Dafür laden wir die Funktionsnummer 11 in das Register C und rufen die BDOS-Adresse auf. Wir erhalten dadurch im A-Register den Wert FF hex, wenn ein Zeichen verfügbar ist; andernfalls hat das A-Register den Wert 0.

Wird mit Funktion 1 eine Konsol-Eingabe gemacht, wartet das System, bis ein Zeichen verfügbar ist. Da diese Prüfung automatisch erfolgt, ist es nicht nötig, vorher mit der Funktion 11 den Konsolstatus zu prüfen. Andererseits ruht das Programm so lange, bis ein Zeichen eingegeben wird.

Bei Benutzung der Funktion 1 werden druckbare ASCII-Zeichen, wie

```
SYSF    MACRO    FUNC, AE
;;(Tagesdatum)
;;Makro zum Generieren von BDOS-Calls.
;;FUNC ist die BDOS-Funktionsnummer für C.
;;DIES IST KEIN INLINE-MAKRO.
;;Lade A nach E, wenn der zweite Parameter angegeben ist.
;;
;;Anwendung:  OPEN:    SYSF   15
;;            PCHAR:   SYSF   2, AE
;;
            PUSH     H
            PUSH     D
            PUSH     B
            MVI      C,FUNC
            IF       NOT NUL AE
            MOV      E,A          ;Konsole und Drucker
            PUSH     PSW          ;rette A
            CALL     BDOS
            POP      PSW
            ELSE
            CALL     BDOS
            ENDIF
            POP      B
            POP      D
            POP      H
            RET
                                  ;;SYSF
            ENDM
*PROGX*
```

Abb. 5.1: Das Makro SYSF zum Generieren von BDOS-Calls

Ziffern und Buchstaben, auf dem Bildschirm angezeigt, nachdem sie eingegeben wurden. Steuerzeichen, wie Return, Line-Feed, Tab (Control-I), Backspace (Control-H) und Control-C, können auf diese Art auch eingegeben werden, sie werden jedoch auf dem Bildschirm nicht angezeigt.

Das Makro READCH ist in Abb. 5.2 wiedergegeben. Tragen Sie es in Ihre Makrobibliothek ein. Wird kein Parameter angegeben, generiert es Befehle zum Lesen eines Zeichens von der Konsole, und das Zeichen wird im A-Register übergeben. Wird dagegen ein Parameter angegeben, wird das Zeichen in dem angegebenen Register übergeben.

```
READCH   MACRO   REG
;;(Tagesdatum)
;;Inline-Makro zum Lesen eines Zeichens von
;;der Konsole; das Zeichen wird in Register
;;A übergeben, wenn der zweite Parameter fehlt.
;;benötigtes Makro: SYSF
;;
;;Anwendung:  READCH
;;          READCH  C
;;
            LOCAL   AROUND
            CALL    RDCH?
            IF      NOT NUL REG
            MOV     REG, A
            ENDIF
            IF      NOT CIFLAG
            JMP     AROUND
RDCH?:      SYSF    1
CIFLAG      SET     TRUE      ;nur eine Kopie
            ENDIF
AROUND:                       ;;READCH
            ENDM
```

Abb. 5.2: Das Makro READCH zum Lesen eines Zeichens von der Konsole

EIN MAKRO ZUM SCHREIBEN EINES ZEICHENS AUF DIE KONSOLE

In einem Programm kann man eine Konsol-Ausgabe dadurch ausführen, daß man das Zeichen in das E-Register lädt, eine 2 in das C-Register und das BDOS über Adresse 5 aufruft. Die Funktionen 4 und 5 ähneln der Funktion 2; unterschiedlich ist nur der Empfänger der Ausgabe. Ist im Register C der Wert 4, wird das Zeichen in Register E zum Stanzer gesendet. Ist die Funktionsnummer 5, geht das Zeichen zum Drucker.

Das Makro PCHAR in Abb. 5.3 führt die BDOS-Funktion 2 aus. Wir werden es noch häufig benutzen, um einzelne Zeichen auf die Konsole auszugeben. Tragen Sie es in Ihre Bibliothek ein. Beachten Sie, daß das Makro SYSF hiervon benutzt wird.

Das Makro PCHAR wird benutzt, um ein Byte aus dem A-Register anzuzeigen. Der Makroname PCHAR wird im Programm benutzt wie ein

```
PCHAR   MACRO   PAR
;;(Tagesdatum)
;;Inline-Makro zum Ausgeben eines Zeichens auf die Konsole.
;;Parameter, falls angegeben, wird in das A-Register geladen
;;benötigtes Makro: SYSF
;;
;;Anwendung:  PCHAR
;;          PCHAR   '*'
;;
            LOCAL   AROUND
            IF      NOT NUL PAR
            MVI     A,PAR
            ENDIF
            CALL    PCH2?
            IF      NOT COFLAG
            JMP     AROUND
PCH2?:      SYSF    2,AE
COFLAG      SET     TRUE        ;nur eine Kopie
            ENDIF
AROUND:                         ;;PCHAR
            ENDM
```

Abb. 5.3: Das Makro PCHAR zum Ausgeben eines Zeichens auf die Konsole

Operationscode. Es kann auch für die Anzeige eines konstanten Wertes benutzt werden. Die Konstante wird beim Makroaufruf als Parameter angegeben. Wollen wir z.B. einen Stern ausgeben, schreiben wir

```
PCHAR '*'
```

Findet der Assembler den Parameter, generiert er zusätzliche Befehle, um das Zeichen in das A-Register zu laden. Er generiert dieselben Befehle wie die beiden Zeilen

```
MVI A,'*'
PCHAR
```

Braucht man zwei gleiche Zeichen, muß man den Parameter nicht noch einmal angeben:

```
PCHAR '$'       ; Dollarzeichen anzeigen
PCHAR           ; zweites Dollarzeichen
```

Diese Befehle geben zwei Dollarzeichen aus. Da der ursprüngliche Wert des Akkumulators verloren geht, kann es ein Problem geben. Wollen Sie z.B. ein bestimmtes Zeichen ausgeben und danach den alten Wert des A-Registers, müssen Sie vorher diesen alten Wert retten. Das kann dann etwa so aussehen:

```
PUSH    PSW
PCHAR   '*'   ; Stern ausgeben
POP     PSW
PCHAR         ; alten Wert ausgeben
```

EIN MAKRO ZUM BEGINN EINER NEUEN ZEILE

PCHAR benutzt man, um einzelne Zeichen auszugeben. Häufig möchten wir aber Return gefolgt von einem Line-Feed ausgeben, was dem Beginn einer neuen Zeile entspricht. Da dies zwei Aufrufe des Makros PCHAR

```
CRLF    MACRO
;;(Tagesdatum)
;;Inline-Makro für die Ausgabe der
;;Zeichen für den Beginn einer neuen Zeile.
;;Alle Register bleiben erhalten.
;;benötigtes Makro: PCHAR
;;
        LOCAL   AROUND
        CALL    CRLF2?
        IF      NOT CRFLAG  ;nur einmal
        JMP     AROUND
CRLF2?:
        PUSH    PSW
        PCHAR   CR
        PCHAR   LF
        POP     PSW
        RET
CRFLAG  SET     TRUE        ;nur eine Kopie
        ENDIF
AROUND:                     ;;CRLF
        ENDM
*PROGX*
```

Abb. 5.4: Das Makro CRLF für den Beginn einer neuen Zeile

erfordert, werden wir ein kurzes Makro schreiben, das dies vereinfacht. Tragen Sie das Makro in Abb. 5.4 in Ihre Makrobibliothek ein.

Das Makro CRLF hat keine Parameter. Es wird im Programm aufgerufen, wenn wir eine neue Zeile beginnen wollen. Am Anfang des Makros wird die globale Subroutine CRLF? aufgerufen, die die gewünschte Aufgabe erledigt. Nach dem Retten des A-Registers wird das Makro PCHAR zweimal aufgerufen. Das A-Register wird wiederhergestellt, und es geht weiter am Beginn des Makros. Der Übersprung erlaubt die Inline-Generierung.

Das Makro CRLF erfordert zwei Flags - COFLAG für das Makro PCHAR und CRFLAG für dieses Makro. Das letztere stellt sicher, daß die Subroutine CRLF? und der Übersprung nur einmal generiert werden. Jeder nachfolgende Aufruf des Makros CRLF erzeugt nur den CALL zum Unterprogramm CRLF?.

EIN PROGRAMM ZUM TESTEN DER MAKROS SYSF, READCH, PCHAR UND CRLF

Mit dem Programm in Abb. 5.5 kann man die Makros SYSF, READCH, PCHAR und CRLF testen. Tippen Sie das Programm ein, assemblieren und starten Sie es. Das Programm beginnt mit den gewohnten Makros ENTER und VERSN. Dann wird mit dem Makro CRLF eine neue Zeile begonnen. Mit dem Makro PCHAR wird ein Doppelpunkt ausgegeben, und das Makro READCH wartet auf eine Eingabe.

```
TITLE   'TEST PCHAR'
;
;(Tagesdatum)
;
FALSE   EQU     0
TRUE    EQU     NOT  FALSE
;
BOOT    EQU     0               ;Warmstart
BDOS    EQU     5               ;BDOS-Eingang
TPA     EQU     100H            ;Programstart
;
CIFLAG  SET     FALSE           ;für READCH
CRFLAG  SET     FALSE           ;für CRLF
```

Abb. 5.5: Ein Programm zum Testen der Makros SYSF, READCH, PCHAR und CRLF

```
COFLAG  SET     FALSE           ;für PCHAR
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum)'
NEXT:
        CRLF
        PCHAR   ':'             ;Prompt
        READCH                  ;Anzahl der Zeichen
        SUI     '0'             ;binär
        JZ      DONE            ;Ende bei null
        MOV     C,A
        PCHAR   BLANK
LOOP:
        PCHAR   '*'
        DCR     C
        JNZ     LOOP
        JMP     NEXT
DONE:
        EXIT
;
        END     START
```

Abb. 5.5: Ein Programm zum Testen der Makros SYSF, READCH, PCHAR und CRLF (Forts.)

Sobald ein Zeichen über die Tastatur eingegeben wird, läuft das Programm weiter. Eine ASCII-Null wird von der Benutzereingabe subtrahiert. Hierdurch wird eine ASCII-Ziffer (0 bis 9) in die entsprechende Binärzahl umgewandelt. Jedes andere Zeichen wird natürlich ebenso behandelt. Wird eine Null eingegeben, beendet sich das Programm; sonst wird der Wert in das C-Register geladen. Ein Leerzeichen und eine der Eingabe entsprechende Anzahl Sterne werden ausgegeben, und das Programm startet von neuem.

Eine Eingabe von 1 bis 9 produziert ebenso viele Sterne. Das große A ergibt 17, das kleine a 49 Sterne. Zeichen wie das Dollar- oder das Pro-

zentzeichen haben ASCII-Werte kleiner als die der Ziffern und produzieren, da sie durch Subtraktion von ASCII-Null verändert wurden, mehrere Zeilen Sterne.

AUSGABE EINES TEXTES

Im vorigen Abschnitt haben wir die BDOS-Funktion 2 benutzt, um einzelne Zeichen nacheinander auf der Konsole auszugeben. Häufig will man jedoch mehrere Zeichen ausgeben wie z.B.

```
?DATEI NICHT GEFUNDEN
```

Dies kann man leicht mit der BDOS-Funktion 9 durchführen. Der Text ist dann im Speicher, beendet mit einem Dollarzeichen. Die Anfangsadresse des Textes wird in das DE-Register geladen, und der Wert 9 in das Register C. Durch den Aufruf von BDOS wird dann der Text auf dem Bildschirm ausgegeben. Das Dollarzeichen wird natürlich nicht angezeigt.

Das Programm in Abb. 5.6 demonstriert die Anwendung der BDOS-Funktion 9 für die Ausgabe eines Textes auf die Konsole. Das Programm benutzt das Makro SYSF. Tippen Sie das Programm ein, assemblieren und starten Sie es. Die Konsol-Ausgabe wird dann

```
Ein Test der BDOS-Funktion 9
```

sein. In diesem Programm beginnt der Text mit den Zeichen Return und Line-Feed. Diese beiden Zeichen sind hier Bestandteil des Textes. Weiter oben hatten wir dafür das Makro CRLF benutzt.

Der übrige Text einschließlich dem Dollarzeichen wird in Apostrophe eingeschlossen. Der Assembler setzt den Text im Speicher direkt hinter das Makro SYSF, das ein Unterprogramm darstellt. Die Anweisung JMP DONE überspringt sowohl die Subroutine als auch den Text.

EIN MAKRO FÜR DIE AUSGABE EINES TEXTES

Die Ausgabe eines Textes ist mit der Funktion 9 bequemer als die zeichenweise Ausgabe mit der Funktion 2. Dennoch müssen wir einen Übersprung über den Text vorsehen und die Subroutine SEND aufrufen. In diesem Abschnitt werden wir ein neues Makro schreiben, um die Ausgabe eines Textes weiter zu vereinfachen. Unser Ziel ist das Makro PRINT. Seine einfache Anwendung zeigt die Anweisung:

```
PRINT 'Ein Test der BDOS-Funktion 9'
```

```
TITLE   'Drucke Konsolpuffer'
;
;(Tagesdatum)
;
BOOT    EQU     0               ;Warmstart
BDOS    EQU     5               ;BDOS-Eingang
TPA     EQU     100H            ;Anwenderbereich
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum).CONSOLE BUFFER'
        LXI     D,TEXT
        CALL    SEND
        JMP     DONE
SEND:
        SYSF    9
TEXT:
        DB      CR,LF,'Ein Test der'
DONE:           ' BDOS-Function 9$'
        EXIT    BOOT            ;Warmstart
;
        END     START
```

Abb. 5.6: Ausgabe des Konsolpuffers

D.h. der Parameter des Makros ist der in Apostrophe eingeschlossene Text.

Machen Sie eine Kopie des Programms in Abb. 5.6, und ändern Sie es nach Abb. 5.8 ab. Wir benutzen es für den Test des Makros PRINT in Abb. 5.7. Sie können das Makro schon jetzt in Ihre Bibliothek eintragen, allerdings werden wir noch in diesem Abschnitt eine allgemeinere Fassung schreiben. Deshalb fügen Sie diese Fassung besser in das Programm nach Abb. 5.8 ein, und zwar direkt hinter die Anweisung MACLIB CPMMAC.

Wenn der Assembler das Makro PRINT findet, setzt er den Text begin-

```
PRINT   MACRO   TEXT
;;(Tagesdatum)
;;Inline-Makro für die Ausgabe eines Textes
;;Makro benötigt: SYSF
;;
;;Anwendung:  PRINT   'Meldung'
;;          PRINT   <CR,LF,'Meldung'>
;;
            LOCAL   MESG,AROUND
            PUSH    D
            LXI     D,MESG
            CALL    PBUF?           ;Text ausgeben
            POP     D
            JMP     AROUND
            IF      NOT PRFLAG      ;Subroutine benötigt
;
;Textausgabe auf die Konsole bis $
;
PBUF?:
            SYSF    9
PRFLAG      SET     TRUE            ;nur eine Kopie
            ENDIF
;
MESG:       DB      TEXT,'$'
AROUND:
                                    ;;PRINT
            ENDM
```

Abb. 5.7: Das Makro PRINT, Version 1

nend bei der Marke MESG im Speicher ab. Ein Dollarzeichen wird automatisch an den Text angehängt, um CP/M das Ende des Textes anzuzeigen. Das DE-Register wird mit PUSH in den Stack gerettet und dann mit der Adresse MESG, dem Beginn des Textes, geladen. Durch den Aufruf des Unterprogramms PBUF? wird der Text ausgegeben. Das DE-Register wird mit POP wieder geladen; ein Sprung über die Subroutine und den Text schließt das Makro PRINT ab.

Mehrere Eigenheiten sollten Sie bei diesem Beispiel beachten. Das Symbol PRFLAG wird zunächst auf FALSE gesetzt, damit das Unterprogramm PBUF? nur einmal generiert wird. PBUF? ist eine globale Varia-

ble, die Marken MESG und AROUND sind dagegen lokale. Sie erscheinen bei jedem Makroaufruf, haben jedoch dann verschiedene Symbole. Den Parameter des PRINT-Makros haben wir in spitze Klammern eingeschlossen:

```
<CR,LF,'Ein Test der BDOS-Funktion 9'>
```

Hierdurch sagen wir dem Assembler, daß Return und Line-Feed in den Text eingeschlossen werden sollen.

Assemblieren Sie das neue Programm, und starten Sie es. Die Ausgabe sollte so sein wie oben.

```
TITLE   'Drucke Konsolpuffer'
;
;(Tagesdatum)
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
BOOT    EQU     0               ;Warmstart
BDOS    EQU     5               ;BDOS-Eingang
TPA     EQU     100H            ;Anwenderbereich
;
PRFLAG  SET     FALSE           ;Drucke Konsolpuffer
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum)'
        PRINT   <CR,LF,'Ein Test der BDOS-Funktion 9>
DONE:
        EXIT    BOOT            ;Warmstart
;
                END             START
```

Abb. 5.8: Ein Programm zum Testen des Makros PRINT, Version 1

Das Makro PRINT, Version 2

Mit dem Makro, das wir im vorigen Abschnitt geschrieben haben, können wir in das Programm eingebettete Texte ausgeben. Ein Dollarzeichen kann man so nicht ausgeben. Manchmal wollen wir auch Texte ausgeben, die ab einer bestimmten Adresse im Speicher stehen. Vielleicht kennen wir diese Adresse noch nicht einmal, bevor wir das Programm starten. Das vorige Makro könnten wir für diesen Zweck anpassen, wenn wir ein Dollarzeichen an das Ende des Textes stellen, doch scheint dieser Weg nicht sehr bequem zu sein. Wir werden nun das Makro PRINT so umschreiben, daß wir damit einen Text ausgeben können, der irgendwo im Speicher steht oder als Parameter im Makroaufruf.

Wir entfernen die BDOS-Funktion 9, die einen Text ausgibt, und benutzen dafür die Funktion 2. Wir werden die Zeichen einzeln mit dem Makro PCHAR ausgeben. Das Makro PRINT berechnet die Textlänge und damit die Anzahl der Aufrufe der im Makro PCHAR erzeugten Subroutine. Dies scheint ein Schritt zurück zu sein, ist es jedoch nicht. Mit dieser Version können wir Texte ab einer beliebigen Adresse ausgeben; sie können sogar Dollarzeichen enthalten.

Tragen Sie diese zweite Version des Makros PRINT in die Makrobibliothek ein. Ändern Sie das Testprogramm in Abb. 5.8 nach Abb. 5.10 ab, und nennen Sie es PRN2. Beachten Sie, daß zwei Flags, COFLAG und PRFLAG, benötigt werden. Beachten Sie auch, daß in diesem Beispiel keine regulären 8080-Befehle erscheinen, sondern nur Makroaufrufe. Assemblieren und starten Sie das Programm. Geben Sie das CP/M-Kommando

```
PRN2 TEST VON PRINT
```

ein. Das Programm wird mit den beiden Zeilen

```
Die ersten 12 Zeichen des Command-Tail sind:
TEST VON PRI
```

antworten. Das Programm ruft das Makro PRINT dreimal auf. Die beiden ersten Aufrufe ähneln den früheren. Der Text wird auf dem Bildschirm ausgegeben:

```
Die ersten 12 Zeichen des Command-Tail sind:
```

Der dritte Aufruf ist jedoch anders. Die Anwesenheit des zweiten Parameters im Makroaufruf sagt dem Assembler, daß der erste Parameter die Adresse eines Textes ist und nicht ein Text selbst. Der erste Parameter

bezeichnet den Anfang des Textes bei DBUFF+2. Wenn CP/M ein Programm startet, wird ab 82 hex der Rest des Kommandos (command line tail) gespeichert. Das HL-Register wird daher mit der Adresse 82 hex (DBUFF+2) geladen.

Sehen wir uns nun mit einem Programm an, wie das PRINT-Makro funktioniert.

```
PRINT   MACRO   TEXT,BYTES
;;(Tagesdatum)
;;Inline-Makro für die Ausgabe eines Textes auf die Konsole.
;;TEXT ist die Adresse des Textes, BYTES ist die Länge.
;;TEXT darf in Apostrophe eingeschlossen sein,
;;wenn BYTES fehlt.
;;benötigtes Makro: PCHAR
;;
;;Anwendung:  PRINT   FCB1+1,11
;;          PRINT   'Dateiende'
;;          PRINT   <CR,LF,'Meldung'>
;;          PRINT   ,12
;;
            LOCAL   AROUND,MESG
            PUSH    H
            PUSH    B
            IF      NUL BYTES
            LXI     H,MESG
            MVI     B,AROUND-MESG
            ELSE
            IF      NOT NUL TEXT
            LXI     H,TEXT
            ENDIF
            MVI     B,BYTES
            ENDIF
            CALL    PBUF?
            POP     B
            POP     H
            IF      NOT PRFLAG OR NUL BYTES
            JMP     AROUND
            ENDIF
            IF      NOT PRFLAG
```

Abb. 5.9: Das Makro PRINT, Version 2

```
PBUF?:      MOV     A,M
            PCHAR
            INX     H
            DCR     B
            JNZ     PBUF?
            RET
PRFLAG      SET     TRUE
            ENDIF
            IF      NUL BYTES
MESG:       DB      TEXT
            ENDIF   ;;PRINT
AROUND:
            ENDM
```

Abb. 5.9: Das Makro PRINT, Version 2 (Forts.)

```
TITLE   'Drucke Konsolpuffer'
;
;(Tagesdatum)
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
BOOT    EQU     0           ;Warmstart
BDOS    EQU     5           ;BDOS-Eingang
TPA     EQU     100H        ;Anwenderbereich
DBUFF   EQU     80H         ;Standardpuffer
;
COFLAG  SET     FALSE       ;Konsol-Ausgabe
PRFLAG  SET     FALSE       ;drucke Konsolpuffer
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
```

Abb. 5.10: Ein Programm zum Testen der Version 2 des Makros PRINT

```
        VERSN   '(Tagesdatum)'
        PRINT   <CR,LF,'Die ersten 12 Zeichen des '>
        PRINT   <'Command-Tail sind: ',CR,LF>
        PRINT   DBUFF+2,12
DONE:
        EXIT    BOOT            ;Warmstart
;
        END     START
```

Abb. 5.10: Ein Programm zum Testen der Version 2 des Makros PRINT (Forts.)

EIN PROGRAMM ZUR ERMITTLUNG DES CPU-TYPS

Der Befehlssatz des 8080 (und des 8085) ist in dem viel größeren Befehlssatz der Z80-CPU enthalten. Daher laufen 8080-Programme normalerweise auch auf einem Z80-Computer. Jedoch werden Programme, die die besonderen Eigenschaften des Z80, wie Blockspeicherbefehle und relative Sprünge, ausnutzen, nicht auf einem 8080-Computer laufen.

Wegen dieser Unterschiede kann es einmal notwendig werden, den CPU-Typ zu bestimmen. Braucht ein Programm z.B. spezielle Z80-Befehle, kann es sich beenden, wenn es auf einem 8080 gestartet wird. Gegebenenfalls kann man unterschiedliche Algorithmen verwenden. Die wirksamere Version wird benutzt, wenn ein Z80 vorliegt, andernfalls wird die 8080-Version gewählt.

Da der Z80 und der 8080 sich bei arithmetischen Operationen unterschiedlich benehmen, kann man sie leicht unterscheiden. Der Unterschied liegt in der Verwendung des Parity-Flags. Dieses Flag reflektiert bei beiden CPUs korrekt das Ergebnis der logischen Operationen. Bei arithmetischen Operationen gibt es jedoch Unterschiede. Beim 8080 wird die Parity (Parität) des Ergebnisses wie bei logischen Operationen angezeigt. Der Z80 hingegen setzt das Flag nur bei einem arithmetischen Überlauf (von Bit 6 nach Bit 7). Daher heißt das Parity-Flag beim Z80 Parity/Overflow-Flag (overflow = Überlauf, parity = Parität, Gleichheit).

Mit den folgenden Befehlen können wir zwischen einem 8080 und einem Z80 unterscheiden:

```
XRA  A
DCR  A
JPE  NOTZ80
```

Der erste Befehl führt ein Exklusiv-Oder des A-Registers mit sich selbst aus. Dadurch wird das A-Register gelöscht. Ebenfalls setzt er das Parity-Flag (d.h. gerade Parität) bei beiden CPUs, da das Ergebnis eine gerade Anzahl von Einsen hat (nämlich Null).

Der nächste Befehl vermindert das A-Register um 1 zu FF hex. Diese arithmetische Operation läßt das 8080-Parity-Flag gesetzt, da die Anzahl der Einsen gerade geblieben ist (nämlich acht). Das Z80-Parity/Overflow-Flag wird gelöscht, da durch die Operation kein Überlauf entstanden ist. Ein 8080 wird also bei dem JPE-Befehl springen, ein Z80 hingegen nicht, da das Flag gelöscht ist.

Die obigen drei Zeilen kann man in ein Programm, das nur auf einem Z80 laufen darf, einfügen, um zu verhindern, daß es auf einem 8080 läuft. Lassen Sie uns mit einem kleinen Assemblerprogramm zeigen, wie dies funktioniert.

Das Programm in Abb. 5.11 gibt

```
CPU ist Z80
```

aus, wenn es auf einem Z80-Computer läuft. Sonst wird

```
CPU ist 8080
```

ausgegeben. Erzeugen Sie eine Datei namens CPU. Geben Sie das Programm ein, assemblieren und starten Sie es.

Das Programm beginnt wie gewöhnlich mit ENTER und VERSN. Das PRINT-Makro gibt den Anfang des Textes aus. Die nächsten drei Zeilen bestimmen den CPU-Typ. Ist es ein 8080, setzt das Programm bei NOTZ80 fort und gibt den Text „8080" aus; sonst fährt das Programm fort und gibt den Text „Z80" aus.

```
TITLE   'CPU prüft, ob 8080 oder Z80'
;
;(Tagesdatum)
;
;Anwendung: CPU
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
```

Abb. 5.11: Das Programm CPU für die Prüfung, ob die CPU ein 8080 oder Z80 ist

```
BOOT    EQU     0               ;Warmstart
BDOS    EQU     5               ;BDOS-Eingang
FCB1    EQU     5CH             ;Eingabe FCB
FCB2    EQU     6CH             ;2. Parameter
DBUFF   EQU     80H             ;Standardpuffer
TPA     EQU     100H            ;Anwenderbereich
;
;Setze Flags im Hauptprogramm, damit nur eine
;Kopie einiger Subroutines generiert wird.
;SET-Zeilen müssen vor dem MACLIB-Call stehen.
;
COFLAG  SET     FALSE           ;Konsole-Ausgabe
PRFLAG  SET     FALSE           ;drucke Konsolpuffer
;Ende der Flags
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum).CPU'
        PRINT   'CPU ist'
        XRA     A               ;null
        DCR     A
        JPE     NOTZ80
        PRINT   'Z80'
        JMP     DONE
NOTZ80:
        PRINT   '8080'
DONE:
        EXIT
;
        END     START
```

Abb. 5.11: Das Programm CPU für die Prüfung, ob die CPU ein 8080 oder Z80 ist (Forts.)

Bevor wir unser nächstes Programm schreiben, müssen wir noch zwei Makros in die Bibliothek eintragen. Das erste Makro konvertiert binäre Zahlen in hexadezimale und gibt sie aus. Das zweite bestimmt die Versionsnummer des CP/M.

EIN MAKRO ZUR KONVERTIERUNG VON BINÄR- IN HEXADEZIMALZAHLEN

Eine Binär-Hexadezimal-Konvertierung wird in vielen Programmen dieses Buches benutzt. Jeder 8-Bit-Wert kann mit zwei Hexadezimalziffern ausgedrückt werden; die Hexadezimalzahl liegt dann im Bereich von 0 bis FF hex.

Wie Sie wissen, werden Informationen in einem Computer als Folge von binären Ziffern (0 oder 1) gespeichert. Jede Ziffer heißt Bit, eine Gruppe von acht Bits heißt Byte. Manchmal wollen wir den Wert eines Bytes wissen, können aber dieses Byte nicht einfach zur Konsole schicken, da die Konsole den ASCII-Code, einen 7-Bit-Code, benutzt. Die Zahl

```
0100 1011
```

z.B. heißt hexadezimal 4B. Dieser Wert entspricht aber dem Buchstaben K. Zur Konsole geschickt würden wir diesen Buchstaben sehen. Wir brauchen also eine Routine, die die ASCII-Ziffer 4 und den ASCII-Buchstaben B ausgibt. Dies nennt man Binär-Hexadezimal-Konvertierung.

Beachten Sie bei der obigen Binärzahl, daß die oberen vier Bits dem linken Hex-Zeichen (4), die unteren vier Bits dem rechten Hex-Zeichen (B) entsprechen:

Nibble	*ASCII-Muster*	*Zeichen*
0100	00110100	4
1011	01000010	B

Wir müssen also das linke Zeichen vor dem rechten ausgeben. Daher müssen wir die oberen vier Bits in die unteren vier schieben. Die neuen oberen Bits werden dann gelöscht. Aus dem Muster

```
0100 1101
```

wird

```
1101 0100
```

und dann

```
0000 0100
```

Vor der Rotation und dem Löschen müssen wir natürlich das Originalzeichen retten, sonst verlieren wir das rechte Nibble.

```
OUTHEX   MACRO   REG
;;(Tagesdatum)
;;Inline-Makro, um eine Binärzahl in
;;REG in zwei Hex-Zeichen umzuwandeln und zu drucken.
;;Byte wird in A erwartet, wenn REG fehlt.
;;benötigtes Makro: PCHAR
;;
            LOCAL   AROUND,HEX1?,HEX2?
            IF      NOT NUL REG
            MOV     A,REG
            ENDIF
            CALL    OUTHX?
            IF      NOT CXFLAG
            JMP     AROUND
OUTHX?:     PUSH    B
            MOV     C,A             ;retten
            RAR
            RAR
            RAR
            RAR
            CALL    HEX1?           ;oberes Byte
            MOV     A,C
            CALL    HEX1?           ;unteres Byte
            MOV     A,C             ;wieder laden
            POP     B
            RET
HEX1?:      ANI     0FH             ;Ausgabe hex Byte
            ADI     '0'             ;mache ASCII
            CPI     '9'+1           ;0--9?
            JC      HEX2?           ;ja
            ADI     'A'--'9'--1     ;mache A--F
HEX2?:
            PCHAR                   ;zur Konsole
            RET
CXFLAG      SET     TRUE
            ENDIF
AROUND:                             ;;OUTHEX
            ENDM
```

Abb. 5.12: Das Makro OUTHEX für die hexadezimale Anzeige eines binären Bytes

Tragen Sie das Makro OUTHEX nach Abb. 5.12 in die Bibliothek ein. Mit ihm werden wir eine Binärzahl in zwei Hex-Ziffern konvertieren, die dann ausgegeben werden. Läßt man den wahlfreien Parameter fort, wird die Zahl im A-Register konvertiert. Steht die Binärzahl in einem anderen Register oder im Speicher, gibt der Parameter dies an.

Für die Konvertierung eines Vier-Bit-Nibbles in ein ASCII-Zeichen gibt es zwei Wege. Das Grundproblem ist, Binärzahlen von 0-15 in die ASCII-Ziffern 0-9 und die ASCII-Buchstaben A-F umzuwandeln. Wir müssen die Binärzahlen in ihre ASCII-Äquivalente in hexadezimaler Schreibweise, d.h. zur Basis 16, konvertieren.

Studieren wir die Bitmuster der ersten zehn Zahlen:

Dezimal	*Binär*	*ASCII*	*Hex*
0	0000	00110000	0
1	0001	00110001	1
2	0010	00110010	2
3	0011	00110011	3
4	0100	00110100	4
5	0101	00110101	5
6	0110	00110110	6
7	0111	00110111	7
8	1000	00111000	8
9	1001	00111001	9

In der obigen Liste erkennen Sie die feste Differenz zwischen der Binärzahl und ihrem ASCII-Gegenstück. Die ASCII-Null hat einen Wert von 30 hex und den Binärwert 0. Die Differenz von 30 hex nennt man ASCII-Bias. Konvertiert man also eine Binärzahl von 0-9 in eine Hex-Ziffer, braucht man nur den Bias von 30 hex zu addieren. Steht die Zahl im A-Register, tut dies der Befehl

```
ADI '0'
```

Hat das Nibble aber einen Wert größer als 9, ist die Konvertierung anders. Die Bitmuster für diese Zahlen sind:

Dezimal	*Binär*	*ASCII*	*Hex*
10	1010	01000001	A
11	1011	01000010	B
12	1100	01000011	C
13	1101	01000100	D
14	1110	01000101	E
15	1111	01000110	F

Bei näherer Betrachtung finden wir zwischen dem Binärwert und dem ASCII-Wert eine Differenz von 37 hex. Wir können bei dieser Gruppe also die Konvertierung durch Addition von 37 hex durchführen.

Die Konvertierung führen wir durch, indem wir zunächst den ASCII-Bias von 30 hex addieren. Dafür benutzen wir den Befehl ADI '0'. Wenn das Originalnibble im Bereich 0-9 war, ist das Ergebnis nun im Bereich ASCII 0-9. Ist das Originalnibble jedoch im Bereich 10-15, dann addieren wir zusätzlich 7. Dadurch erhalten wir die ASCII-Zeichen A-F. Dieser zusätzliche Summand ist um eins kleiner als die Differenz zwischen ASCII A und ASCII 9. Wir benutzen also den Befehl

```
ADI     'A'-'9'-1       ; ergibt A-F
```

Der Assembler rechnet für diesen Operanden den Wert 7 aus, und so wird aus einer binären 2 die ASCII-Ziffer 2, aber aus einer binären 10 wird der ASCII-Buchstabe A.

Schneller und einfacher ist ein anderer Algorithmus, er ist jedoch schwerer zu durchschauen. Die Befehle von ADI '0' bis ADI 'A'-'9'-1 werden ersetzt durch:

```
ADI     90H
DAA
ACI     40H
DAA
```

Hierbei wird der Dezimalbefehl DAA (Decimal Adjust Accumulator) verwendet. Der DAA-Befehl ist für die BCD-Arithmetik vorgesehen. Nach jeder Addition wird DAA ausgeführt. Der Befehl addiert 6 auf jedes Nibble, wenn sein Wert größer als 9 ist.

Betrachten wir diese Methode für die Konvertierung einer binären 2 und einer binären 10 (hexadezimal A):

	Binäre Zwei	*Binäre Zehn*
Originalwert	0000 0010	0000 1010
90 hex	1001 0000	1001 0000
ADI	1001 0010	1001 1010
DAA	1001 0010	0000 0000
40 hex	0100 0000	0100 0000
ACI	1101 0010	0100 0001
DAA	0011 0010	0100 0001
ASCII-Wert	2	A

Bei der binären 2 ändert der erste DAA-Befehl nicht das Ergebnis. Der zweite DAA-Befehl ändert die 1101 des linken Nibble durch Addition von 6 in 0011. Das Ergebnis ist 32 hex, die ASCII-Ziffer 2. Bei der binären Zehn ändert der erste DAA-Befehl 10011010 in 00000000 und setzt das Carry-Flag. Der zweite DAA-Befehl ist ohne Wirkung. Das Ergebnis ist 41 hex, der ASCII-Buchstabe A.

Wir entwickeln nun ein Makro zur Bestimmung der Versionsnummer von CP/M.

EIN MAKRO ZUR ERMITTLUNG DER CP/M-VERSIONSNUMMER

Das ursprüngliche CP/M hatte die Versionsnummer 1.3. Die nachfolgenden Versionen waren 1.4, 2.0, 2.1, 2.2 etc. Viele CP/M-Programme laufen mit allen Versionen. Einige wirksame Funktionen gibt es erst seit der Version 2, und Programme, die davon Gebrauch machen, können mit Version 1 nicht ausgeführt werden. Auch wir werden in Kapitel 8 ein Programm schreiben, das die integrierten Disketten-Parametertabellen benutzt und daher nicht in Version 1 läuft. Programme, die Funktionen der Version 2 benötigen, sollten die Versionsnummer des CP/M bestimmen, unter welchem sie laufen, und sich beenden, wenn sie kleiner als 2 ist.

```
CPMVER   MACRO
;;(Tagesdatum)
;;Inline-Makro zur Bestimmung der CP/M Version.
;;Akkumulator enthält Version in BCD mal 10.
;;A = 22 für Version 2.2, A = 0 für Ver 1.4.
;;
         PUSH    H
         PUSH    D
         PUSH    B
         MVI     C,12
         CALL    BDOS
         MOV     A,L          ;;nicht nötig
         POP     B
         POP     D
         POP     H
                              ;CPMVER
         ENDM
```

Abb. 5.13: Das Makro CPMVER zur Bestimmung der CP/M-Versionsnummer

Die CP/M-Versionsnummer erhält man mit der BDOS-Funktion 12. Bei Version 2 und danach wird diese Nummer mit 10 multipliziert und sowohl im A- als auch im L-Register als gepackte BCD-Zahl übergeben. Version 2.2 z.B. wird angezeigt durch 22 hex. Versionen kleiner als 2 liefern eine Null.

Mit dem Makro CPMVER kann man herausfinden, ob Version 2 oder höher benutzt wird. Das Makro nach Abb. 5.13 tragen Sie in Ihre Makrobibliothek ein.

EIN PROGRAMM FÜR DIE ANZEIGE DES IOBYTE

In Kapitel 3 haben wir gelernt, wie man die Zuordnung der vier logischen Geräte - Konsole, Leser, Stanzer und Drucker - ändern kann. Außerdem haben wir die IOBYTE-Funktion in mehrere BIOS-Routinen eingefügt. So können wir die Konsol-Ausgabe zum Drucker schicken, indem wir das IOBYTE auf 1 setzen.

Wir haben gelernt, daß man das IOBYTE mit dem Debugger, mit STAT oder mit BASIC ändern kann. Bequemer wäre jedoch ein Programm, mit dem man das IOBYTE ändern und anzeigen kann. Dieses Programm werden wir in zwei Schritten entwickeln.

Wir beginnen mit einem Programm, das den momentanen Wert des IOBYTE liest und hexadezimal ausgibt. Zusätzlich wird dieses Programm die CP/M-Versionsnummer anzeigen. Dazu benötigen wir mehrere der bereits geschriebenen Makros.

Machen Sie eine Kopie des Programms in Abb. 5.14, und nennen Sie sie IOBYTE. Assemblieren und starten Sie es. Das Programm gibt hexadezimal den momentanen Wert des IOBYTE und die CP/M-Versionsnummer aus. Das Programm erhält die Versionsnummer mit dem Makro CPMVER als gepackte BCD-Zahl. Für die Ausgabe benutzen wir das Makro OUTHEX, unseren Binär-Hexadezimal-Konverter.

Der Originalwert wird im C-Register gerettet. Das Makro UPPER schiebt die oberen vier Bits in die unteren und löscht die oberen vier Bits. Ein logisches OR mit dem Operanden '0' konvertiert die Binärzahl in eine ASCII-Ziffer, die mit PCHAR ausgegeben werden kann. Ein Dezimalpunkt wird mit PCHAR ausgegeben. Danach wird der Originalwert aus dem C-Register geladen, und die vier unteren Bits werden ebenso nach ASCII konvertiert und ausgegeben.

Bevor wir das IOBYTE-Programm fertigstellen, müssen wir zwei weitere Makros hinzufügen.

```
TITLE   'Anzeige von IOBYTE und Version'
;gibt CP/M-Version auch aus
;
;(Tagesdatum)
;
;Anwendung: IOBYTE
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
IOBYTE  EQU     3
BOOT    EQU     0               ;Warmstart
BDOS    EQU     5               ;BDOS-Eingang
FCB1    EQU     5CH             ;Eingabe FCB
TPA     EQU     100H            ;Anwenderbereich
;
;Setze Flags im Hauptprogramm, damit nur eine
;Kopie einiger Subroutines generiert wird.
;SET-Zeilen müssen vor dem MACLIB-Call stehen.
;
COFLAG  SET     FALSE           ;Konsole Ausgabe
CXFLAG  SET     FALSE           ;binär nach hex
HXFLAG  SET     FALSE           ;hex nach binär in HL
PRFLAG  SET     FALSE           ;drucke Konsolpuffer
RCFLAG  SET     FALSE           ;lies Konsolpuffer
;Ende der Flags
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum).IOBYTE1'
        PRINT   'IOBYTE ist'
        MVI     C,7             ;hole IOBYTE
        CALL    BDOS
        OUTHEX
        PRINT   'hex'
```

Abb. 5.14: Ein Programm zur Anzeige des IOBYTE-Wertes und der CP/M-Versionsnummer

```
DONE:
        PRINT       ' für CP/M version '
        CPMVER
        MOV     C,A             ;retten
        UPPER                   ;obere Hälfte
        ORI     '0'             ;konvertiere nach dezimal
        PCHAR                   ;linke Ziffer
        PCHAR   PERIOD
        MOV     A,C
        ANI     0FH
        ORI     '0'             ;konvertiere nach dezimal
        PCHAR                   ;rechte Ziffer
        EXIT
;
        END     START
```

Abb. 5.14: Ein Programm zur Anzeige des IOBYTE-Wertes und der CP/M-Versionsnummer (Forts.)

Ein Makro zum Lesen des Konsolpuffers

Weiter vorn in diesem Kapitel haben wir zwei Arten von Ausgaberoutinen betrachtet. Die eine benutzt die BDOS-Funktion 2 und gibt einzelne Zeichen aus. Die zweite benutzt die BDOS-Funktion 9 und gibt einen kompletten Text aus. Analog dazu kann man Zeichen von der Konsole einzeln mit der BDOS-Funktion 1 oder zeilenweise mit der BDOS-Funktion 10 einlesen.

Das Makro READCH benutzt man zum Lesen einzelner Zeichen. Wir werden nun ein Makro entwickeln, mit dem man die Zeichen einer ganzen Zeile von der Konsole einliest.

Liest man Zeichen von der Konsole mit der Funktion 10, werden sie in einem Speicherbereich abgelegt, den man Konsolpuffer nennt. Dieser Bereich muß vor dem BDOS-Aufruf eingerichtet werden. Zwei weitere Bytes unmittelbar vor diesem Puffer gehören zu diesem Puffer. Das erste dieser beiden definiert die Pufferlänge, d.h. die maximale Anzahl von Zeichen, die der Puffer aufnehmen kann. Das zweite Byte zeigt die tatsächliche Anzahl der eingegebenen Zeichen in diesem Puffer.

Für die BDOS-Funktion 10 lädt man die Adresse des ersten Hilfsbytes in das DE-Register, den Wert 10 in das C-Register und ruft dann BDOS auf.

Jedes eingegebene Zeichen speichert CP/M in den Puffer und schreibt es auf den Bildschirm. Die Funktion wird beendet, wenn Return eingegeben wird oder wenn die Anzahl der eingegebenen Zeichen gleich der Zahl im ersten Hilfsbyte ist. CP/M setzt die Anzahl der gelesenen Zeichen in das zweite Hilfsbyte. Folgende CP/M-Control-Zeichen können bei dieser Eingabemethode verwendet werden:

Zeichen	*Bedeutung*
E	Neue Zeile beginnen
H	Backspace
I	Tabulator
P	An/Abschalten des Druckers
R	Zeile neu anzeigen
U	Zeile abbrechen
X	Zeile abbrechen

Wir entwickeln nun das Makro READB in Abb. 5.15 zum Lesen des Konsolpuffers. Die Befehle am Beginn des Makros zeigen den BDOS-Aufruf zum Füllen des Konsolpuffers. Der Pufferbereich selbst steht am Ende des Makros und hat den Namen RBUF. Die Hilfsbytes heißen RBUFM und RBUFC.

```
RBUFM:  DB      RBUFE-RBUF      ;Maximale Anzahl
RBUFC:  DS      1               ;Aktuelle Anzahl
RBUF:   DS      16              ;Pufferbeginn
RBUFE:                          ;Pufferende
```

Das DE-Register wird mit der Adresse RBUFM geladen, der Adresse der maximalen Pufferlänge. Beachten Sie, daß der Assembler diese Länge durch Subtraktion der Pufferanfangsadresse (RBUF) von der Pufferendadresse (RBUFE) berechnet. Nach Beendigung der Operation und Rückkehr zum aufrufenden Programm enthält der Speicher RBUFC die Anzahl der tatsächlich eingelesenen Zeichen.

Wenn eine Zeile von Zeichen im Konsolpuffer mit der BDOS-Funktion 10 bereitgestellt wurde, muß man die Zeichen aus dem Puffer abholen. Der Mittelteil des Makros READB enthält für diesen Zweck eine separate globale Subroutine namens GETCH. Bei jedem Aufruf dieser Subroutine wird das nächste Zeichen in das A-Register geladen.

Nachdem die Subroutine GETCH dem Puffer ein Zeichen entnommen hat, wird die Zahl der noch verbleibenden Zeichen (in RBUFC) vermindert. Der Pufferzeiger RBUFP wird zunächst auf das erste Zeichen eingestellt. Nach der Entnahme des Zeichens wird er um eins erhöht.

Wenn man mit GETCH ein gültiges Zeichen erhalten hat, ist das Carry-Flag gelöscht. Sind im Puffer jedoch keine Zeichen mehr verfügbar, wird das Carry-Flag gesetzt. Daher muß man dieses Flag nach jedem Aufruf der Subroutine GETCH prüfen. Der Pufferpointer ist ein globales Symbol; man kann auf ihn also von jeder Stelle des Programms aus zugreifen. Tragen Sie das Makro READB in Ihre Makrobibliothek ein.

```
READB   MACRO
;;(Tagesdatum)
;;Inline-Makro für die Eingabe einer Zeile von der Konsole.
;;Puffer liegt am Ende des Makros.
;;Die Zeichen werden mit der globalen Subroutine
;;GETCH aus dem Puffer gelesen.
;;Puffer-Zeiger RBUFP ist auch global.
;;
            LOCAL   AROUND,RBUFM,RBUF,RBUFC,RBUFE
            CALL    RDB2?
            IF      NOT RCFLAG
            JMP     AROUND
RDB2?:
            PUSH    H
            PUSH    D
            PUSH    B
            LXI     D,RBUFM
            MVI     C,10
            CALL    BDOS
            LXI     H,RBUFM+2
            SHLD    RBUFM-2
            POP     B
            POP     D
            POP     H
            RET
;globale Routine zum Lesen eines Zeichens aus dem Puffer
GETCH:
            LDA     RBUFC       ;Zeichenzahl
            SUI     1           ;minus 1
            RC                  ;Puffer leer
            STA     RBUFC
            PUSH    H
```

Abb. 5.15: Das Makro READB zum Lesen des Konsolpuffers

```
            LHLD    RBUFP
            MOV     A,M             ;lies Zeichen
            INX     H               ;erhöhen
            SHLD    RBUFP
            POP     H
            RET
;
RCFLAG      SET     TRUE            ;nur eine Kopie
;
RBUFP:      DW      RBUF            ;Pufferzeiger
;Konsolpufferadresse
RBUFM:      DB      RBUFE-RBUF      ;max. Länge
RBUFC:      DS      1               ;aktuelle Länge
RBUF:       DS      16              ;Pufferanfang
RBUFE:                              ;Pufferende
            ENDIF
AROUND:                             ;;READB
            ENDM
```

Abb. 5.15: Das Makro READB zum Lesen des Konsolpuffers (Forts.)

Ein Makro zur Konvertierung Hexadezimal nach Binär

Weiter vorn in diesem Kapitel haben wir ein Makro betrachtet, das eine binäre Zahl in eine hexadezimale verwandelt; hier werden wir das gegenteilige Programm untersuchen, das hexadezimale in binäre Zahlen umwandelt. Das Makro HEXHL in Abb. 5.16 liest von der Konsole

```
HEXHL   MACRO
;;(Tagesdatum)
;;Inline-Makro zum Konvertieren von ASCII-Hex-Zeichen
;;im Puffer in eine 16-bit Binärzahl in HL.
;;Zeichenfolge ist adressiert durch POINTR.
;;Carry-Flag gesetzt bei ungültigen Hex-Zeichen.
;;benötigte Makros: READB, UCASE
;;
```

Abb. 5.16: Das Makro HEXHL zur Konvertierung einer Folge von ASCII-Hex-Zeichen in eine 16-Bit-Binärzahl

```
        LOCAL   AROUND, RDHL2, NIB?
        CALL    RDHL?
;;
        IF      NOT HXFLAG      ;nur eine Kopie
        JMP     AROUND
RDHL?:
        LXI     H,0             ;Start mit 0
RDHL2:
;lies Zeichen aus dem Konsolpuffer
        CALL    GETCH
        CMC
        RNC                     ;Ende der Zeile
        UCASE                   ;wandeln in Großbuchstaben
        CALL    NIB?            ;nach binär
        RC                      ;Fehler
        DAD     H               ;mal 2
        DAD     H               ;mal 4
        DAD     H               ;mal 8
        DAD     H               ;mal 16
        ORA     L               ;neues Zeichen
        MOV     L,A             ;speichern
        JMP     RDHL2           ;weiter
;
;konvertiere ASCII nach binär
;
NIB?:   SUI     '0'             ;ASCII-Grundwert
        RC                      ;<0
        CPI     'F'-'0'+1
        CMC
        RC                      ;>F
        CPI     10
        CMC
        RNC                     ;0--9
        SUI     'A'-'9'-1
        CPI     10
        RET
HXFLAG  SET     TRUE            ;nur eine Kopie
        ENDIF
AROUND:                         ;;HEXHL
        ENDM
```

Abb. 5.16: Das Makro HEXHL zur Konvertierung einer Folge von ASCII-Hex-Zeichen in eine 16-Bit-Binärzahl (Forts.)

ASCII-Zeichen ein und konvertiert sie in eine 16-Bit-Zahl im HL-Register. Die Zeichen müssen erst gelesen werden, daher muß das Makro READCH vor dem Makro HEXHL aufgerufen werden. Dieses Makro benutzt das Makro UCASE.

Das Makro HEXHL bearbeitet eine Reihe von hexadezimalen ASCII-Ziffern. Ein Zwischenraum oder das Ende des Eingabepuffers beenden diesen Vorgang. Wenn ein ungültiges Hex-Zeichen entdeckt wird, wird das Carry-Flag gesetzt. Man sollte daher dieses Flag am Ende überprüfen. Kopieren Sie das Makro HEXHL in Ihre Makrobibliothek.

IOBYTE, Version 2

Das letzte Programm, das wir geschrieben haben, diente der Anzeige des IOBYTE bei Adresse 3. In dieses Programm werden wir nun die Möglichkeit zum Ändern des IOBYTE einfügen.

Machen Sie eine Kopie des ersten IOBYTE-Programms (Abb. 5.14), und ändern Sie es nach Abb. 5.17 ab. Assemblieren und starten Sie die neue Version. Wird das Programm wie bisher ohne Parameter in der Kommandozeile gestartet, werden das IOBYTE und die CP/M-Version angezeigt. Wird andererseits eine gültige Hex-Zahl als Parameter angegeben, wird das IOBYTE auf diesen Wert gesetzt. Gibt man eine ungültige Zahl ein, wird der Wert erneut angefordert.

```
TITLE   'IOBYTE: Anzeigen oder ändern'
;zeigt auch die CP/M-Version an
;
;(Tagesdatum)
;
;Anwendung:   IOBYTE
;             IOBYTE C0
;
;führt einen Warmstart durch, um die Speicher-Pointer zu löschen
;
FALSE        EQU     0
TRUE         EQU     NOT  FALSE
;
IOBYTE       EQU     3               ;Speicheradresse
BOOT         EQU     0               ;Warmstart
```

Abb. 5.17: Das Programm IOBYTE zum Anzeigen und Ändern des IOBYTE

```
BDOS        EQU     5           ;BDOS-Eingang
FCB1        EQU     5CH         ;Eingabe FCB
FCB2        EQU     6CH         ;2. Parameter
DBUFF       EQU     80H         ;Standardpuffer
TPA         EQU     100H        ;Anwenderbereich
;
;Setze Flags im Hauptprogramm, damit nur eine
;Kopie einiger Subroutines generiert wird.
;SET-Zeilen müssen vor dem MACLIB-Call stehen.
;
COFLAG      SET     FALSE       ;Konsole-Ausgabe
CXFLAG      SET     FALSE       ;binär nach hex
HXFLAG      SET     FALSE       ;hex nach binär in HL
PRFLAG      SET     FALSE       ;drucke Konsolpuffer
RCFLAG      SET     FALSE       ;lies Konsolpuffer
;Ende der Flags
;
            MACLIB  CPMMAC
;
ORG         TPA
;
START:
            ENTER
            VERSN   '(Tagesdatum).IOBYTE '
            LXI     H,FCB1+1    ;Parameter
            MOV     A,M         ;erstes Byte
            CPI     BLANK       ;leer?
            JZ      NOPAR       ;ja
;benutze FCB als Puffer
            SHLD    RBUFP
            LDA     80H         ;Pufferlänge + 1
            DCR     A           ;ignoriere Leerzeichen
            STA     RBUFP+3     ;retten Zähler
;
AGAIN:
            HEXHL               ;hex nach binär
            JC      BADPAR      ;Eingabe-Fehler
            MOV     E,L
            MVI     C,8         ;speichere IOBYTE
            CALL    BDOS
```

Abb. 5.17: Das Programm IOBYTE zum Anzeigen und Ändern des IOBYTE (Forts.)

```
        JMP     DONE
BADPAR:
        PRINT   'Hex-Wert eingeben: '
        READB                       ;noch einmal
        JMP     AGAIN
NOPAR:
        PRINT   'IOBYTE  ist'
        MVI     C,7                 ;lies IOBYTE
        CALL    BDOS
        OUTHEX
        PRINT   'hex'
DONE:
        PRINT   'für CP/M version
        CPMVER
        MOV     C,A                 ;retten
        UPPER                       ;obere Hälfte
        ORI     'O'                 ;konvertiere nach binär
        PCHAR   PERIOD
        MOV     A,C
        ANI     0FH
        ORI     '0'                 ;konvertiere nach dezimal
        PCHAR                       ;rechte Ziffer
        EXIT    BOOT                ;Warmstart
;
        END     START
```

Abb. 5.17: Das Programm IOBYTE zum Anzeigen und Ändern des IOBYTE (Forts.)

Wenn Ihr BIOS die IOBYTE-Funktion enthält, können Sie die neue Version dieses Programms testen. (Wie man die IOBYTE-Funktion in das BIOS einbaut, ist in Kapitel 3 beschrieben.) Angenommen, das IOBYTE hat momentan den Wert 0, und der Wert 1 bewirkt, daß Konsolausgaben zum Drucker gehen. Ändern Sie das IOBYTE auf den Wert 1 mit dem Kommando

```
IOBYTE 1
```

Konsol-Ausgaben sollten nun auf dem Drucker erscheinen. Um den alten Zustand wiederherzustellen, geben Sie das Kommando

```
IOBYTE 0
```

Das Programm beginnt mit den Makros ENTER und VERSN. Dann wird geprüft, ob in der Kommandozeile ein Parameter angegeben wurde. CP/M stellt den ersten Parameter, falls vorhanden, in den ersten Dateisteuerblock (FCB1) beginnend bei 5C hex. Wenn kein Laufwerksparameter angegeben wird, wie in unserem Beispiel, wird das Byte bei 5C hex gelöscht. Der Parameter beginnt dann bei der nächsten Adresse, 5D hex. Wird kein Parameter in der Kommandozeile angegeben, steht hier, bei der Adresse 5D hex (FCB1+1), ein Leerzeichen. Das Programm gibt dann den momentanen Wert des IOBYTE und die CP/M-Versionsnummer aus.

Wenn ein Parameter in der Kommandozeile angegeben wird, ist das Byte bei FCB+1 kein Leerzeichen. Im nächsten Schritt werden dann ein oder zwei ASCII-Zeichen in eine binäre Zahl umgewandelt, und das Ergebnis wird in das IOBYTE bei Adresse 3 gespeichert. Dies wird mit dem Makro HEXHL gemacht. Beachten Sie dabei, daß dieses Makro die Zeichen dem Konsolpuffer entnimmt. Daher setzen wir den Pufferzeiger (RBUFP) auf den Anfang des Dateisteuerblocks (FCB1+1).

Außerdem müssen wir die Zeichenzahl setzen. Diese entnehmen wir dem Standardkonsolpuffer bei 80 hex. Der erste Parameter beginnt bei der Adresse 82 hex, und die Zahl der eingegebenen Zeichen steht bei Adresse 80 hex. Diese Zahl ist allerdings um 1 zu groß, da das Leerzeichen vor dem Parameter mitgezählt wird. Mit den folgenden Befehlen wird daher der Zeichenzähler geladen, vermindert und in unserem Konsolpuffer bei Adresse RBUFC gespeichert:

```
LDA     80H         ;Pufferlänge +1
DCR     A           ;Blank ignorieren
STA     RBUFP+3     ;Zahl speichern
```

Das Makro HEXHL wird nun für die Konvertierung des Parameters in eine binäre Zahl benutzt. Wenn ein ungültiges hexadezimales Zeichen auftritt, wird der Wert erneut vom Benutzer angefordert. Dafür wird dann das Makro READB benutzt. Wir werden nun das Makro HEXHL in einem anderen Programm benutzen, um an eine beliebige Adresse zu springen.

EIN PROGRAMM ZUM VERZWEIGEN ZU EINER BELIEBIGEN SPEICHERADRESSE

Unter CP/M wird ein Programm gestartet, indem man seinen Namen und etwaige Parameter eingibt. CP/M kopiert dann die Datei von der Diskette in den Speicher beginnend bei der TPA (Adresse 100 hex). CP/M springt

dann zu der Adresse 100 hex, um das Programm zu starten. Manchmal möchte man jedoch zu einer anderen Adresse springen, z.B. zu einem Urlader oder einem Monitor bei einer hohen Adresse. Angenommen, wir haben zwei verschiedene BIOS-Versionen, jede auf einer anderen Systemdiskette. Wir könnten von einem System zum anderen wechseln durch einfaches Austauschen der Disketten und anschließenden Sprung zum Urlader. In diesem Fall könnten wir den DDT oder SID mit dem Debuggerkommando G für diesen Sprung zu der gewünschten Adresse verwenden.

```
TITLE   'GO zu einer beliebigen Adresse'
;
;(Tagesdatum)
;
;Anwendung:  GO        (Adresse)
;           GO
;           *(Adresse)
;
;Makro-Bibliothek für CP/M System-Calls
;
FALSE       EQU     0
TRUE        EQU     NOT FALSE
;
BOOT        EQU     0           ;Warmstart
BDOS        EQU     5           ;BDOS-Eingang
FCB1        EQU     5CH         ;Eingabe FCB
FCB2        EQU     6CH         ;2. Parameter
DBUFF       EQU     80H         ;Standardpuffer
TPA         EQU     100H        ;Anwenderbereich
;
;Setze Flags im Hauptprogramm, damit nur eine
;Kopie einiger Subroutines generiert wird.
;SET-Zeilen müssen vor dem MACLIB-Call stehen.
;
COFLAG SET FALSE ;Konsole-Ausgabe
HXFLAG SET FALSE ;hex nach binär in HL
PRFLAG SET FALSE ;drucke Konsolpuffer
RCFLAG SET FALSE ;lies Konsolpuffer
;Ende der Flags
;
```

Abb. 5.18: Das Programm GO, um zu einer beliebigen Adresse zu springen

```
                MACLIB   CPMMAC
;
ORG             TPA
;
START:
                VERSN    '(Tagesdatum).GO '
                LXI      H,FCB1+1       ;Parameter
                MOV      A,M            ;erstes Byte
                CPI      BLANK          ;leer?
                JZ       NOPAR          ;ja
;benutze FCB als Puffer
                SHLD     RBUFP          ;Pointer retten
                LDA      80H            ;Konsolpuffer-Länge + 1
                DCR      A              ;ignoriere Leerzeichen
                STA      RBUFP+3        ;Zähler retten
AGAIN:
                HEXHL                   ;hex nach binär
                JC       NOPAR          ;Eingabe-Fehler
                PCHL                    ;Sprung zu Adresse
;
;ungültiger Parameter, noch einmal
;
NOPAR:
                PRINT    'Hex-Adresse eingeben: '
                READB                   ;Eingabe Konsolzeile
                JMP      AGAIN          ;noch einmal
;
                END      START
```

Abb. 5.18: Das Programm GO, um zu einer beliebigen Adresse zu springen (Forts.)

Alternativ hierzu kann das Programm in Abb. 5.18 benutzt werden, um zu einer beliebigen Adresse zu springen. Die gewünschte hexadezimale Adresse kann in der Kommandozeile oder nach dem Start des Programms angegeben werden. Das Kommando

```
GO E800
```

z.B. bewirkt einen Sprung zu der Adresse E800 hex.

Dies Programm ähnelt dem IOBYTE, Version 2, sehr. Die Makros

VERSN, HEXHL, READB und PRINT werden benötigt. Wir haben übrigens den übergebenen Stackpointer nicht gerettet.

Erstellen Sie die Datei GO. Geben Sie das Programm ein, assemblieren und starten Sie es. Wenn Sie einen Monitor im Speicher haben, springen Sie mit dem GO-Kommando hierhin. Aber auch wenn Sie keine Adresse wissen, können Sie das Programm testen. Geben Sie das Kommando GO ohne Parameter ein. Wenn das Programm eine Adresse verlangt, geben Sie 0 ein. Dadurch führt CP/M einen Warmstart aus.

EIN PROGRAMM ZUR AUSGABE EINES SEITENVORSCHUBS AUF DEN DRUCKER

Das letzte Programm in diesem Kapitel erlaubt uns, einen oder mehrere Seitenvorschübe auf dem Drucker zu erzeugen. Für dieses Programm brauchen wir ein neues Makro; wir nennen es LCHAR. Das Makro LCHAR hat für den Drucker dieselbe Aufgabe wie PCHAR für die Konsole. Man könnte die beiden Makros auch leicht kombinieren, der Aufruf des Makros würde jedoch komplizierter werden. Daher belassen wir es bei den beiden getrennten Makros.

Tragen Sie das Makro LCHAR (Abb. 5.19) in Ihre Makrobibliothek ein. Sie tun dies am einfachsten, indem Sie das Makro PCHAR duplizieren und dann alle Zeichenfolgen „PCH" in „LCH" umändern. Dann müssen Sie noch den ersten Parameter beim Aufruf des Makros SYSF von 2 in 5 ändern.

Geben Sie das Programm nach Abb. 5.20 in eine Datei namens PAGE ein, und assemblieren Sie es. Das Programm beginnt mit den Makros ENTER und VERSN. Im Dateisteuerblock wird getestet, ob ein Parameter im Kommando angegeben wurde. Diesmal benutzen wir hierfür den Befehl

```
LDA FCB1+1
```

Steht bei dieser Adresse ein Leerzeichen, dann wurde kein Parameter angegeben, und wir geben einen Seitenvorschub auf dem Drucker aus. Wird jedoch in der Kommandozeile ein Parameter angegeben, benutzen wir ihn für die Anzahl der Seitenvorschübe. Um diese Zahl zu begrenzen, benutzen wir von der eingegebenen Zahl nur die niedrigsten drei Bits. Wir können also maximal sieben Seitenvorschübe ausgeben.

Der Hauptteil des Programms besteht aus zwei Schleifen. Die äußere zählt die Seiten, die innere die Zeilen. Mit dem Makro LCHAR werden Zeilenvorschübe zum Drucker geschickt. Dieses Programm ist sehr ein-

fach, es zeigt jedoch mehrere wichtige Eigenschaften. Im vorigen Programm z.B. haben wir bei der Adresse 5D hex (FCB1+1) einen Dateinamen als Parameter in der Kommandozeile erwartet. In diesem Programm hingegen erwarten wir als Parameter eine Dezimalzahl.

```
LCHAR       MACRO   PAR
;;(Tagesdatum)
;;Inline-Makro, um ein Zeichen zu drucken.
;;Optional kann PAR in das A-Register geladen werden.
;;benötigtes Makro: SYSF
;;
;;Anwendung:  LCHAR   '*'
;;           LCHAR   CR
;;           LCHAR
;;
            LOCAL   AROUND
            IF      NOT NUL PAR
            MVI     A,PAR
            ENDIF
            CALL    LCH2?
            IF      NOT LOFLAG
            JMP     AROUND
LCH2?:      SYSF    5, AE           ;drucke Zeichen
LOFLAG      SET     TRUE
            ENDIF
AROUND:                             ;;LCHAR
            ENDM
```

Abb. 5.19: Das Makro LCHAR zum Drucken eines Zeichens

```
TITLE   'PAGE: Seitenvorschub auf dem Drucker'
;
;(Tagesdatum)
;
;Anwendung:  PAGE
;           PAGE 3
;
```

Abb. 5.20: Das Programm PAGE für Seitenvorschübe auf dem Drucker

```
FALSE     EQU     0
TRUE      EQU     NOT FALSE
;
BDOS      EQU     5           ;BDOS Eingang
TPA       EQU     100H        ;Anwenderbereich
FCB1      EQU     5CH         ;Parameter
LPAG      EQU     66          ;Zeilen pro Seite
;
;Setze Flags im Hauptprogramm, damit nur eine
;Kopie einiger Subroutines generiert wird.
;SET-Zeilen müssen vor dem MACLIB-Call stehen.
;
LOFLAG    SET     FALSE       ;Drucker-Ausgabe
;Ende der Flags
;
          MACLIB  CPMMAC
;
ORG       TPA
;
START:
          ENTER
          VERSN   '(Tagesdatum).PAGE '
          MVI     C,1         ;für eine Seite
          LDA     FCB1+1      ;Parameter?
          CPI     BLANK
          JZ      NPAGE       ;nein
          ANI     3           ;Maximalzahl
          MOV     C,A
NPAGE:
          MVI     B,LPAG
LINES:
          LCHAR   LF
          DCR     B
          JNZ     LINES
          DCR     C           ;mehr Seiten?
          JNZ     NPAGE       ;ja
DONE:
          EXIT
;
          END     START
```

Abb. 5.20: Das Programm PAGE für Seitenvorschübe auf dem Drucker (Forts.)

ZUSAMMENFASSUNG

In Kapitel 4 haben wir unsere Makrobibliothek mit einigen allgemeinen Routinen begonnen. In diesem Kapitel haben wir Makros zur Bedienung der peripheren Geräte mit dem CP/M-BDOS hinzugefügt. Darunter sind Makros zum Schreiben von Zeichen auf die Konsole, zum Einlesen von Zeichen von der Konsole, zum Lesen und Schreiben des Konsolpuffers, sowie für Zahlenkonvertierungen. Wir haben vier Programme geschrieben, primär um mehr über die Organisation von CP/M herauszufinden. Natürlich sind diese Programme auch sonst nützlich.

Das Verzeichnis unserer Makrobibliothek sollte jetzt so aussehen:

```
;;Makros in dieser Bibliothek                    Flags
;;AMBIG    MACRO   OLD,NEW                      (keine)
;;COMPAR   MACRO   FIRST,SECOND,BYTES           CMFLAG
;;COMPRA   MACRO   FIRST,SECOND,BYTES           CMFLAG
;;CPMVER   MACRO                                (keine)
;;CRLF     MACRO                                (keine)
;;ENTER    MACRO                                (keine)
;;EXIT     MACRO   SPACE?                       (keine)
;;FILL     MACRO   ADDR,BYTES,CHAR              FLFLAG
;;HEXHL    MACRO   POINTR                       HXFLAG,RCFLAG
;;LCHAR    MACRO   PAR                          LOFLAG
;;MOVE     MACRO   FROM,TO,BYTES                MVFLAG
;;OUTHEX   MACRO   REG                          CXFLAG,COFLAG
;;PCHAR    MACRO   PAR                          COFLAG
;;PRINT    MACRO   TEXT,BYTES                   PRFLAG,COFLAG
;;READB    MACRO   BUFFR                        RCFLAG
;;READCH   MACRO   REG                          CIFLAG,COFLAG
;;SBC      MACRO                                (keine)
;;SYSF     MACRO   FUNC,AE                      (keine)
;;UCASE    MACRO   REG                          (keine)
;;UPPER    MACRO   REG                          (keine)
;;VERSN    MACRO   NUM                          (keine)
```

Kapitel 6

Das Lesen von Disketten-Dateien mit dem BDOS

EINFÜHRUNG

In Kapitel 5 haben wir Makros und Programme geschrieben, bei denen wir CP/M-Operationen mit dem BDOS ausgeführt haben. Dabei haben wir Diskettenoperationen außer acht gelassen. In diesem Kapitel werden wir nun unsere Möglichkeiten um das Lesen von Diskettendateien erweitern. Wie man Diskettendateien schreibt, lernen wir in Kapitel 7. Wir beginnen mit einer Zusammenfassung der Diskettenorganisation und der Art der Speicherung durch CP/M. Dann werden wir mehrere wichtige Makros für Diskettenoperationen entwickeln. Um die Anwendung dieser Makros zu zeigen, werden wir mehrere Programme schreiben: SHOW für die Anzeige von ASCII-Dateien auf der Konsole, DUMP für die Anzeige von COM-Dateien in hexadezimal und ASCII, ADDRESS für die Adressierung eines Umschlages aus einer bestehenden Brief-Datei und PAIR für die Zählung von Steuerzeichenpaaren.

DER DATEI-STEUERBLOCK (FCB)

Die Diskettenoberfläche ist in konzentrische Spuren aufgeteilt, die wiederum in Sektoren unterteilt sind. Die Diskettensteuerung ist in der Lage, diese Sektoren einzeln zu adressieren. CP/M adressiert jedoch größere Einheiten, die man Blöcke nennt. Eine Diskette mit einfacher Schreibdichte hat eine Blockgröße von 1024 (1K) oder 2048 (2K) Bytes. Disketten mit doppelter Schreibdichte haben Blockgrößen von 2K, 4K, 8K oder 16K Bytes. In jedem Sektor sind 128 Bytes Information; d.h. ein 1K-Block umfaßt 8 und ein 2K-Block 16 Sektoren.

Jede Datei auf einer CP/M-Diskette wird durch einen 32-Byte langen Dateisteuerblock (FCB) beschrieben, der in das Diskettenverzeichnis geschrieben wird. Die ersten 16 Bytes enthalten den Dateinamen, seine Länge und andere Merkmale. Die übrigen 16 Bytes geben die Nummern der diese Datei umfassenden Blöcke an.

Bevor man auf eine Datei zugreifen kann, muß eine Kopie des FCB im Speicher erzeugt werden. Da sich die Datei ändert, ändert sich auch der

FCB im Speicher. Am Ende einer Schreiboperation wird der FCB auf der Diskette aus dem Speicher aktualisiert. Wir müssen zwei FCBs auseinanderhalten, da sie sich manchmal unterscheiden. Da es für diese Unterscheidung unglücklicherweise keine eindeutige Namensgebung gibt, werden wir in diesem Buch die Ausdrücke „Speicher-FCB" und „Disketten-FCB" gebrauchen, wenn diese Unterscheidung nötig ist. Wir wählen den Namen „FCB", wenn beide Versionen gleich sind.

Bevor wir uns Details des FCB verdeutlichen, wollen wir auf die Binärverschlüsselung zurückblicken. Daten werden als Folge von binären Ziffern (0 oder 1) auf die Diskette geschrieben, ebenso wie im Speicher. Wie wir in Kapitel 5 gesehen haben, zeigt sich Information mal in binärer Form, mal in ASCII. Die Bitmuster für eine binäre Fünf und eine ASCII-Fünf, z.B., sind:

Binäre 5	*ASCII 5*
00000101	00110101

Einige der Bytes im FCB sind binär verschlüsselt, andere wiederum in ASCII. Normalerweise schreiben wir binäre Zahlen in hexadezimaler Form, eine binäre Drei z.B. als 03. ASCII-Zeichen könnte man auch hexadezimal darstellen, es ist jedoch nützlicher, sie in ASCII anzugeben. Eine ASCII-Drei zeigt sich also als 3, und nicht als 33.

Wenden wir uns nun den Details des FCB zu. Das erste Byte des Disketten-FCB ist die Benutzernummer, eine Binärzahl von 0-F hex. Das erste Byte des Speicher-FCB ist die Laufwerksnummer, eine Binärzahl von 0-10 hex. Den Laufwerken A, B und C entsprechen die Zahlen 1, 2 und 3. Die größte Zahl ist 10 hex, entsprechend Laufwerk P. Der Wert 0 an dieser Stelle spezifiziert das momentane Standardlaufwerk. Die nächsten acht Bytes (Bytes 1-8) zeigen den Dateinamen in ASCII. Dieses Feld ist ggf. mit Leerzeichen aufgefüllt. Die Erweiterung steht in den nächsten drei Bytes (Bytes 9-11). Dies ist ein wahlfreies Feld, das den Dateityp beschreibt. Man benutzt z.B. BAS für BASIC-Dateien, FOR für FORTRAN-Dateien, BAK für Backup-Dateien, etc. Auch dies Feld wird ggf. mit Leerzeichen gefüllt.

Für große Dateien ist mehr als ein FCB für die vollständige Spezifikation aller Blöcke erforderlich. In diesem Fall gibt es mehr als einen FCB mit demselben Dateinamen. Das nächste Byte (12) dient der Unterscheidung mehrerer FCBs mit demselben Namen. Die Zahl an dieser Stelle nennt man Extent-Nummer (Extent = Abschnitt). Für kleine Dateien ist die Nummer null. Die nächsten beiden Bytes betrachten wir nicht. Das letzte Byte in dieser Hälfte des FCB (15) gibt die Anzahl der Sätze (128-Byte-

Sektoren) an. Die übrigen 16 Bytes des FCB enthalten die Nummern der Blöcke auf der Diskette.

Das Beispiel in Abb. 6.1 zeigt fünf Disketten-FCBs. Beachten Sie, daß in der Abbildung die Dateinamen in ASCII, die übrige Information in hexadezimal dargestellt ist. Die Zwischenräume zwischen den Spalten dienen nur der Übersicht.

```
00 CPMIO   ASM 00000055 02030405060708090A0B0C0000000000
00 CPMIO   HEX 0000000C 0D0000000000000000000000000000000
00 SORT    COM 00000080 121314151617181 91A1B1C1D1E1F2021
00 SORT    COM 01000080 222324252627282 92A2B2C2D2E2F3031
00 SORT    COM 0200000A 323300000000000000000000000000000
```

Abb. 6.1: Fünf Dateisteuerblöcke für drei Dateien

Das erste Byte jedes Eintrags ist 0, weil alle Dateien vom Benutzer Null erstellt wurden. Die erste Datei, CPMIO.ASM, enthält 55 (85 dezimal) Sätze und steht in den Blöcken 02 bis 0C.

Die nächste Datei, CPMIO.HEX, umfaßt 0C Sätze und steht in Block 0D.

Die Einträge drei, vier und fünf heißen alle SORT.COM; sie beziehen sich alle auf dieselbe Datei. Sie unterscheiden sich durch die „Extent"-Nummer. Die Datei ist so groß, daß ein FCB zu ihrer Beschreibung nicht ausreicht. Die ersten 80 Sätze (Block 12-21) beschreiben den ersten Abschnitt (0). Die Blöcke 22-31 gehören zu dem zweiten Abschnitt (1). Die übrigen 0A Sätze gehören zum dritten Abschnitt (2).

Später in diesem Kapitel werden wir Makros schreiben, mit denen man Dateien aktiviert und liest. Wir müssen jedoch mit falsch geschriebenen Dateinamen rechnen und schreiben daher erst ein Makro für die Ausgabe von Fehlermeldungen.

EIN MAKRO ZUM AUSGEBEN EINER FEHLERMELDUNG MIT ABBRUCH DES PROGRAMMS

Immer, wenn ein Programm Eingaben von der Konsole erfordert, sollte man die Eingabe auf Gültigkeit prüfen. Im Fehlerfall sollte man eine Meldung ausgeben. Wird statt einer Ziffer ein Buchstabe eingegeben, sollte man dies dem Anwender mitteilen.

Etwas anderes sollten wir noch beachten. Die Anweisungen in einem Computer werden normalerweise der Reihe nach ausgeführt. Tritt aber ein Fehler auf, will man das Programm vielleicht an anderer Stelle fortsetzen oder auch beenden. Lassen Sie uns diese beiden Dinge – Ausgabe einer Meldung und Sprung zu einer Adresse – in einem Makro namens ERRORM kombinieren.

Das Makro PRINT haben wir bereits geschrieben, um Meldungen auszugeben. Wir benutzen das Makro PRINT jetzt für die Ausgabe einer Fehlermeldung. Dann werden wir zu einer anderen Adresse verzweigen. Das Makro ERRORM ist in Abb. 6.2 abgebildet. Tragen Sie es in Ihre Makrobibliothek ein.

Dieses Makro hat zwei Parameter. Der erste Parameter ist der Text der Meldung, der zweite Parameter ist die Adresse, zu der verzweigt wird, wenn die Meldung ausgegeben wurde. Wird dieser Parameter weggelassen, wird ein Warmstart ausgeführt.

Beachten Sie, daß der Parameter im Aufruf des Makros PRINT in spitze Klammern eingeschlossen ist. Dies ist notwendig, wenn der erste Parameter für ERRORM in spitze Klammern eingeschlossen ist. Der Makro-

```
ERRORM      MACRO   TEXT,WHERE
;;(Tagesdatum)
;;Makro für die Ausgabe einer Konsolnachricht.
;;Nachricht ist in Apostrophe eingeschlossen.
;;Optionaler zweiter Parameter ist Sprungadresse.
;;Fehlt der 2. Parameter: Warmstart.
;;Benötigte Makros: PRINT, CRLF
;;
;;Anwendung: ERRORM 'Meldung'
;;
            CRLF
            PRINT   <text>
            IF      NUL WHERE
            JMP     BOOT            ;Ende
            ELSE
            JMP     WHERE
            ENDIF                   ;;ERRORM
            ENDM
```

Abb. 6.2: Das Makro ERRORM zur Ausgabe einer Fehlermeldung und zum Programmabbruch

assembler entfernt einen Satz spitzer Klammern bei jeder Expandierung eines Makros. D.h., ein Paar Klammern wird bei der Expandierung des Makros ERRORM entfernt, das zweite Paar bei der Expandierung des Makros PRINT. Die spitzen Klammern sind notwendig, da gelegentlich Kommas benutzt werden, um einzelne Parameter voneinander zu trennen. Der Assembler benutzt Kommas zur Trennung zwischen Parametern, wenn sie nicht zwischen spitzen Klammern auftreten. Der Ausdruck

```
ERRORM <CR,LF,'?Datei existiert'>
```

enthält nur einen Parameter, der Ausdruck

```
ERRORM CR,LF,'?Datei existiert'
```

hingegen hat drei Parameter.

Wir haben jetzt die Grundlagen der CP/M-Dateiorganisation kennengelernt und ein Makro für die Ausgabe von Fehlermeldungen geschrieben. Wir werden jetzt lernen, wie man auf eine existierende Datei zugreift.

DAS ERÖFFNEN EINER BESTEHENDEN DATEI

Eine existierende Datei muß mit der BDOS-Funktion 15 eröffnet werden, bevor man auf sie zugreifen kann. Ein Speicher-FCB muß vor dem Funktionsaufruf bereitgestellt und teilweise gefüllt werden. Die Open-Funktion füllt den Rest des Speicher-FCB aus dem Disketten-FCB. Die notwendige Information ist:

Byte	*Inhalt*
0	Laufwerksnummer
1-8	Dateiname
9-11	Dateityp
12	Abschnitt (nullsetzen)
32	Satznummer (nullsetzen)

Wie wir früher in diesem Kapitel gesehen haben, enthält der Speicher-FCB an der Stelle 0 die Laufwerksnummer. Der Wert ist 0 für das Standardlaufwerk, 1 für Laufwerk A, 2 für Laufwerk B, etc. Der Dateiname und der Dateityp stehen in den nächsten 11 Bytes in der gewohnten ASCII-Darstellung. Die Abschnittsnummer an Position 12 wird nullgesetzt. Wenn sequentielle Verarbeitung gewünscht wird, muß die Satznummer an Position 32 ebenfalls gelöscht werden.

Es ist normalerweise nötig, diese Informationen vor jedem Eröffnen einer Datei bereitzustellen. Um dies zu vereinfachen, werden wir ein Makro schreiben. Bevor wir dies jedoch tun, wollen wir untersuchen, wie CP/M uns beim Aufbau des Speicher-FCB helfen kann.

Der Aufbau eines Speicher-FCB mit CP/M

Beim Start eines Programms durch CP/M kann man einen oder mehrere Parameter angeben. Die Parameter in der Kommandozeile nennt man kollektiv Tail. Sie werden automatisch im Konsolpuffer ab 82 hex gespeichert. CP/M bereitet auch einen Speicher-FCB mit dem ersten Parameter vor, der die Laufwerksnummer, den Dateinamen und den Dateityp (Bytes 0-11) umfaßt. Der FCB beginnt bei Adresse 5C hex. Wenn ein zweiter Parameter in der Kommandozeile angegeben wird, beginnt CP/M einen zweiten FCB bei Adresse 6C hex.

Wir können uns den Speicher-FCB mit DDT oder SID ansehen. Wir starten den Debugger mit einem einzelnen Parameter. Dann können wir den entsprechenden Speicherbereich ausgeben, um zu sehen, was CP/M gemacht hat. Wenn man den Debugger mit einem Parameter aufruft, versucht er, die Datei zu laden. Jedoch wird der FCB gelöscht, wenn der Debugger die Datei gefunden und geladen hat. Daher müssen wir für dieses Beispiel eine nicht existierende Datei wählen.

Angenommen, der DDT steht auf Laufwerk A. Gehen Sie auf Laufwerk B, und geben Sie das Kommando

```
A:DDT FIRST.EXT
```

oder

```
A:SID FIRST.EXT
```

(Beachten Sie, daß die Datei FIRST.EXT auf dem Laufwerk B nicht existieren darf.) Man kann das Kommando mit Groß- oder Kleinbuchstaben eingeben. Dieses Kommando veranlaßt CP/M, den DDT in den Speicher zu laden und zu starten. CP/M beginnt außerdem einen Speicher-FCB für die Datei FIRST.EXT bei Adresse 5C hex. Der DDT versucht, die Datei FIRST.EXT zu laden. Ein Fragezeichen erscheint, da die Datei nicht existiert.

Jetzt hat CP/M einen FCB bei Adresse 5C hex eingerichtet und den Command Tail in den Konsolpuffer ab Adresse 82 hex gespeichert. Untersuchen Sie diesen Bereich mit dem Kommando

```
D50,8F
```

Die folgende Anzeige sollte erscheinen:

```
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 46 49 52 .............FIR
0060 53 54 20 20 20 45 58 54 00 00 00 26 00 20 20 20 ST   EXT...&.
0070 20 20 20 20 20 20 20 20 00 00 00 00 00 00 00 00         ........
0080 0A 20 46 49 52 53 54 2E 45 58 54 00 00 00 00 00 . FIRST.EXT.....
```

Sie erinnern sich, daß der Debugger drei Teile anzeigt. Die erste Zahl jeder Zeile ist ihre Adresse in hexadezimaler Form. Die nächsten 16 Bytes zeigen den Inhalt des Speichers hexadezimal. Die entsprechenden ASCII-Zeichen werden ausgegeben, wenn sie druckbar sind. Ein Punkt erscheint, wenn ein Zeichen nicht druckbar ist.

Betrachten Sie die letzte Zeile der Ausgabe. In der ASCII-Darstellung sehen Sie den Command Tail, FIRST.EXT, im Konsolpuffer ab Adresse 82 hex. Die Länge des Command Tail (0A in diesem Fall) steht in Adresse 80 hex. Bei Adresse 81 hex steht immer ein Leerzeichen (20 hex). Wenn Sie das Kommando mit Kleinbuchstaben eingeben, konvertiert CP/M sie in Großbuchstaben. Das Ergebnis ist dasselbe.

Betrachten Sie nun den FCB bei 5C hex. Das erste Byte gibt die Laufwerksnummer an. Sie ist hier 0 und gibt das Standardlaufwerk an. Als nächstes erscheint der Dateiname FIRST. Da er weniger als acht Zeichen hat, ist er mit Leerzeichen aufgefüllt. Den Punkt zwischen dem Dateinamen und dem Dateityp sieht man im Konsolpuffer, nicht jedoch im FCB. Der Dateityp steht an seinem Platz an Position 9 im FCB. Werden weniger als drei Zeichen eingegeben, wird auch er mit Leerzeichen aufgefüllt.

Wir wollen nun das Beispiel leicht abändern. Gehen Sie mit Control-C zu CP/M, und geben Sie das Kommando

```
A:DDT B:FIRST.EXT
```

oder

```
A:SID B:FIRST:EXT
```

Dieses Kommando gleicht funktional dem vorigen, außer daß das Laufwerk B angegeben wurde. Untersuchen Sie den Bereich von 50 bis 8F hex erneut mit dem Kommando

```
D50,8F
```

In der Ausgabe sieht man, daß im Command Tail ab Adresse 82 hex das Laufwerk B erscheint.

```
0050 00 00 00 00 00 00 00 00 00 00 00 00 02 46 49 52 .............FIR
0060 53 54 20 20 20 45 58 54 00 00 00 26 00 20 20 20 ST   EXT...&.
0070 20 20 20 20 20 20 20 20 00 00 00 00 00 00 00 00          ........
0080 0C 20 42 3A 46 49 52 53 54 2E 45 58 54 00 00 06 . B:FIRST.EXT...
```

Im vorigen Beispiel wurde das Laufwerk nicht angegeben, und der FCB begann mit dem Wert 0. Jetzt ist durch die Angabe von Laufwerk B der Wert bei 5C hex 2.

Ein Makro zum Eröffnen einer Datei

Die vorangegangenen Beispiele zeigen, daß CP/M einen Speicher-FCB erzeugt, wenn ein Dateiname als Parameter in der Kommandozeile angegeben wird. Um auf die Datei zugreifen zu können, müssen wir noch die Abschnittsnummer und den Satzzähler löschen und die Datei mit der BDOS-Funktion 15 eröffnen.

Nach Rückkehr vom BDOS hat das A-Register den Wert FF hex, wenn die gewünschte Datei nicht existiert. Wir können das Programm fortsetzen oder es mit einer Fehlermeldung abbrechen.

Wir schreiben jetzt ein Makro zum Erstellen eines Speicher-FCB und zum Aufruf der BDOS-Funktion 15. Tragen Sie das Makro in Ihre Bibliothek ein (nach Abb. 6.3). Denken Sie an den Namen im Verzeichnis.

Sehen wir uns das Makro OPEN näher an. Der erste Befehl lädt das DE-Register mit der FCB-Adresse:

```
LXI D,POINTR
```

Das Symbol POINTR ist ein formaler Parameter. Der entsprechende Eintrag beim Makroaufruf ist erforderlich. Die nächsten drei Befehle speichern eine 0 an die Stellen 12 und 32 des Speicher-FCB. Die globale Subroutine OPEN2? wird dann aufgerufen, um die BDOS-Funktion 15 durchzuführen. Nach der Rückkehr vom BDOS steht FF hex im A-Register, wenn die Datei nicht eröffnet werden konnte. Der nächste Befehl erhöht das A-Register um 1. Dadurch wird das Z-Flag gelöscht, wenn die Datei eröffnet wurde. Das Programm setzt dann bei der lokalen Marke AROUND fort. Wurde die Datei jedoch nicht gefunden, wird der restliche Code ausgeführt.

Beim Aufruf des Makros OPEN wird man gewöhnlich den zweiten Parameter, der WHERE entspricht, weglassen, da wir sicherstellen wollen, daß die Datei tatsächlich existiert. In diesem Fall ist der Ausdruck IF NUL WHERE wahr, und das Makro ERRORM wird aufgerufen. Es erzeugt die Meldung

```
keine Quell-Datei
```

```
OPEN        MACRO   POINTR, WHERE
;;(Tagesdatum)
;;Inline-Makro für die Eröffnung einer existierenden Datei.
;;POINTR zeigt auf einen Dateisteuerblock.
;;Extent und laufende Satznummer werden gelöscht.
;;Sprung nach WHERE, wenn Datei nicht existiert.
;;Sonst Fehlermeldung und Sprung nach DONE .
;;Benötigte Makros: SYSF, ERRORM
;;
            LOCAL   AROUND
            LXI     D,POINTR
            XRA     A               ;null
            STA     POINTR+12       ;Extent
            STA     POINTR+32       ;laufende Satznummer
            CALL    OPEN2?
            INR     A               ;0=ok, FF heißt Fehler
            JNZ     AROUND
            IF      NUL WHERE
            ERRORM  'keine Quell-Datei', DONE
            ELSE
            JMP     WHERE
            ENDIF
            IF      NOT OPFLAG
OPEN2?:     SYSF    15              ;eröffne Datei
OPFLAG      SET     TRUE            ;nur eine Kopie
            ENDIF
AROUND:                             ;;OPEN
            ENDM
```

Abb. 6.3: Das Makro OPEN zum Eröffnen einer Datei

und beendet das Programm mit einem Sprung zur globalen Marke DONE.

Manchmal will man auch sicherstellen, daß eine Datei nicht existiert. Dann benutzen wir den zweiten Parameter beim Makroaufruf. Bei dem Aufruf

```
            OPEN    FCB1,CONT2
            ERRORM  'Dateiname existiert'
CONT2:
```

bezeichnet FCB1 den Speicher-FCB. Ist nun aber der Ausdruck IF NUL WHERE falsch, wird der Makroaufruf ERRORM im Makro OPEN nicht generiert. Das Programm wird bei der Marke CONT2 fortgesetzt, wenn die Datei nicht gefunden wird. Ist die Datei jedoch vorhanden, wird eine Meldung ausgegeben und das Programm beendet.

Wir brauchen noch drei weitere Makros für Diskettenoperationen, bevor wir das nächste Programm schreiben können. Das erste setzt die Pufferadresse zum Lesen der Datei, das zweite liest die Datei, und das dritte erzeugt aus einem Dateinamen einen Speicher-FCB. Beginnen wir mit dem Makro SETDMA.

EIN MAKRO ZUM SETZEN DER DMA-ADRESSE

Mit der BDOS-Funktion 20 liest man einen Sektor (128 Bytes) von der Diskette in den Speicher. Wir haben schon gesehen, daß die kleinste Informationseinheit, die CP/M von der Diskette lesen kann, ein Block von 1k oder mehr ist. Wird jedoch ein Sektor verlangt, sucht CP/M den entsprechenden Block und liest den Sektor in einen 128-Byte-Puffer. Die

```
SETDMA      MACRO   POINTR
;;(Tagesdatum)
;;Inline-Makro zum Setzen der DMA-Adresse, wo
;;der nächste Sektor gelesen oder geschrieben wird.
;;Benötigte Makros: SYSF
;;
            LOCAL   AROUND
            IF      NOT NUL POINTR
            LXI     D,POINTR
            ENDIF
            CALL    DMA2?
            IF      NOT DMFLAG
            JMP     AROUND
DMA2?:
            SYSF    26              ;setze DMA-Adresse
DMFLAG      SET     TRUE            ;nur eine Kopie
            ENDIF
AROUND:                             ;;SETDMA
            ENDM
```

Abb. 6.4: Das Makro SETDMA zum Setzen der DMA-Adresse

Speicheradresse dieses Puffers nennt man DMA-Adresse (Disk Memory Access).

Bei jedem Warmstart wird diese DMA-Adresse automatisch auf 80 hex gesetzt. Diese Adresse ist jedoch nicht immer angebracht. In diesem Kapitel sahen wir bereits, daß CP/M auch den Konsolpuffer hierhin legt, und auch der Debugger benutzt den Bereich zunächst als Stack. Darüber hinaus wollen wir manchmal eine ganze Datei beginnend bei 100 hex in den Speicher lesen. In diesem Fall ist die DMA-Adresse für den ersten Sektor 100 hex, für den zweiten 180 hex, für den dritten 200 hex, etc. Wir müssen also die DMA-Adresse ändern oder auch wieder auf 80 hex setzen können, falls sie von einem Programm auf einen anderen Wert gesetzt wurde.

Das Makro SETDMA in Abb. 6.4 setzt mit Hilfe der BDOS-Funktion 26 die DMA-Adresse. Beim Aufruf wird normalerweise die DMA-Adresse angegeben. Wird sie nicht angegeben, wird vorausgesetzt, daß das DE-Register vor dem Makroaufruf mit einer passenden Adresse geladen wurde. Tragen Sie das Makro in die Bibliothek ein.

Wir wollen nun ein Makro zum Lesen eines Sektors schreiben.

EIN MAKRO ZUM LESEN EINES SEKTORS VON DER DISKETTE

Bevor man eine Datei lesen kann, muß man einen Speicher-FCB mit dem Dateinamen aufbauen und die Datei mit der BDOS-Funktion 15 eröffnen. Ggf. muß man auch die DMA-Adresse mit der BDOS-Funktion 26 setzen. Danach kann man einen 128-Byte-Sektor mit der BDOS-Funktion 20 von der Diskette lesen. Die Information wird beginnend bei der momentanen DMA-Adresse in den Speicher gelesen.

Mit dem Makro READS in Abb. 6.5 können wir einen Sektor lesen. Tragen Sie es in die Bibliothek ein. Dieses Makro hat zwei Parameter. Der erste Parameter ist die Adresse eines Speicher-FCB. Der Assembler lädt diese Adresse, falls angegeben, in das DE-Register. In diesem Buch verwenden wir für den ersten Parameter gewöhnlich den Namen FCB1. Wird der Parameter weggelassen, wird angenommen, daß das DE-Register bereits geladen ist.

Der zweite Parameter wird, falls angegeben, gedruckt, nachdem ein Sektor gelesen worden ist. Das erlaubt zwar die Überwachung des Einlesens langer Dateien, verlangsamt den Vorgang allerdings erheblich.

Unser nächstes Makro verlangt die Eingabe eines Dateinamens und erstellt daraus einen Speicher-FCB.

```
READS       MACRO   POINTR, STAR
;;(Tagesdatum)
;;Inline-Makro zum Lesen eines Sektors.
;;POINTR zeigt auf den Dateisteuerblock.
;;Optionaler 2. Parameter ist das
;;nach dem Lesen eines Sektors auszugebende Zeichen.
;;Z-Flag ist gelöscht, wenn Dateiende erreicht.
;;Benötigte Makros: SYSF, PCHAR
;;
;; Anwendung: READS   FCB1
;;           READS   FCBS, '*'
;;
            LOCAL   AROUND
            IF      NOT NUL STAR
            PCHAR   STAR            ;ausgeben
            ENDIF
            IF      NOT NUL POINTR
            LXI     D,POINTR
            ENDIF
            CALL    READ2?
            ORA     A               ;setze Flags
            IF      NOT RDFLAG
            JMP     AROUND
READ2?:     SYSF    20              ;Sektor lesen
RDFLAG      SET     TRUE            ;nur eine Kopie
            ENDIF
AROUND:                             ;;READS
            ENDM
```

Abb. 6.5: Das Makro READS zum Lesen einer Datei

EIN MAKRO ZUR EINGABE EINES DATEINAMENS

Am Anfang dieses Kapitels haben wir gesehen, daß ein in der Kommandozeile angegebener Parameter im Standard-Konsolpuffer ab 82 hex gespeichert wird. Aus dem Parameter wird auch ein Speicher-FCB ab Adresse 5C hex gebildet. Das erste Byte bezeichnet das gewünschte Laufwerk. Der Wert 0 bezeichnet das Standardlaufwerk, 1 das Laufwerk A, etc. CP/M wandelt Klein- in Großbuchstaben um, füllt Dateinamen und Dateityp bei Bedarf mit Leerzeichen auf und entfernt den Punkt aus dem Dateinamen.

Ist ein Programm aber erst einmal gestartet, kann CP/M aus einem Dateinamen keinen FCB mehr bilden. Viele der Programme, die wir in diesem Buch schreiben, erwarten, daß ein Dateiname eingegeben wird. Wird in der Kommandozeile ein Dateiname angegeben, erzeugt CP/M einen Speicher-FCB. Wird jedoch kein Name angegeben, muß das Programm dafür sorgen. Es muß nun selbst die eingegebenen Zeichen verarbeiten. D.h. das Byte 0 des FCB muß 0 werden, wenn kein Laufwerk genannt wird oder 1, wenn Laufwerk A spezifiziert wird. Kleinbuchstaben müssen in große umgewandelt werden, etc.

Das Makro GFNAME in Abb. 6.6 fragt nach einem Dateinamen und erzeugt einen Speicher-FCB. Tragen Sie dieses Makro in Ihre Bibliothek ein. Der Speicher-FCB wird mit dem Parameter bezeichnet. Dies ist gewöhnlich die Adresse 5C hex, kann aber auch eine andere Adresse sein. Der Dateiname kann mit kleinen oder großen Buchstaben eingegeben werden. Falls gewünscht, kann ein Laufwerk angegeben werden.

```
GFNAME   MACRO    FCB
;;(Tagesdatum)
;;Inline-Makro für die Eingabe des Dateinamens von der Konsole
;;und Speicherung im FCB. Aus Kleinbuchstaben werden große.
;;Benötigte Makros: READB, FILL, UCASE
;;Subroutine GETCH ist Teil des Makros READB.
;;
         LOCAL    AROUND,PNAME,ENAME,EXTEN,GCHAR,GNAM2
         PUSH     H
         PUSH     D
         PUSH     B
         LXI      H,FCB
         SHLD     FCBS?
         CALL     GNAM?
         POP      B
         POP      D
         POP      H
         IF       NOT FNFLAG
         JMP      AROUND
FCBS?:             DS     2          ;orig. Pointer
GNAM?:
         CRLF
```

Abb. 6.6: Das Makro GFNAME für die Eingabe eines Dateinamens

```
GNAM2:
        PRINT   <' ',CR>
        PRINT   'Dateinamen eingeben: '
        LHLD    FCBS?
        XRA     A               ;null
        MOV     M,A             ;Standardlaufwerk
        INX     H
        FILL    ,11,BLANK
        XCHG
        READB                   ;Konsolpuffer
        CALL    GETCH           ;1. Zeichen
        JC      GNAM2           ;weiter
        CPI     BLANK
        JZ      GNAM2           ;weiter
        UCASE
        STAX    D
        CALL    GETCH
        RC                      ;kurzer Name
        CPI     BLANK
        RZ                      ;dito
        MVI     B,7             ;Länge-1
        UCASE
        CPI     PERIOD
        JZ      ENAME
        CPI     ':'             ;Laufwerk?
        JNZ     PNAME           ;nein
        LDAX    D               ;lies Laufwerk
        SUI     'A'-1           ;mache binär
        STAX    D               ;speichern
        CALL    GETCH           ;starte Dateiname
        JC      GNAM2           ;nur Laufwerk
        UCASE
        INR     B
        DCX     D
PNAME:                          ;erster Name
        INX     D
        STAX    D
        CALL    GETCH
        RC
        CPI     BLANK
```

Abb. 6.6: Das Makro GFNAME für die Eingabe eines Dateinamens (Forts.)

```
        RZ
        UCASE
        CPI     PERIOD
        JZ      ENAME
        DCR     B
        JNZ     PNAME           ;ok
        JMP     GNAM2           ;wenn 9 Zeichen
ENAME:
        LHLD    FCBS?           ;FCB
        LXI     D,9             ;Extent-Offset
        DAD     D
        XCHG
        MVI     B,3
EXTEN:                          ;Dateityp
        CALL    GETCH
        RC
        CPI     BLANK
        RZ
        UCASE
        STAX    D
        INX     D
        DCR     B
        JNZ     EXTEN
        RET                     ;fertig
;
FNFLAG  SET     TRUE
        ENDIF
AROUND:                         ;;GFNAME
        ENDM
```

Abb. 6.6: Das Makro GFNAME für die Eingabe eines Dateinamens (Forts.)

AUSGABE EINER ASCII-DATEI AUF DIE KONSOLE

Wie wir wissen, werden Informationen auf der Diskette und im Speicher als Folge von Bits gespeichert. Es gibt allerdings verschiedene Codierungen. Quelldateien werden in ASCII codiert, ausführbare Programme binär mit Meldungen in ASCII. Diese Unterscheidung ist wichtig, wenn wir uns eine Datei ansehen wollen. Eine ASCII-Datei kann man direkt zu einer Konsole oder einem Drucker senden, da dies ASCII-Geräte sind. Übertragen wir jedoch eine Binärdatei zur Konsole, wird sie ziemlich unleserlich sein.

Eine ASCII-Datei können wir uns mit dem CP/M-Kommando TYPE gefolgt von einem Dateinamen auf dem Bildschirm ansehen. Dies Kommando hat jedoch einige Nachteile. Erstens läuft die Anzeige so schnell durch, daß man vielleicht die gewünschte Stelle verpaßt. Mit Control-S kann man die Anzeige stoppen und mit einer beliebigen Taste fortsetzen. Mit Control-S kann die Anzeige auch erneut stoppen. Drückt man jedoch eine andere Taste als Control-S, wird die Anzeige abgebrochen, und man muß von vorn beginnen.

Ein anderer Nachteil von TYPE ist, daß es ein in den CCP eingebautes Kommando ist, das man aus Programmen wie WordStar nicht benutzen kann. Das Programm SHOW in Abb. 6.7 löst beide Probleme.

SHOW zeigt eine ASCII-Datei schirmweise auf der Konsole an. Wird die Leertaste gedrückt, erscheint der nächste Schirm. Durch Drücken der Return-Taste erscheint die nächste Zeile. Mit einer beliebigen anderen Taste kann das Programm beendet werden.

Da SHOW ein ausführbares Programm ist, kann man es auch im WordStar aufrufen. Zur Anzeige des Quellprogramms von SHOW gibt man z.B. das Kommando

```
SHOW SHOW.ASM
```

Ein Laufwerk kann angegeben werden. Ist z.B. das ausführbare Programm auf Laufwerk A und die Quelldatei auf Laufwerk B, gibt man das Kommando

```
A:SHOW B:SHOW.ASM
```

SHOW kann auch ohne Parameter aufgerufen werden. Es wird dann nach dem Dateinamen gefragt. Wenn die Eingabe fehlerhaft ist, kann sie wiederholt werden. Existiert die Datei nicht oder ist sie eine COM-Datei, wird das Programm mit einer entsprechenden Meldung beendet.

SHOW arbeitet mit einem Bildschirm von 24 Zeilen. Hat Ihr Bildschirm eine andere Zeilenzahl, ändern Sie die Definition des Symbols LMAX von 24 auf die gewünschte Zahl.

Geben Sie das Programm nach Abb. 6.7 ein, und assemblieren Sie es. Geben Sie das Quellprogramm SHOW aus. Wird nur das Kommando SHOW gegeben, verlangt das Programm einen Dateinamen. Drücken Sie die Leertaste zur Anzeige des nächsten Schirms oder Return zur Anzeige der nächsten Zeile. Drücken Sie irgendeine andere Taste, um das Programm zu beenden.

```
TITLE   'Zeige ASCI-Datei auf Konsole'
;
;(Tagesdatum)
;
;Anwendung: SHOW DISKFILE.EXT
;
;Mit 'Leerzeichen' wird der nächste Schirm angezeigt;
;mit 'RETURN' die nächste Zeile.
;Gleiche Funktion wie TYPE, aber
;SHOW kann von WordStar aus aufgerufen werden.
;
FALSE       EQU     0
TRUE        EQU     NOT FALSE
;
BOOT        EQU     0           ;Warmstart
BDOS        EQU     5           ;BDOS-Eingang
FCB1        EQU     5CH         ;Eingabe FCB
DBUFF       EQU     80H         ;Standardpuffer
TPA         EQU     100H        ;Anwenderbereich
LMAX        EQU     24          ;Zeilen pro Schirm
;
;Setze Flags im Hauptprogramm, damit von
;einigen Subroutines nur eine Kopie generiert wird.
;Setze die Flags vor dem MACLIB-Call.
;
CIFLAG      SET     FALSE       ;Zeicheneingabe von der
                                Konsole
CMFLAG      SET     FALSE       ;ASCII-Vergleich
COFLAG      SET     FALSE       ;Zeichenausgabe auf die
                                Konsole
CRFLAG      SET     FALSE       ;neue Zeile
DMFLAG      SET     FALSE       ;setze DMA
FLFLAG      SET     FALSE       ;Füll-Zeichen
FNFLAG      SET     FALSE       ;Dateiname lesen
OPFLAG      SET     FALSE       ;Datei eröffnen
PRFLAG      SET     FALSE       ;Konsolpuffer drucken
RCFLAG      SET     FALSE       ;Konsolpuffer lesen
RDFLAG      SET     FALSE       ;Datei lesen
;Ende der Flags
;
```

Abb. 6.7: Das Programm SHOW für die Ausgabe einer ASCII-Datei auf die Konsole

```
                MACLIB   CPMMAC
;
ORG             TPA
;
START:
                ENTER
                VERSN    '(Tagesdatum).SHOW'
                LDA      FCB1+1
                CPI      BLANK          ;Dateiname?
                JNZ      OPEND          ;ja
                GFNAME   FCB1           ;Dateiname lesen
OPEND:
                COMPRA   'COM',FCB1+9   ;COM-Datei?
                JZ       NOCOM          ;ja
                OPEN     FCB1           ;Quell-Datei
                SETDMA   DBUFF          ;benutze Standard
                LXI      H,100H         ;setze Pointer
NEXTSC:
                CALL     SCREEN         ;nächster Schirm
FREE2:
                READCH                  ;warte auf Eingabe
                CPI      BLANK          ;Leerzeichen?
                JZ       NEXTSC         ;nächster Schirm
                CPI      CR
                JNZ      DONE           ;Abbruch
                PCHAR    CR
                MVI      B,1            ;setze auf 1 Zeile
                CALL     LINE           ;1 Zeile
                JMP      FREE2
;
;Routine zum Füllen des Konsolschirms
;
SCREEN:
                MVI      B,LMAX         ;Zeilenzähler
                PCHAR    CR
NEXTLN:
                CALL     LINE
                DCR      B              ;zählen
                JNZ      NEXTLN         ;weiter
```

Abb. 6.7: Das Programm SHOW für die Ausgabe einer ASCII-Datei auf die Konsole (Forts.)

```
                RET
;
;Routine zum Anzeigen einer Zeile
;
LINE:
                MOV     A,H             ;prüfe Pointer
                ORA     A               ;noch 80-FF?
                JZ      LIN3            ;ja
                READS   FCB1            ;lies einen Sektor
                JNZ     EOFILE          ;Dateiende
                LXI     H,DBUFF         ;Zeiger zurücksetzen
                JMP     LINE
LIN3:
                MOV     A,M
                INX     H
                ANI     7FH             ;lösche Paritybit
                CPI     EOF             ;Dateiende
                JZ      EOFILE          ;ja
                MOV     D,A             ;retten
                CPI     CR              ;Zeilenende?
                JNZ     LIN2            ;nein
                MOV     A,B             ;prüfe Position
                CPI     1               ;letzte Zeile?
                RZ                      ;ja, kein CR
                MOV     A,D             ;lade CR
LIN2:
                PCHAR                   ;Ausgabe auf Konsole
                MOV     A,D             ;lade
                CPI     CR              ;Zeilenende?
                JNZ     LINE            ;nein
                RET
NOCOM:
                ERRORM  'Benutze DUMP für eine COM-Datei',DONE
EOFILE:
                READCH                  ;letze Seite
DONE:
                EXIT
;
                END     START
```

Abb. 6.7: Das Programm SHOW für die Ausgabe einer ASCII-Datei auf die Konsole (Forts.)

Nach dem Start von SHOW wird zunächst das zweite Byte des FCB1 (das erste Byte des Dateinamens) geprüft. Ein Leerzeichen an dieser Stelle zeigt an, daß kein Dateiname angegeben wurde. In diesem Fall wird er mit dem Makro GFNAME verlangt.

Dann wird sichergestellt, daß der Dateityp nicht COM ist. Ist alles in Ordnung, wird die Datei eröffnet. Der Standard-DMA-Puffer bei Adresse 80 hex wird zum Lesen der Sektoren benutzt. Um sicher zu sein, daß diese Adresse nicht von einem vorangegangenen Programm verändert wurde, setzen wir sie neu.

Bevor wir das nächste Programm schreiben, fügen wir noch ein neues Makro in unsere Bibliothek ein.

EIN MAKRO ZUM ABBRUCH EINES PROGRAMMS VON DER KONSOLE

Manchmal ist es aus dem einen oder anderen Grund nötig, ein Programm vorzeitig zu beenden - vielleicht, weil eine Zahl von der Konsole aus falsch eingegeben wurde, oder weil ein Programm die gewünschte Information bereits gegeben hat und der Rest des Programms nicht mehr benötigt wird. Aus diesem Grund erlauben viele Betriebssysteme den vorzeitigen Abbruch eines Programms. Bei CP/M ist dies leider nicht vorgesehen. Daher wollen wir hierfür das Makro ABORT schreiben. Tragen Sie es nach Abb. 6.8 in Ihre Bibliothek ein.

```
ABORT       MACRO   CHAR
;;(Tagesdatum)
;;Inline-Makro, um ein Programm abzubrechen,
;;wenn die Taste CHAR gedrückt wird.
;;Fehlt CHAR, kann es eine beliebige Taste sein.
;;Sprung nach DONE zu Abbruch.
;;Anwendung: ABORT ESC
;;
;;Benötigte Makros: READCH
;;
            LOCAL   AROUND
            PUSH    H
            PUSH    D
            PUSH    B
            MVI     C,11            ;Konsolstatus
```

Abb. 6.8: Das Makro ABORT zum Abbruch eines Programms von der Konsole aus

```
        CALL    BDOS
        POP     B
        POP     D
        POP     H
        RRC
        JNC     AROUND          ;kein Zeichen
        READCH                  ;Zeichen lesen
        IF      NUL CHAR
        JMP     DONE
        ELSE
        CPI     CHAR
        JZ      DONE
        ENDIF
AROUND:                         ;;ABORT
        ENDM
```

Abb. 6.8: Das Makro ABORT zum Abbruch eines Programms von der Konsole aus (Forts.)

Sehen wir nun, wie das Makro ABORT arbeitet. Der Aufruf des Makros kann überall im Programm stehen, wo Sie einen Abbruch zulassen wollen. Mit der BDOS-Funktion 11 wird der Konsolstatus getestet. Sie liefert den Wert FF hex im A-Register, wenn eine Taste gedrückt worden ist; sonst den Wert 0. Im Makro wird der Inhalt des A-Registers zyklisch in das Carry-Flag geschoben, das getestet wird. Zeigt der Konsolstatus an, daß keine Taste gedrückt worden ist, wird der Rest des Makros übergangen.

Ist das Carry-Flag gesetzt, wurde eine Taste gedrückt. Wenn das Makro ohne Parameter aufgerufen wurde, wird das Programm beendet. Wurde jedoch beim Aufruf ein Parameter angegeben, wird das eingegebene Zeichen mit diesem Parameter verglichen. Bei Übereinstimmung wird das Programm durch einen Sprung zu der Marke DONE beendet.

Wir wollen dies an einem Beispiel verdeutlichen. Wir werden immer den Aufruf

```
ABORT ESC
```

bei Programmen in diesem Buch verwenden. Durch die generierten Befehle wird das Makro das Programm nur beenden, wenn die ESC-Taste gedrückt wird. Andere Tasten werden ignoriert.

Wir sind nun bereit, unser nächstes Programm zu schreiben.

AUSGABE EINER BINÄRDATEI AUF DIE KONSOLE

Das Programm in Abb. 6.7 zeigt eine ASCII-Datei auf dem Bildschirm an, kann aber für eine Binärdatei nicht benutzt werden. Manchmal will man aber eine Binärdatei (COM) untersuchen. Dies ist mit dem Programm in Abb. 6.9 möglich. Es kann natürlich auch eine ASCII-Datei anzeigen, die Ausgabe ist jedoch nicht so leicht lesbar wie bei dem Programm SHOW. Geben Sie das Programm DUMP ein. Assemblieren und starten Sie es. Das Kommando ist wie bei dem Programm SHOW.

```
TITLE   'DUMP binäre Datei zur Konsole'
;
;(Tagesdatum)
;
;Anwendung: DUMP (Dateiname)
;
;Zwischenr = nächster Schirm
;<CR>     = nächste Zeile
;<ESC>    = Abbruch
;
FALSE       EQU     0
TRUE        EQU     NOT FALSE
;
BOOT        EQU     0           ;Warmstart
BDOS        EQU     5           ;BDOS-Eingang
FCB1        EQU     5CH         ;Eingabe-FCB
DBUFF       EQU     80H         ;Standardpuffer
TPA         EQU     100H        ;Anwenderbereich
LMAX        EQU     23          ;Zeilen pro Schirm
;
;Setze Flags im Hauptprogramm, damit von
;einigen Subroutines nur eine Kopie generiert wird.
;Setze die Flags vor dem MACLIB-Call.
;
CIFLAG      SET     FALSE       ;Eingabe Konsole Zeichen
COFLAG      SET     FALSE       ;Ausgabe Konsole Zeichen
CRFLAG      SET     FALSE       ;neue Zeile
CXFLAG      SET     FALSE       ;binär in C nach hex
DMFLAG      SET     FALSE       ;setze DMA
FLFLAG      SET     FALSE       ;Füll-Zeichen
```

Abb. 6.9: Das Programm DUMP für die Anzeige einer Binärdatei

```
FNFLAG      SET     FALSE           ;Dateiname lesen
GTFLAG      SET     FALSE           ;Zeichen von Puffer lesen
OPFLAG      SET     FALSE           ;Datei eröffnen
PRFLAG      SET     FALSE           ;Konsolpuffer drucken
RCFLAG      SET     FALSE           ;Konsolpuffer lesen
RDFLAG      SET     FALSE           ;Sektor lesen
;Ende der Flags
;
            MACLIB  CPMMAC
;
ORG         100H
;
START:
            ENTER
            VERSN   '(Tagesdatum).DUMP '
            LDA     FCB1+1
            CPI     BLANK           ;Dateiname?
            JNZ     OP3             ;ja
            GFNAME  FCB1            ;Dateiname lesen
OP3:
            OPEN    FCB1            ;Eingabe Datei
            SETDMA  DBUFF           ;Sektor Adresse
            LXI     H,TPA           ;Anzeige Pointer
            SHLD    PNTR
            PRINT   'Leertaste für nächsten Bildschirm, '
            PRINT   '<CR> nächste Zeile, <ESC> zum
                    Abbrechen'
NEWLN:                              ;neue Zeile
            CRLF
            PUSH    H               ;Pufferzeiger
            LHLD    PNTR            ;Anzeige Pointer
            OUTHEX  H               ;Adresse
            OUTHEX  L
            LXI     D,10H           ;nächste Zeile
            DAD     D
            SHLD    PNTR
            POP     H               ;Pufferzeiger
            PCHAR   BLANK
NEXT:
            MOV     A,H             ;prüfe Pointer
```

Abb. 6.9: Das Programm DUMP für die Anzeige einer Binärdatei (Forts.)

```
        ORA     A           ;noch 80--FF hex?
        JZ      NEXT2       ;ja
        READS   FCB1        ;einen Sektor lesen
        JNZ     DONE        ;Dateiende
        LXI     H,DBUFF
        JMP     NEXT
NEXT2:
        OUTHEX  M
        INX     H
        MOV     A,L
        ANI     0FH         ;Zeilenende?
        JZ      PASC        ;ja
        ANI     3           ;Leerzeichen?
        JNZ     NEXT        ;nein
        PCHAR   BLANK
        JMP     NEXT
PASC:                       ;ASCII-Dump
        PRINT   ''
        PUSH    H           ;Pufferzeiger
        MOV     A,L
        SUI     10H
        MOV     L,A         ;ändere Zeiger
PAS2:
        MOV     A,M
        INX     H
        CPI     7FH         ;Bit 7 an?
        JNC     PAS3        ;ja
        CPI     BLANK       ;Control-Zeichen?
        JNC     PAS4        ;nein
PAS3:
        MVI     A,PERIOD    ;ändern in Punkt
PAS4:
        PCHAR               ;drucken
        MOV     A,L
        ANI     0FH         ;Zeilenende?
        JNZ     PAS2        ;nein
        POP     H           ;Pufferzeiger
        ABORT   ESC         ;Abbruch?
        LDA     LINE
        DCR     A
```

Abb. 6.9: Das Programm DUMP für die Anzeige einer Binärdatei (Forts.)

```
                STA     LINE
                JNZ     NEWLN
                MVI     A,LMAX
                STA     LINE
;
;stop bis Leertaste gedrückt wird
FREEZ:
                READCH                  ;warte auf Eingabe
                CPI     BLANK           ;Leerzeichen?
                JZ      NEWLN
                ANI     1FH             ;konvertiere in Control
                CPI     CR              ;nächste Zeile?
                JNZ     FREZ2           ;nein
                MVI     A,1             ;1 Zeile
                STA     LINE
                JMP     NEWLN
LINE:
                DB      LMAX            ;Zeilenzähler
FREZ2:
                CPI     ESC             ;Abbruch?
                JNZ     FREEZ           ;nein
DONE:
                EXIT
;
PNTR:           DS      2               ;Anzeige Pointer
                END     START
```

Abb. 6.9: Das Programm DUMP für die Anzeige einer Binärdatei (Forts.)

Die Ausgabe von DUMP ähnelt der des DDT. Jede Zeile beginnt mit der entsprechenden Speicheradresse (beginnend bei der TPA). Dann werden 16 Bytes hexadezimal ausgegeben. Die ASCII-Zeichen werden dann ausgegeben, wenn sie druckbar sind, sonst werden sie durch Punkte ersetzt. Die Anzeige stoppt, wenn der Bildschirm gefüllt ist. Mit dem Drücken der Leertaste erscheint der nächste Bildschirminhalt, mit Return die nächste Zeile. Mit ESC kann man das Programm beenden. (Die Eingabe wird immer am Ende einer Zeile geprüft.)

Die verbleibenden zwei Programme in diesem Kapitel verdeutlichen den Gebrauch der Datei-bezogenen Makros. Beide Programme benutzen unser neues Makro zum Lesen existierender Dateien.

AUTOMATISCHES SCHREIBEN VON ADRESSEN AUF UMSCHLÄGE

Wenn Sie für das Schreiben Ihrer Briefe ein Textprogramm wie den WordStar benutzen, können Sie den Brief mit dem Drucker drucken. Sie brauchen aber immer noch eine Schreibmaschine zum Adressieren des Umschlages. Mit dem Programm in Abb. 6.10 können Sie automatisch den Umschlag drucken, nachdem Sie den Brief gedruckt haben.

Eine WordStar-kompatible Datei für den Beginn eines Briefes mag so aussehen:

..Name des Absenders
.op (keine Seitenzahlen)
(Leerzeile)
Datum
(Leerzeile)
Name des Empfängers
Straße
Postleitzahl Ort
(Leerzeile)
Anrede

Viele Textprogramme interpretieren ein besonderes Zeichen in Spalte 1 als Beginn einer Kommandozeile. Dieses Zeichen ist häufig ein Punkt, da ein Punkt in der ersten Spalte normalerweise nicht vorkommt. Die erste Zeile dieser Datei beginnt mit zwei Punkten, dem WordStar-Symbol für eine Kommentarzeile.

Das Programm in Abb. 6.10 kann Namen und Adresse des Empfängers extrahieren und auf einen Umschlag drucken. Wenn der Name des Absenders als Kommentar am Beginn der Datei steht, wird er im Absenderfeld gedruckt. Erstellen Sie die Datei ADDRESS, und geben Sie das Programm nach Abb. 6.10 ein. Assemblieren und starten Sie es.

Das Programm ADDRESS wird durch Eingabe seines Namens und dem Namen der Briefdatei gestartet. Der Name kann auch später eingegeben werden. Das Programm bietet eine weitere Möglichkeit. Der Absendername steht normalerweise in der linken oberen Ecke des Umschlags. Gibt man nach dem Dateinamen ein separates L ein, wird der Name des Absenders mit dem des Empfängers ausgerichtet. Diese Form ist geeigneter für Adreßaufkleber.

```
TITLE   'ADDRESS adressiert Briefumschläge'
;
;(Tagesdatum)
;
;Anwendung:
;  ADDRESS DISKFILE.EXT    (für Umschlag)
;  ADDRESS DISKFILE.EXT L  (für Aufkleber)
;
;die Briefdatei hat die Form:
;..Absender
;.op und andere Punktkommandos (optional)
;Leerzeile (optional)
;Datum (1 Zeile)
;Leerzeile (eine oder mehrere)
;Adresse
;Leerzeile
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
BOOT    EQU     0               ;Warmstart
BDOS    EQU     5               ;BDOS-Eingang
;
FCB1    EQU     5CH             ;1. Parameter
FCB2    EQU     6CH             ;2. Parameter
DBUFF   EQU     80H             ;Standardpuffer
TPA     EQU     100H            ;Anwenderbereich
BEL     EQU     7
;
;Setze Flags im Hauptprogramm, damit von
;einigen Subroutines nur eine Kopie generiert wird.
;Setze diese Flags vor dem MACLIB-Call.
;
CMFLAG  SET     FALSE           ;ASCII-Vergleich
COFLAG  SET     FALSE           ;Ausgabe Konsole-Zeichen
CRFLAG  SET     FALSE           ;neue Zeile
FLFLAG  SET     FALSE           ;Füll-Zeichen
FNFLAG  SET     FALSE           ;Dateiname lesen
LOFLAG  SET     FALSE           ;Drucker-Ausgabe
OPFLAG  SET     FALSE           ;Datei eröffnen
```

Abb. 6.10: Das Programm ADDRESS zum Beschriften von Briefumschlägen

```
PRFLAG  SET     FALSE           ;Drucker-Ausgabe
RCFLAG  SET     FALSE           ;Konsole lesen
RDFLAG  SET     FALSE           ;Datei lesen
;Ende der Flags
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum).ADDRESS '
        LDA     FCB1+1
        CPI     BLANK           ;Dateiname?
        JNZ     OPEND           ;ja
        GFNAME  FCB1            ;Lesen Dateiname
OPEND:
        COMPRA  'COM', FCB1+9   ;COM Datei?
        JZ      NOCOM           ;ja
        MVI     A,35            ;Umschlag-Einrückung
        STA     INDNC           ;speichere Zähler
        LDA     FCB2+1          ;2. Parameter?
        CPI     BLANK
        JZ      NOPAR           ;nein
        MVI     A,14
        STA     INDNC           ;Aufkleber-Einrückung
NOPAR:
        OPEN    FCB1            ;Quell-Datei
        READS   FCB1            ;1. Sektor
        LXI     H,DBUFF         ;Text-Puffer
;suche Punkt mit Absender
        MOV     A,M
        CPI     PERIOD
        JNZ     NOPER           ;kein Absender
        INX     H
        MOV     A,M
        CPI     PERIOD          ;2. Punkt?
        JNZ     FPER            ;nein
        INX     H               ;überspringe Punkt
        MVI     B,14            ;Einrückung
```

Abb. 6.10: Das Programm ADDRESS zum Beschriften von Briefumschlägen (Forts.)

```
        CALL    PLINE           ;für Absender
FPER2:                          ;suche weitere Punkte
        MOV     A,M
        CPI     PERIOD
        JNZ     FBLNK
        CALL    LINE
        JMP     FPER2
;
;kein Absender, verarbeite weitere Punktkommandos
;
FPER:
        CALL    LINE            ;nächste Zeile
        CPI     PERIOD
        JZ      FPER
NOPER:
        MVI     B,1
        CALL    LINEFD          ;überspringe Absender
FBLNK:                          ;suche Leerzeichen
        MOV     A,M
        CPI     BLANK+1
        JNC     FDATE           ;kein Leerzeichen oder CR
        CALL    LINE
        JMP     FBLNK
FDATE:                          ;suche Datum
        CALL    LINE            ;gehe bis Leerzeichen
        MOV     A,M
        CPI     BLANK+1
        JC      FDATE           ;mehr Leerzeichen
;Leerzeilen bis Adresse
        MVI     B,9
        CALL    LINEFD
;
;Adresse drucken
;
ADDR2:
        MOV     A,M
        CPI     BLANK+1         ;mehr
        JC      DONE
        LDA     INDNC           ;Einrückung
        MOV     B,A             ;für Adresse
```

Abb. 6.10: Das Programm ADDRESS zum Beschriften von Briefumschlägen (Forts.)

```
            CALL    PLINE
            JMP     ADDR2
;
;Leerzeilen zum Drucker, in B steht Anzahl
;
LINEFD:
            LCHAR   LF
            DCR     B
            JNZ     LINEFD
            RET
;
;Ausgabe Zeile auf Drucker und Konsole
;
PLINE:
            CALL    INDEN
            MOV     A,M             ;1. Zeichen
            PCHAR
            LCHAR
PLINE2:
            CALL    CPOINT          ;prüfe Pointer
            MOV     A,M             ;nächstes Zeichen
            PCHAR
            LCHAR
            ANI     7FH             ;lösche Parity
            CPI     LF
            JNZ     PLINE2          ;ja
            INX     H
            RET
;
;Gehe zum Beginn der nächsten Zeile, nach LF
;
LINE:
            CALL    CPOINT          ;prüfe Pointer
            MOV     A,M             ;nächstes Zeichen
            ANI     7FH             ;lösche Parity
            CPI     LF
            JNZ     LINE            ;ja
            INX     H
            RET
;
```

Abb. 6.10: Das Programm ADDRESS zum Beschriften von Briefumschlägen (Forts.)

```
;Erhöhe HL Pointer, prüfe ob größer als 80+80 hex.
;Lies nächsten Sektor, wenn das der Fall ist.
;
CPOINT:
        INX     H               ;Pointer
        MOV     A,H             ;prüfe Pointer
        ORA     A               ;<100H?
        RZ                      ;ja, ok
        READS   FCB1            ;nächster Sektor
        JNZ     DONE            ;Dateiende
        LXI     H,DBUFF         ;Pointer zurücksetzen
        RET
INDEN:
        MVI     A,BLANK
INDEN2:
        PCHAR
        LCHAR
        DCR     B
        JNZ     INDEN2
        RET
;
NOCOM:  ERRORM  '?COM-Datei',DONE
;
DONE:
        EXIT
INDNC:  DS      1               ;Einrückung
;
        END     START
```

Abb. 6.10: Das Programm ADDRESS zum Beschriften von Briefumschlägen (Forts.)

PRÜFUNG AUF PAARIGE STEUERZEICHEN

Unser letztes Programm sucht nach paarigen Steuerzeichen in Dateien. Mit dem Makro GFNAME geben wir einen Dateinamen von der Konsole ein, mit dem Makro OPEN eröffnen wir eine Datei, und mit dem Makro ERRORM geben wir Meldungen aus und beenden das Programm. Außerdem führen wir die Inline-Makros REPT, IRP und IRPC ein.

Manche Textprogramme benutzen paarige Steuerzeichen, um besondere Funktionen zum Drucken zu kennzeichnen. In WordStar-Dateien wird

z.B. eine zu unterstreichende Passage in Control-S-Zeichen eingeschlossen. Andere Steuerzeichen werden für Fettdruck, hoch- und tiefgestellte Zeichen verwendet. Geht das zweite Zeichen eines Paares verloren, wird das Dokument recht eigenartig aussehen. Fehlt z.B. das zweite Unterstreichungs-Steuerzeichen, wird der ganze Rest unterstrichen.

Mit dem Programm in Abb. 6.11 kann man ein Dokument auf paarige Steuerzeichen überprüfen. Es ist für den WordStar entworfen, kann aber für andere Textprogramme leicht modifiziert werden. Insbesondere zählt das Programm die Anzahl der Zeichen Control-B, -D, -S, -T, -V und -X. Wird bei einem dieser Zeichen eine ungerade Anzahl ermittelt, erscheint eine Fehlermeldung. Ist die Anzahl gerade, erscheint die Meldung „No unbalanced pairs". Wenn allerdings zwei Control-Zeichen desselben Typs fehlen, merkt das Programm diesen Fehler nicht.

```
TITLE   'PAIR Überprüfung auf paarige Control-Zeichen'
;
;(Tagesdatum)
;
;Anwendung: PAIR DISKFILE.EXT
;
;Überprüfe, daß Control-B, -D, -S, -T, -V, and -X paarig
;auftreten
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
BOOT    EQU     0               ;Warmstart
BDOS    EQU     5               ;BDOS-Eingang
FCB1    EQU     5CH             ;Eingabe-FCB
DBUFF   EQU     80H             ;Standardpuffer
TPA     EQU     100H            ;Anwenderbereich
;
;Setze Flags im Hauptprogramm, damit von
;einigen Subroutines nur eine Kopie generiert wird.
;Setze die Flags vor dem MACLIB-Call.
;
CIFLAG  SET     FALSE           ;Eingabe Konsole Zeichen
CMFLAG  SET     FALSE           ;ASCII-Vergleich
COFLAG  SET     FALSE           ;Ausgabe Konsole Zeichen
```

Abb. 6.11: Das Programm PAIR zum Zählen von paarigen Control-Zeichen

```
CRFLAF  SET     FALSE           ;neue Zeile
DMFLAG  SET     FALSE           ;setze DMA
FLFLAG  SET     FALSE           ;Füll-Zeichen
FNFLAG  SET     FALSE           ;Dateiname lesen
OPFLAG  SET     FALSE           ;Datei eröffnen
PRFLAG  SET     FALSE           ;Konsolpuffer drucken
RCFLAG  SET     FALSE           ;Konsolpuffer lesen
RDFLAG  SET     FALSE           ;Datei lesen
;Ende der Flags
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   ' (Tagesdatum).PAIR '
        LDA     FCB1+1
        CPI     BLANK           ;Dateiname?
        JNZ     OPEND           ;ja
        GFNAME  FCB1            ;Dateiname lesen
OPEND:
        COMPRA  'COM', FCB1+9   ;COM Datei?
        JZ      NOCOM           ;ja
        PRINT   <CR,LF,'Suche nach unpaarigen'>
        PRINT   '^B, ^D, ^S, ^I, ^V, ^X'
        OPEN    FCB1            ;Quell-Datei
        SETDMA  DBUFF           ;benutze Standard
        LXI     H,100H          ;setze Pointer
LINE:
        MOV     A,H             ;prüfe Pointer
        ORA     A               ;noch 80-FF?
        JZ      LIN3            ;ja
        READS   FCB1            ;lies einen Sektor
        JNZ     EOFILE          ;Dateiende
        LXI     H,DBUFF         ;Pointer zurücksetzen
        JMP     LINE
LIN3:
        MOV     A,M
        INX     H
```

Abb. 6.11: Das Programm PAIR zum Zählen von paarigen Control-Zeichen (Forts.)

```
            ANI     7FH             ;lösche Parity
            CPI     EOF             ;Dateiende
            JZ      EOFILE          ;ja
;
;;Inline-Makro zum Zählen von Control-Zeichen
            IRPC    X?,BDSTVX
            LOCAL   AROUND
CTR&X?      EQU     '&X?' - '§'
            CPI     CTR&X?
            JNZ     AROUND
            LDA     CNT&X?
            INR     A
            STA     CNT&X?
            JMP     LINE
CNT&X?:     DB      0
AROUND:
            ENDM
            JMP     LINE            ;nein
NOCOM:
            ERRORM  'COM-Datei?' ,DONE
UFLAG:      DB      0
;
EOFILE:
;;Inline-Makro zum Anzeigen von Control-Zeichen
            IRPC    X?,BDSTVX
            LOCAL   AROUND
            LDA     CNT&X?
            RAR                     ;;ungerade
            JNC     AROUND          ;;nein
            PRINT   <CR,LF, unpaariges ^'>
            PCHAR   '&X?'
            LDA     UFLAG
            INR     A
            STA     UFLAG
AROUND:
            ENDM
            LDA     UFLAG
            ORA     A               ;ok
            JNZ     DONE            ;nein
            PRINT   <CR,LF,'Keine unpaarigen Control-Zeichen'>
```

Abb. 6.11: Das Programm PAIR zum Zählen von paarigen Control-Zeichen (Forts.)

```
DONE:
        EXIT
;
        END     START
```

Abb. 6.11: Das Programm PAIR zum Zählen von paarigen Control-Zeichen (Forts.)

Das Programm wird wie die anderen in diesem Kapitel durch Eingabe seines Namens gefolgt von dem Namen der zu lesenden Datei gestartet. Die Befehle sind ähnlich, aber wir führen hier etwas Neues ein. Zwei Makros für unbestimmte Wiederholung, IRPC, werden benutzt. Diese Makros erleichtern die Programmierung von Befehlsfolgen, die sich nur durch einen Buchstaben unterscheiden.

Die Makros, die wir bisher benutzt haben, wurden am Beginn des Programms oder in einer separaten Makrobibliothek definiert. Man konnte dann diese Makros mit ihren Parametern überall im Programm aufrufen. Anders ist es mit den Wiederhol-Makros. Sie werden dort im Programm definiert, wo sie verwendet werden.

Diese Makros beginnen mit einem der Namen REPEAT, IRP oder IRPC und enden wie gewöhnlich mit ENDM. Es folgt das erste der beiden Wiederhol-Makros:

```
;Inline-Makro zum Zählen von Control-Zeichen
        IRPC    X?,BDSTVX
        LOCAL   AROUND
CTR&X?  EQU     '&X?' - '@'
        CPI     CTR&X?
        JNZ     AROUND
        LDA     CNT&X?
        INR     A
        STA     CNT&X?
        JMP     LINE
CNT&X?: DB      0
AROUND:
        ENDM
```

Dies Makro erzeugt sechs geringfügig unterschiedliche Befehlsfolgen. Der erste Parameter, X?, ist eine formale Variable. Der zweite Parameter enthält die Referenzparameter – sechs Zeichen, die Buchstaben B, D, S,

T, U und X. Das Makro wird also sechsmal durchlaufen. Das erste Mal wird X? durch B ersetzt. Das Ampersand (kaufmännisches Und) ist ein Verbindungszeichen. Es verbindet den Text CTR mit der formalen Variablen X?. Beim ersten Durchlauf wird die folgende Passage generiert:

```
0002+=          CTRB    EQU     'B' - '@'
0396+FE02               CPI     CTRB
0398+C2A603             JNZ     ??0037
039B+3AA503             LDA     CNTB
039E+3C                 INR     A
039F+32A503             STA     CNTB
03A2+C36903             JMP     LINE
03A5+00         CNTB:   DB      0
```

Weitere fünf ähnliche Sequenzen folgen dann. Das Ampersand wird jedesmal nach dem Einsetzen des aktuellen Parameters entfernt. Manche Assembler belassen das Ampersand in der endgültigen Liste. Der Befehl JNZ ??0037 bewirkt einen Sprung zum Ende der Passage bei Adresse 3A6. Beachten Sie, daß Control-B, eine binäre Zwei, durch Subtraktion eines @-Zeichens (40 hex, populär „Klammeraffe" genannt) erhalten wird. Die anderen Control-Zeichen werden ähnlich erzeugt.

ZUSAMMENFASSUNG

In diesem Kapitel haben wir erfahren, wie mit dem Dateisteuerblock Diskettendateien beschrieben und behandelt werden, und wie man eine Datei liest. Wir haben Makros geschrieben, um eine Fehlermeldung auszugeben, eine existierende Datei zu eröffnen, die DMA-Adresse zu setzen, einen Sektor zu lesen, mit einem Programm einen Dateinamen einzulesen und ein Programm abzubrechen. Wir haben auch das Makro IRPC kennengelernt.

Wir haben mehrere Programme geschrieben, die den Gebrauch der Makros verdeutlichen. SHOW zeigt eine ASCII-Datei auf der Konsole an; DUMP gibt eine COM-Datei hexadezimal und in ASCII aus; ADDRESS kopiert die Adresse aus einem Brief auf einen Umschlag; PAIR überprüft eine Textdatei auf paarige Steuerzeichen. Im nächsten Kapitel werden wir uns mit dem Schreiben von Dateien beschäftigen.

Das Verzeichnis unserer Makrobibliothek sollte jetzt so aussehen:

```
;;Makros in dieser Bibliothek                    Flags
;;ABORT    MACRO  CHAR                          CIFLAG,COFLAG
;;AMBIG    MACRO  OLD,NEW                       (keine)
```

```
;;COMPAR  MACRO  FIRST,SECOND,BYTES  CMFLAG
;;COMPRA  MACRO  FIRST,SECOND,BYTES  CMFLAG
;;CPMVER  MACRO                      (keine)
;;CRLF    MACRO                      (keine)
;;ENTER   MACRO                      (keine)
;;ERRORM  MACRO  TEXT,WHERE          COFLAG,CRFLAG,PRFLAG
;;EXIT    MACRO  SPACE?              (keine)
;;FILL    MACRO  ADDR,BYTES,CHAR     FLFLAG
;;GFNAME  MACRO  FCB                 FNFLAG,FLFLAG,REFLAG,
                                     COFLAG,CRFLAG,PRFLAG
;;HEXHL   MACRO  POINTR              HXFLAG,RCFLAG
;;LCHAR   MACRO  PAR                 LOFLAG
;;MOVE    MACRO  FROM,TO,BYTES       MVFLAG
;;OPEN    MACRO  POINTR,WHERE        OPFLAG,COFLAG,PRFLAG
;;OUTHEX  MACRO  REG                 CXFLAG,COFLAG
;;PCHAR   MACRO  PAR                 COFLAG
;;PRINT   MACRO  TEXT,BYTES          PRFLAG,COFLAG
;;READB   MACRO  BUFFR               RCFLAG
;;READCH  MACRO  REG                 CIFLAG,COFLAG
;;READS   MACRO  POINTR,STAR         RDFLAG,COFLAG
;;SBC     MACRO                      (keine)
;;SETDMA  MACRO  POINTR              DMFLAG
;;SYSF    MACRO  FUNC,AE             (keine)
;;UCASE   MACRO  REG                 (keine)
;;UPPER   MACRO  REG                 (keine)
;;VERSN   MACRO  NUM                 (keine)
```

Kapitel 7

Das Schreiben von Diskettendateien mit dem BDOS

EINFÜHRUNG

Durch die Programme, die wir bisher geschrieben haben, wurde die Diskette nicht verändert. Wir haben Programme zum Lesen, aber nicht zum Schreiben von Dateien entwickelt. In diesem Kapitel werden wir die Makros MAKE, UNPROT, PFNAME, DELETE, SETUP2, RENAME, CLOSE und WRITES schreiben, mit denen man Dateien erstellen und ändern kann. Außerdem werden wir mehrere Programme schreiben: COPY zum Duplizieren von existierenden Dateien, CRYPT zum Verschlüsseln von Dateien, RENAME zum Umbenennen von Dateien und DELETE zum Löschen von Dateien. Beachten Sie, daß wir die Namen RENAME und DELETE sowohl für Programme als auch für Makros benutzen. Da CP/M den Programmnamen und der Assembler den Makronamen benutzt, entsteht daraus kein Konflikt. Wir wollen mit dem Makro MAKE beginnen.

EIN MAKRO ZUM ERZEUGEN EINER NEUEN DATEI

In Kapitel 6 haben wir gesehen, daß man eine existierende Datei mit der BDOS-Funktion 15 eröffnen muß, bevor man sie lesen kann. Um eine neue Datei zu erstellen, müssen wir die BDOS-Funktion 22 aufrufen. Der erste Teil eines Dateisteuerblocks (FCB) wird wie für die Eröffnung einer existierenden Datei im Speicher vorbereitet. Das erste Byte dieses Speicher-FCB bezeichnet das Laufwerk. Eine 0 spezifiziert das Standardlaufwerk, eine 1 Laufwerk A, eine 2 Laufwerk B etc. Dateiname und Dateityp belegen die nächsten 11 Bytes. Das DE-Register wird mit der Adresse des FCB geladen, das C-Register mit dem Wert 22, und ein CALL der Adresse 5 führt die Funktion dann aus.

Mit dem Makro MAKE in Abb. 7.1 wird für die Erstellung einer neuen Datei ein FCB für das Disketten-Inhaltsverzeichnis vorbereitet. Der

Parameter POINTR bezeichnet die Adresse des Speicher-FCB. Tragen Sie das Makro MAKE in die Bibliothek ein.

Einer oder mehrere Blöcke von Sektoren sind für das Disketten-Inhaltsverzeichnis reserviert. Die Anzahl ist fest, variiert aber bei den verschiedenen Diskettenformaten. Die Anzahl der darin enthaltenen Einträge ist auch fest, da vier FCBs in einem 128-Byte-Sektor gespeichert werden können. Es kann vorkommen, daß alle Einträge belegt sind. Daher prüft die BDOS-Funktion 22, ob noch genügend Platz für einen neuen Eintrag vorhanden ist.

Nach Rückkehr von der BDOS-Funktion 22 hat das A-Register den Wert FF hex, wenn das Inhaltsverzeichnis voll ist. Das Makro MAKE prüft daher das A-Register. Wenn das Verzeichnis voll ist, wird eine Fehlermeldung ausgegeben, und das Programm setzt bei DONE fort. Durch das Flag MKFLAG wird sichergestellt, daß die Subroutine MAKE2? nur einmal generiert wird. Mit dem Makro SYSF wird der BDOS-Aufruf durch-

```
MAKE    MACRO   POINTR
;;(Tagesdatum)
;;Inline-Makro zum Erstellen einer neuen Datei.
;;POINTR zeigt auf Dateisteuerblock.
;;Extent und laufende Satznummer werden gelöscht.
;;Benötigte Makros: SYSF, ERRORM
;;
        LOCAL   AROUND
        LXI     D,POINTR
        XRA     A               ;null
        STA     POINTR+12       ;Extent
        STA     POINTR+32       ;laufende Satznummer
        CALL    MAKE2?
        INR     A               ;0=ok, FF heißt Fehler
        JNZ     AROUND
        ERRORM  'Kein Platz im Inhaltsverzeichnis', DONE
        IF      NOT MKFLAG
MAKE2?: SYSF    22              ;erzeuge neue Datei
MKFLAG  SET     TRUE            ;nur eine Kopie
        ENDIF
AROUND:                         ;;MAKE
        ENDM
```

Abb. 7.1: Das Makro MAKE zum Erstellen einer neuen Datei

geführt, mit dem Makro ERRORM wird die Meldung ausgegeben, wenn das Verzeichnis voll ist.

Mit dem nächsten Makro ändern wir den Status einer Datei von „Nur-Lesen" in „Lesen/Schreiben".

AUFHEBEN DES DATEISCHUTZES

CP/M-Dateien kann man gegen versehentliches Löschen schützen, indem man sie mit „Nur-Lesen" kennzeichnet. Wenn man eine Datei ändern oder löschen will, muß sie mit „Lesen/Schreiben" gekennzeichnet sein. In CP/M wird dieser Schutz durch Codieren des ersten Zeichens des Dateinamens verwirklicht. Wenn das höchstwertige Bit dieses Zeichens gesetzt ist, gilt die Datei als schreibgeschützt. Ist das Bit 0, kann die Datei geändert oder gelöscht werden.

Lassen Sie uns dies mit dem DDT oder SID untersuchen. Schalten Sie auf Laufwerk A, und bestimmen Sie die Dateiattribute mit dem Kommando

```
STAT *.COM
```

Eine Liste aller COM-Dateien erscheint. Das Symbol R/O steht bei geschützten Dateien (read only – nur Lesen). Das Symbol R/W dagegen zeigt eine ungeschützte Datei (read/write – Lesen/Schreiben) an. Die Liste mag so aussehen:

```
Recs  Bytes  Ext  Acc
   6     2K    1  R/O A:SAVEUSER.COM
   6     2K    1  R/W A:SHOW.COM
  42     6K    1  R/O A:STAT.COM
  10     2K    1  R/O A:SUBMIT.COM
  12     2K    1  R/O A:SYSGEN.COM
```

Für den nächsten Schritt brauchen wir eine geschützte Datei. Falls alle Dateien ungeschützt sind, ändern Sie das Attribut einer dieser Dateien, z.B. STAT selbst, mit dem Kommando

```
STAT STAT.COM $R/O
```

Achten Sie auf die Leerstelle vor dem Dollarzeichen. Überprüfen Sie die Funktion, indem Sie noch einmal STAT eingeben.

Starten Sie den Debugger DDT oder SID ohne weitere Parameter. Wir schreiben nun mit Hilfe des A-Kommandos ein kleines Programm beginnend bei 4000 hex:

```
A4000
4000 LXI D,5C
4003 MVI C,0F
4005 CALL 5
4008 RST 7
```

(Mit einem Extra-Return beenden Sie das Kommando.) Starten Sie das Programm noch nicht. Durch seine Ausführung wird die im FCB ab 5C hex – Adresse im DE-Register – genannte Datei eröffnet. Register C wird mit dem Wert 0F hex (15 dez), der BDOS-Funktion OPEN, geladen. Nach Rückkehr vom BDOS-CALL wird der Debugger durch den Restart 7, über Adresse 38 hex, fortgesetzt.

Wir bauen nun mit dem Debugger den ersten Teil eines FCB bei Adresse 5C hex auf. Geben Sie das Kommando

```
ISTAT.COM
```

Mit dem I-Kommando wird ein Speicher-FCB mit dem Dateinamen STAT.COM für das Standardlaufwerk initialisiert. Schauen Sie sich das Ergebnis mit dem Kommando D50,6F an:

```
0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 53 54 41 .............STA
0060: 54 20 20 20 20 43 4F 4D 00 00 00 00 00 20 20 20 T    COM.....
```

Beachten Sie, daß die vier Zeichen nach dem Dateinamen STAT Leerzeichen sind. Geben Sie nun das Debuggerkommando G4000 ein. Dadurch wird das bei 4000 hex eingegebene Programm gestartet. Das Standardlaufwerk wird angesprochen, und danach wird der Debugger wieder aktiviert. Untersuchen Sie den Speicher wieder mit dem Kommando D50,6F:

```
0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 53 54 41 .............STA
0060: 54 20 20 20 20 C3 4F 4D 00 00 80 2A 06 07 08 00 T    .OM...*....
```

Wir sehen, daß der ASCII-Name sich von

```
STAT COM
```

in

```
STAT .OM
```

geändert hat.

Nach Ausführung der BDOS-Funktion 15 (OPEN) hat CP/M den Dateityp im Speicher-FCB mit dem Dateityp des Disketten-FCB überschrie-

ben. Das erste Zeichen, der Buchstabe C, ist geändert. Sehen Sie auf die hexadezimale Darstellung dieses Zeichens (Adresse 65 hex). Der Originalwert 43 hex ist nun C3 hex. Die Hexadezimal- und ASCII-Werte sind:

```
43 4F 4D COM
C3 4F 4D .OM
```

Der Vergleich der beiden zeigt, daß das höchstwertige Bit für die Anzeige des Schreibschutzes benutzt wurde:

Hex	*Binär*
43	0100 0011
C3	1100 0011

Sie können nun zu CP/M zurückkehren, das Attribut von STAT ändern und die obigen Schritte wiederholen. Der Dateityp wird nun auch nach der OPEN-Funktion COM bleiben.

Wir schreiben nun das Makro UNPROT (in Abb. 7.2), um den Dateischutz einer Datei mit der BDOS-Funktion 30 aufzuheben. Mit dieser BDOS-Funktion kann man die vier Attribute ändern – „Nur-Lesen (R/O), Lesen/Schreiben (R/W), System (SYS) und Directory (DIR). Wir werden sie nur benutzen, um den Schutz aufzuheben. Das Makro löscht das höchstwertige Bit im durch den Parameter POINTR bezeichneten FCB. Das erste Zeichen des Dateityps wird in das A-Register geladen. Das höchstwertige Bit wird durch logisches AND mit dem Operanden 7F hex (01111111) gelöscht. Das Ergebnis wird zurückgeschrieben. Mit dem Makro SYSF wird die BDOS-Funktion 30 aufgerufen, wodurch der Dateityp des Disketten-FCB mit dem des Speicher-FCB gleichgesetzt wird.

Tragen Sie das Makro UNPROT in die Bibliothek ein. Durch das Flag UNFLAG wird die Subroutine UNPR2? nur einmal generiert. Mit dem nächsten Makro kann man den Dateinamen aus einem Speicher-FCB anzeigen.

EIN MAKRO ZUR AUSGABE EINES FCB-DATEINAMENS

Wie wir gesehen haben, enthält der erste Teil eines Speicher-FCB das Laufwerk an der Stelle 0 und den Dateinamen in den Stellen 1 bis 8. Namen mit weniger als acht Zeichen werden mit Leerzeichen aufgefüllt. In den Stellen 9-11 steht der Dateityp. Daher wird der Name LONGNAME.EXT als LONGNAMEEXT gespeichert. Ein kurzer

```
UNPROT  MACRO   POINTR
;;(Tagesdatum)
;;Inline-Makro zum Ändern des Dateiattributs R/O in R/W.
;;POINTR zeigt auf Dateisteuerblock.
;;Benötigtes Makro: SYSF
;;
        LOCAL   AROUND
        LXI     D,POINTR
        LDA     POINTR+9        ;laden von Dateityp
        ANI     7FH             ;setze auf R/W
        STA     POINTR+9        ;speichern am Beginn von
                                Dateityp
        CALL    UNPR2?
        IF      NOT UNFLAG
        JMP     AROUND
UNPR2?:
        SYSF    30              ;setze Dateiattribut
UNFLAG  SET     TRUE            ;nur eine Kopie
        ENDIF
AROUND:                         ;;UNPROT
        ENDM
```

Abb. 7.2: Das Makro UNPROT zum Aufheben des Dateischutzes

Name wie A.TYP wird als A_____TYP gespeichert (der Strich bezeichnet Leerstellen). Da wir gelegentlich den Dateinamen aus einem FCB ausgeben wollen, lassen Sie uns hierfür ein Makro schreiben.

Mit dem Makro PFNAME (in Abb. 7.3) wird der Dateiname aus dem angegebenen FCB in der bei CP/M üblichen Form ausgegeben. Leerzeichen werden entfernt, und ein Punkt wird zwischen dem Namen und dem Typ eingefügt. Die Makros PRINT und PCHAR werden benutzt. Tragen Sie das Makro in Ihre Bibliothek ein.

EIN MAKRO ZUM LÖSCHEN EINER DATEI

Wir haben gesehen, daß das erste Byte des Speicher-FCB mit der Laufwerksnummer, der Disketten-FCB dagegen mit der Benutzernummer beginnt. Um eine Datei zu löschen, muß man das erste Byte des Disketten-FCB auf den Wert E5 hex setzen. Der übrige Teil des FCB und die Datei selbst werden nicht verändert. Es wird hierdurch der von der Datei

```
PFNAME   MACRO    FCB
;;(Tagesdatum)
;;Inline-Makro zum Ausgeben des Dateinamen als
;; FIRST.EXT
;;FCB ist Dateisteuerblock.
;;benötigte Makros: PCHAR, PRINT
;;
         LOCAL    PFNA2?,        PFNA3?
         PUSH     H
         PUSH     B
         MVI      B,8            ;Namenslänge
         LXI      H,FCB + 1      ;Start
PFNA3?:
         MOV      A,M            ;lies Zeichen
         CPI      BLANK
         JZ       PFNA2?         ;Ende
         PCHAR                   ;ausgeben
         INX      H
         DCR      B
         JNZ      PFNA3?
PFNA2?:
         POP      B
         POP      H
         PCHAR    '.'
         PRINT    FCB+9,3        ;Dateityp
                                 ;;PFNAME
         ENDM
```

Abb. 7.3: Das Makro PFNAME zum Drucken des Dateinamens aus einem FCB

belegte Diskettenspeicher für neue Dateien freigegeben. Erst wenn eine neue Datei geschrieben wird, wird dieser Platz verändert.

Mit der BDOS-Funktion 19 löscht man eine Datei. Wir kleiden sie in das Makro DELETE in Abb. 7.4 ein. Im Makro wird zunächst die FCB-Adresse in das DE-Register geladen. Im ersten Zeichen des Dateityps (bei FCB1+9) wird geprüft, ob die Datei schreibgeschützt ist. Ist dies der Fall, wird der Dateiname ausgegeben und über die Tastatur nachgefragt, ob die Datei gelöscht werden soll. Ist die Antwort J, wird der Schutz mit dem Makro UNPROT aufgehoben. Ist die Antwort nicht J, wird das Programm bei WHERE fortgesetzt, falls der zweite Parameter angegeben

wurde. Sonst geht es bei DONE weiter. Beachten Sie, daß das Makro DELETE einige andere Makros benutzt. Tragen Sie dieses Makro in die Bibliothek ein.

```
DELETE   MACRO   POINTR, WHERE
;;(Tagesdatum)
;;Inline-Makro zum Löschen einer Datei.
;;POINTR zeigt auf Dateisteuerblock.
;;Ist die Datei geschützt, weiter bei WHERE oder DONE.
;;
;;benötigte Makros: SYSF, UNPROT, READCH,
;; PFNAME, PRINT, UCASE, CRLF
;;
LOCAL    AROUND, DEL3?
         LXI     D,POINTR
         LDA     POINTR+9
         ANI     80H             ;geschützt?
         JZ      DEL3?           ;nein
         CRLF
         PFNAME  POINTR
         PRINT   ' Schreibgeschützte Datei. Loeschen? '
         READCH
         UCASE
         CPI     'J'
         IF      NOTNUL WHERE
         JNZ     WHERE
         ELSE
         JNZ     DONE            ;quit
         ENDIF
         UNPROT  POINTR
DEL3?:
         CALL    DEL2?
         IF      NOT DEFLAG
         JMP     AROUND
DEL2?:
         SYSF    19              ;lösche Datei
DEFLAG   SET     TRUE            ;nur eine Kopie
         ENDIF
AROUND:                          ;;DELETE
         ENDM
```

Abb. 7.4: Das Makro DELETE zum Löschen einer Datei

Wenn die Datei geschützt ist, gibt es eine Schwierigkeit. Eine ungeschützte Datei kann gelöscht werden, ohne daß sie vorher eröffnet wird, eine geschützte Datei nicht. Wir haben bereits gesehen, daß bei einer geschützten Datei das erste Zeichen des Dateityps verändert ist. Es ist wichtig, daß vor der DELETE-Funktion die Datei eröffnet wird, weil BDOS die Datei sonst nicht findet. Wollen Sie z.B. die Datei FIRST.COM löschen, müssen Sie vorher nach der Datei FIRST..OM suchen. Die OPEN-Funktion bringt den Dateinamen in die von BDOS benötigte Form. Die OPEN-Funktion ist in unserem DELETE-Makro nicht enthalten, aber wir werden in unseren Programmen stets darauf achten, daß wir vor dem DELETE die Datei eröffnen.

UNTERSUCHUNG VON ZWEI DATEISTEUERBLÖCKEN MIT DEM DEBUGGER

Wir wissen bereits, wie CP/M uns beim Aufbau eines Speicher-FCB aus den Parametern der Kommandozeile hilft. Mit dem Debugger DDT oder SID haben wir die Untersuchung gemacht. Lassen Sie uns nun die Untersuchung zweier Parameter in der Kommandozeile fortsetzen. Nehmen Sie die Namen zweier Dateien, die es nicht gibt, weil der Debugger sonst die Datei lädt und den Speicher-FCB löscht. Geben Sie das Kommando

```
A:DDT FIRST.EXT SECOND.TYP
```

oder

```
A:SID FIRST.EXT SECOND.TYP
```

Betrachten Sie das Ergebnis mit dem Kommando

```
D50,9F
```

Der Bildschirm sollte nun so aussehen:

```
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 46 49 52 .............FIR
0060 53 54 20 20 20 45 58 54 00 00 00 00 00 53 45 43 ST   EXT.....SEC
0070 4F 4E 44 20 20 54 59 50 00 00 00 00 00 FF 00 BF OND  TYP........
0080 15 20 46 49 52 53 54 2E 45 58 54 20 53 45 43 4F . FIRST.EXT SECO
0090 4E 44 2E 54 59 50 00 00 00 00 00 00 00 00 00 00 ND.TYP..........
```

Sie sehen, daß der erste Parameter, FIRST.EXT, im FCB bei 5C hex erscheint. Das erste Byte ist null und gibt das Standardlaufwerk an, weil kein Laufwerk angegeben wurde. Der Dateiname erscheint in Großbuch-

staben. Drei Leerzeichen füllen das 8-Byte-Feld auf. Die drei Buchstaben des Dateityps erscheinen danach.

Der zweite Parameter, SECOND.TYP, wurde ähnlich behandelt. Der zweite FCB beginnt bei 6C hex. Der „command tail" enthält beide Parameter und beginnt bei 80 hex.

Gehen Sie mit Control-C zu CP/M; dann geben Sie das Kommando

```
A:DDT B:FIRST.EXT B:SECOND.TYP
```

oder

```
A:SID B:FIRST.EXT B:SECOND.TYP
```

Geben Sie den Speicher wieder mit dem Debugger aus; folgendes Ergebnis sollte erscheinen:

```
0050 00 00 00 00 00 00 00 00 00 00 00 00 02 46 49 52 .............FIR
0060 53 54 20 20 20 45 58 54 00 00 00 00 02 53 45 43 ST   EXT.....SEC
0070 4F 4E 44 20 20 54 59 50 00 00 00 00 00 FF 00 BF OND  TYP........
0080 19 20 42 3A 46 49 52 53 54 2E 45 58 54 20 42 3A . B:FIRST.EXT B:
0090 53 45 43 4F 4E 44 2E 54 59 50 00 00 00 00 00 00 SECOND.TYP......
```

Im Command Tail ist zu sehen, daß das Laufwerk B diesmal angegeben wurde. Außerdem sind die Laufwerksnummern bei 5C und 6C hex nun 2.

Gehen Sie wieder mit Control-C zu CP/M, und geben Sie für den dritten Test das Kommando:

```
DDT FIRST.EXT *.TYP
```

oder

```
SID FIRST.EXT *.TYP
```

Untersuchen Sie den Speicher von 50 bis 9F:

```
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 46 49 52 .............FIR
0060 53 54 20 20 20 45 58 54 00 00 00 00 00 3F 3F 3F ST   EXT.....???
0070 3F 3F 3F 3F 3F 54 59 50 00 00 00 00 00 FF 00 BF ?????TYP........
0080 10 20 46 49 52 53 54 2E 45 58 54 20 2A 2E 54 59 . FIRST.EXT *.TY
0090 50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 P...............
```

Bei diesem Test sieht der erste FCB bei 5C hex aus wie bei den vorigen. Jedoch der Anfang des zweiten FCB bei 6C hex besteht aus Fragezeichen.

Wenn in einem Dateinamen ein Stern auftritt, erweitert CP/M das Feld mit Fragezeichen, dem Wild-Card-Zeichen. Der Command Tail jedoch zeigt weiterhin den Stern.

ERÖFFNEN EINER DATEI, WENN ZWEI DATEINAMEN GEGEBEN SIND

Später in diesem Kapitel werden wir ein Programm zum Duplizieren einer bestehenden Datei schreiben. Wenn man das Kommando

```
COPY FIRST SECOND
```

eingibt, bereitet CP/M automatisch zwei FCBs bei 5C und 6C hex vor. Ein anderes Programm, das wir schreiben, vergleicht zwei Dateien. Das Kommando wird sein:

```
VERIFY FIRST SECOND
```

Die bisher geschriebenen Programme erfordern nur einen einzigen Parameter. Wir haben einen FCB bei 5C hex für die Datei benutzt. Bei zwei Parametern wird es komplizierter.

CP/M hat zwei FCBs bei 5C und 6C hex vorbereitet. Wenn Ihr Programm die erste Datei eröffnet, wird der zweite Dateiname zerstört. Sie erinnern sich, daß der komplette FCB 32 Bytes lang ist. Das Programm baut den ersten Teil auf, und CP/M füllt den Rest auf, wenn die Datei eröffnet wird. Der FCB bei 5C hex wird also nach dem Eröffnen bei 7B hex enden. Die zweite Hälfte des ersten FCB überschreibt den Anfang des zweiten FCB.

Sie können sich mit dem Debugger davon überzeugen. Starten Sie den DDT oder SID ohne Parameter. Dann geben Sie das Kommando

```
ISTAT.COM
```

wie bereits früher in diesem Kapitel. Ein FCB bei 5C hex für die Datei STAT.COM wird daraus entstehen. Der zweite FCB bei Adresse 6C hex wird automatisch mit Leerzeichen gefüllt, da kein zweiter Parameter angegeben wurde. Löschen Sie den zweiten FCB-Bereich auf den Wert 26 hex, dem ASCII-&, mit dem Kommando

```
F6C,7F,26
```

Geben Sie das Ergebnis aus mit dem Kommando

```
D50,7F
```

Das sieht dann so aus:

```
0050: E0 D9 00 FF 00 FF 00 FF 00 FF 00 00 00 53 54 41 .............STA
0060: 54 20 20 20 20 43 4F 4D 00 00 00 00 26 26 26 26 T    COM....&&&&
0070: 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 &&&&&&&&&&&&&&&&
```

Beachten Sie, daß die Und-Zeichen den zweiten FCB bei 6C belegen. Schreiben Sie mit dem Debugger-Kommando A das folgende kleine Programm:

```
A4000
4000 LXI D,5C
4003 MVI C,0F
4005 CALL 5
4008 RST 7
```

Starten Sie das Programm mit G4000. Das Programm eröffnet eine Datei mit der BDOS-Funktion 15. Nach Rückkehr zum Debugger geben Sie das Kommando

```
D50,7F
```

Sie sehen, daß die Und-Zeichen im zweiten FCB verschwunden sind. Die OPEN-Funktion hat die Information im zweiten FCB zerstört.

Um dies zu vermeiden, muß man den zweiten FCB umspeichern, bevor die Datei eröffnet wird. Hierfür schreiben wir das Makro SETUP2 (Abb. 7.5).

```
SETUP2   MACRO
;;(Tagesdatum)
;;Inline-Makro zum Eröffnen zweier Dateien.
;;Eingabe-Datei ist der erste Parameter einer Kommando-
;;zeile. Der Dateisteuerblock ist FCB1 bei 5C hex.
;;Die Ausgabe-Datei ist der zweite Parameter.
;;Der Dateisteuerblock ist zunächst FCB2 bei
;;6C hex. Der Zieldateiname wird in den
;;Makrobereich gespeichert.
;;Wenn nur eine Datei angegeben wird oder beide gleich sind,
;;erhält die zweite Datei den Typ $$$. Das Makro CLOSE
;;benennt die Original-Datei BAK und gibt den Original-
```

Abb. 7.5: Das Makro SETUP2 zur Verarbeitung zweier Dateien

```
;;namen der Zieldatei, wenn S2FLAG wahr ist.
;;
;;andere benötigte Makros: MOVE, OPEN, MAKE, DELETE,
;; ERRORM, AMBIG, COMPAR
;
        LOCAL   AROUND, SET2?, SET3?, SET4?
S2FLAG  SET     TRUE                    ;für das MAKRO CLOSE
;
        LDA     FCB2+1                  ;zweiter Parameter
        CPI     BLANK                   ;vorhanden?
        JNZ     SET4?
;dupliziere Dateinamen und -typ, erhalte Laufwerknamen
        MOVE    FCB1+1, FCB2+1, 11      ;erhalte Laufwerk
SET4?:
        AMBIG   FCB1, FCB2              ;ersetze ??? im Namen
        COMPAR  FCB1, FCB2, 12          ;beide gleich?
        JZ      DUPNM?                  ;ja
SET2?:
        MOVE    FCB2, DFCB, 16          ;neues Ziel
        OPEN    FCB1                    ;Quelldatei
        OPEN    DFCB, SET3?             ;Ziel
SET3?:
        DELETE  DFCB                    ;lösche Datei
        MAKE    DFCB                    ;eine neue
        JMP     AROUND                  ;Fehlertexte
DUPNM?:
        MVI     A,TRUE
        STA     DUPL                    ;setze 'dup'-Flag
        MOVE    '$$$', FCB2+9           ;Quelldatei
        JMP     SET2?                   ;weiter
;
DUPL:   DB      FALSE                   ;'dup'-Flag
;
;Dateisteuerblock für Ziel/datei
;
DFCB:   DS      33                      ;Datei 2 FCB
;
AROUND:                                 ;Fortsetzung Hauptprogramm
                                        ;SETUP2
        ENDM
```

Abb. 7.5: Das Makro SETUP2 zur Verarbeitung zweier Dateien (Forts.)

Das Makro SETUP2 erwartet zwei Parameter in der Kommandozeile - der eine ab 5C hex und der andere ab 6C hex. Wollen wir z.B. eine Datei irgendwie ändern, gibt der erste Parameter den Namen einer bestehenden Datei, der zweite Parameter den Namen der neuen Datei an. Das Makro SETUP2 eröffnet die erste Datei und kreiert einen Verzeichniseintrag für die zweite Datei.

Bei manchen Anwendungen ist es für den Benutzer bequem, nur einen Parameter anzugeben. Angenommen, wir wollen die Datei PAYROLL .AUG verschlüsseln. Die verschlüsselte Datei soll wie die Originaldatei heißen, und die Originaldatei soll PAYROLL.BAK heißen.

Das Makro setzt zu Beginn das Flag S2FLAG. Das Makro CLOSE, das wir später noch schreiben werden, benutzt dieses Flag. Dadurch wird der Assembler veranlaßt, bei der Übersetzung des Makros CLOSE das Flag DUPL zu prüfen.

Das Makro SETUP2 prüft dann, ob ein zweiter Parameter angegeben wurde. Falls nicht, wird der erste FCB in den zweiten ab 6C kopiert. Dann wird der Dateiname auf Fragezeichen überprüft. Wenn ein Stern im Namen auftritt, füllt CP/M den Rest des Feldes mit Fragezeichen auf. Mit dem Makro AMBIG werden dann die Fragezeichen durch die entsprechenden Zeichen des ersten Namens ersetzt.

Im nächsten Schritt wird geprüft, ob die beiden Dateinamen identisch sind. Dies ist natürlich auch der Fall, wenn nur ein Dateiname angegeben wurde. In diesem Fall wird der Dateityp der Zieldatei auf $$$ gesetzt, der CP/M-Standard-Dateityp für temporäre Dateien, und das Flag DUPL wird gesetzt.

Im Makro CLOSE wird das DUPL-Flag geprüft, um zu sehen, ob nur ein Name bzw. zwei gleiche Namen angegeben wurde. In zweiten Fall wird der Typ der ersten Datei BAK, und der Typ der zweiten Datei gleich dem Typ der ersten Datei gesetzt.

Der zweite Parameter wird nun von 6C hex in den Standard-Datei-Steuerblock DFCB umgespeichert, der im Makro SETUP2 liegt. Die Anweisung DS (define storage – definiere Speicher) 33 reserviert 33 Bytes für den FCB. Nun ist alles getan, um die erste Datei zu eröffnen. Das wird mit dem Makro OPEN aus dem vorigen Kapitel getan. Es beendet das Programm mit einer Meldung, falls die Datei nicht existiert.

Der nächste Schritt ist sehr wichtig. Wenn wir eine Datei mit dem Kommando SAVE schreiben, wird eine existierende Datei gleichen Namens überschrieben. Wir erzeugen jedoch ein Disketten-FCB aus einem Speicher-FCB mit der BDOS-Funktion 22. Damit erlaubt CP/M uns, einen doppelten Dateinamen zu erzeugen, und das würde zwei Dateien mit

demselben Namen ergeben. Daher muß man, bevor man eine neue Datei erstellt, sicherstellen, daß es keine weitere Datei mit diesem Namen gibt. Dies macht man am einfachsten mit der BDOS-DELETE-Funktion. Dadurch wird eine Datei gelöscht, falls sie existiert. Existiert keine solche Datei, wird die Funktion ignoriert. (Durch das Löschen wird der Speicher-FCB nicht verändert.) Das Makro SETUP2 löscht daher die als zweiten Parameter angegebene Datei.

EIN MAKRO ZUM UMBENENNEN EINER DATEI

Jede CP/M-Datei ist durch einen oder mehrere FCB-Einträge im Disketten-Inhaltsverzeichnis definiert. Man kann den Namen einer Datei ändern, indem man den FCB ändert. Hierfür gibt es die BDOS-Funktion 23. Durch diese Funktion wird der Disketten-FCB, aber nicht die Datei selbst geändert. Im Programm bereitet man die ersten 12 Bytes eines Speicher-FCB vor und eröffnet dann die Datei mit der BDOS-Funktion 15. Ein Speicher-FCB mit dem neuen Namen wird dann 16 Bytes hinter dem alten Namen aufgebaut. Die Laufwerksnummer für die Originaldatei hat den normalen Wert, 0 für das Standardlaufwerk, 1 für Laufwerk A, etc., das Laufwerk für die neue Datei jedoch ist immer 0. Beginnt der FCB für die alte Datei bei 5C hex, so beginnt der FCB mit dem neuen Namen bei 6C hex.

Mit dem Makro in Abb. 7.6 kann man eine Datei umbenennen. Tragen Sie das Makro RENAME in die Bibliothek ein. Der Parameter POINTR bezeichnet den Speicher-FCB mit dem alten Namen. Der Programmierer muß die Datei eröffnen und 16 Bytes hinter dem alten Namen einen Speicher-FCB mit dem neuen Namen bereitstellen. Jetzt wird das Makro RENAME aufgerufen. Achten Sie darauf, daß der neue Name erst nach dem Eröffnen bereitgestellt wird, da der neue Name sonst durch die OPEN-Funktion zerstört wird.

Die BDOS-Funktion 23 sucht den Disketten-FCB, der dem durch POINTR bezeichneten Speicher-FCB gleicht. Danach ändert sie den Namen in den neuen Namen bei POINTR+16 um.

Im Makro RENAME wird zunächst geprüft, ob die Datei geschützt ist. Ist dies der Fall, wird der Schutz mit dem Makro UNPROT aufgehoben. Mit BDOS wird dann die Datei umbenannt. Das Makro RENAME gibt sowohl den alten als auch den neuen Namen auf der Konsole aus. Ein Pfeil nach rechts zeigt an, daß der alte Name in den neuen geändert wurde. Wird SORT.ASM z.B. in SORT.BAK umbenannt, sehen Sie die folgende Ausgabe:

```
SORT ASM ==> SORT BAK
```

```
RENAME    MACRO     POINTR
;;(Tagesdatum)
;;Inline-Makro zum Umbenennen einer existierenden Datei.
;;POINTR zeigt auf Originalnamen.
;;Neuer Name ist bei POINTR + 10H.
;;benötigte Makros: SYSF, PRINT, UNPROT, CRLF
;;
          LOCAL     AROUND, REN2?
          LXI       D,POINTR
          LDA       POINTR+9
          ORI       80H             ;;Datei R/O?
          JZ        REN2?           ;;nein
          UNPROT    POINTR          ;;Schutz aufheben
REN2?:
          CALL      RENAM?
          CRLF
          PRINT     POINTR+1,11
          PRINT     '==>'
          PRINT     POINTR+11H,11
          IF        NOT RNFLAG
          JMP       AROUND
RENAM?:   SYSF      23              ;Datei umbenennen
RNFLAG    SET       TRUE            ;nur eine Kopie
          ENDIF
AROUND:                             ;;RENAME
          ENDM
```

Abb. 7.6: Das Makro RENAME zum Umbenennen einer Datei

EIN MAKRO ZUM SCHREIBEN EINES SEKTORS AUF DIE DISKETTE

In Kapitel 7 haben wir ein Programm zum Lesen eines Sektors von der Diskette geschrieben. Der Sektor wird in den Standardpuffer ab Adresse 80 hex gespeichert, falls man die DMA-Adresse nicht mit der BDOS-Funktion 26 geändert hat. Die gegenteilige Operation, das Schreiben eines Sektors aus diesem Puffer, ist ähnlich. Sie wird ausgeführt mit der BDOS-Funktion 21. Wiederum beginnt der Pufferbereich bei 80 hex, falls man es nicht mit der BDOS-Funktion 26 ändert.

Tragen Sie das Makro WRITES nach Abb. 7.7 in Ihre Bibliothek ein. Dies Makro hat zwei wahlfreie Parameter. Der erste, POINTR, bezeich-

```
WRITES    MACRO     POINTR,STAR
;;(Tagesdatum)
;;Inline-Makro zum Schreiben eines Sektors.
;;POINTR zeigt auf Dateisteuerblock.
;;STAR ist ein Zeichen, das nach jedem Sektor ausgegeben wird.
;;benötigte Makros: SYSF, PCHAR, ERRORM
;;
          LOCAL     AROUND
          IF        NOT NUL STAR
          PCHAR     STAR
          ENDIF
          IF        NOT NUL POINTR
          LXI       D,POINTR
          ENDIF
          CALL      WRIT2?
          ORA       A               ;setze Flag
          IF        WRFLAG
          JNZ       NROOM?
          ELSE                      ;erstes Mal
          JZ        AROUND          ;ok
NROOM?:
          ERRORM    'Kein Platz auf der Diskette', DONE
;
WRIT2?:   SYSF      21              ;Sektor schreiben
WRFLAG    SET       TRUE            ;nur eine Kopie
          ENDIF                     ;;WRFLAG
AROUND:                             ;;WRITES
          ENDM
```

Abb. 7.7: Das Makro WRITES zum Schreiben eines Sektors

net den FCB, in dem der Dateiname steht. Wenn dieser Parameter fehlt, nimmt das Makro an, daß das DE-Register bereits mit der FCB-Adresse geladen ist.

Der zweite Parameter, STAR, ist ein ASCII-Zeichen, das nach dem Schreiben jedes Sektors auf dem Bildschirm ausgegeben wird. Dadurch kann man die Operation beim Schreiben mehrerer Sektoren besser verfolgen. (Allerdings wird, wie wir schon beim Makro READS gesehen haben, der Ablauf dadurch verlangsamt.) Falls auf der Diskette kein Speicherplatz mehr zur Verfügung steht, wird mit dem Makro ERRORM eine entsprechende Meldung ausgegeben.

EIN MAKRO ZUM SCHLIESSEN EINER DATEI

Bei der Erstellung einer Datei werden Sektoren nacheinander aus dem Speicher auf die Diskette geschrieben. Nach dem Schreiben eines jeden Sektors wird im Speicher-FCB notiert, wo sich der Sektor befindet. Der Disketten-FCB dagegen wird noch nicht verändert. Nach dem Schreiben des letzten Sektors muß man die Datei mit der BDOS-Funktion 16 schließen. Dadurch wird der Disketten-FCB aus dem Speicher übernommen.

Mit dem Makro CLOSE in Abb. 7.8 kann man eine Datei schließen. Es kann für sich allein benutzt werden oder in Verbindung mit dem Makro SETUP2. Insbesondere, wenn nur eine Ausgangsdatei, nicht aber eine Zieldatei im Kommando angegeben wurde, wird das Makro CLOSE alles nötige erledigen. Ist der Name der Original-Datei z.B.

```
COPY.ASM
```

erzeugt das Makro SETUP2 die temporäre Datei COPY.$$$. Das Makro CLOSE löscht dann die Datei COPY.BAK, falls vorhanden. Dann wird COPY.ASM in COPY.BAK umbenannt; und schließlich wird COPY.$$$ in COPY.ASM umbenannt.

Tragen Sie das Makro CLOSE in Ihre Bibliothek ein. Dieses Makro benutzt sieben weitere Makros: SYSF, OPEN, PRINT, MOVE, DELETE und RENAME.

```
CLOSE   MACRO   POINTR
;;(Tagesdatum)
;;Inline-Makro zum Schließen einer neuen Datei.
;;POINTR zeigt auf Dateisteuerblock.
;;Wenn die Datei nicht existiert, weiter bei DONE.
;;Wenn S2FLAG von SETUP2 wahr ist, wird geprüft, ob
;;'dup'-Flag DUPL gesetzt ist. Ändere
;;Quell-Datei in BAK und neue Datei in Orig. Namen.
;;Setze S2FLAG zu Beginn auf 'falsch':
;;Anwendung: CLOSE DFCB
;;
;;benötigte Makros: SYSF, ERRORM, OPEN,
;; PRINT, MOVE, DELETE, RENAME
;;
        LOCAL   AROUND, CLOSE3
```

Abb. 7.8: Das Makro CLOSE zum Schließen einer Datei

```
        IF      NOT NUL POINTR
        LXI     D,POINTR
        ENDIF
        CALL    CLOS2?
        INR     A                       ;FF hex heißt Fehler
        IF      NOT S2FLAG              ;SETUP2 Makro
        JNZ     AROUND                  ;ok
        ELSE
        JZ      CLOS3?
        LDA     DUPL                    ;Namen duplizieren?
        ORA     A
        JZ      AROUND                  ;nein
        MOVE    'BAK', FCB1+10H+9
        MOVE    FCB1+9,
                DFCB+10H+9,3            ;orig. Dateityp
        MOVE    FCB1, FCB1+10H, 9
        MOVE    DFCB, DFCB+10H, 9
        DELETE  FCB1+10H                ;BAK-Name falls vorh.
        RENAME  FCB1                    ;Orig. in BAK
        RENAME  DFCB                    ;$$$ in Orig.
        MOVE    'BAK', FCB1+9           ;restauriere Namen
        OPEN    FCB1
        JMP     AROUND
        ENDIF                           ;S2FLAG
        IF      NOT CLFLAG              ;nur eine Kopie
CLOS3?: ERRORM  'Datei nicht gefunden?', DONE
CLOS2?: SYSF    16                      ;Datei schließen
CLFLAG  SET     TRUE                    ;nur eine Kopie
        ENDIF   ;CLFLAG
AROUND:                                 ;;CLOSE
        ENDM
```

Abb. 7.8: Das Makro CLOSE zum Schließen einer Datei (Forts.)

DUPLIZIEREN EINER DATEI

Wir sind jetzt soweit, daß wir ein Programm zum Kopieren von Dateien schreiben können. Unsere umfangreiche Makrobibliothek macht uns diese Aufgabe einfacher. Dieses Programm ist eigentlich nicht besonders sinnvoll, da man hierfür das CP/M-Programm PIP benutzen kann. Es wird jedoch der Ausgangspunkt für weitere Programme sein, wie das Verschlüsseln einer Datei.

Unser COPY-Programm erwartet zwei Parameter in der Kommandozeile – eine Ausgangsdatei und eine Zieldatei. Das Kommando könnte z.B.

```
COPY FIRST SECOND
```

sein. Das Programm erstellt damit eine neue Datei namens SECOND, die eine exakte Kopie der Datei FIRST ist. Beachten Sie, daß dies Kommando „natürlicher" ist als das PIP-Kommando. Die Ausgangsdatei wird erst angegeben, gefolgt von der Zieldatei. Auch wird kein Gleichheitszeichen zwischen den Dateinamen benötigt.

In Kapitel 6 haben wir bereits gesehen, daß man eine Datei eröffnen muß, bevor man sie benutzt. Wir brauchen also eine Anweisung, mit der wir die Datei FIRST eröffnen.

Die Zieldatei muß anders behandelt werden als die Ausgangsdatei. Der Programmierer muß sicherstellen, daß eine Datei mit dem Namen SECOND nicht schon existiert. Falls dies der Fall ist, muß sie gelöscht werden.

Im Hinblick auf mögliche Erweiterungen unseres COPY-Programms wollen wir die Möglichkeit eines einzelnen Parameters in Betracht ziehen, wie z.B.

```
COPY FIRST.EXT
```

Hierbei ist die angegebene Datei sowohl Ausgang als auch Ziel. Eine temporäre Zieldatei für die Aufnahme des Ergebnisses wird erzeugt. Bei Programmende wird der Dateityp der Ausgangsdatei in BAK geändert, und der temporären Zieldatei wird der Name der Ausgangsdatei gegeben.

Erstellen Sie eine Quelldatei nach Abb. 7.9, und nennen Sie sie COPYS .ASM oder COPYS.MAC entsprechend dem verwendeten Assembler. (COPYS steht für copy sector – kopiere Sektor.) Sie können auch eines der Programme aus dem vorigen Kapitel verwenden und es nach Abb. 7.9 abändern.

Das Programm besteht hauptsächlich aus den Definitionen der Symbole und Flags. Die eigentlichen Anweisungen zum Kopieren finden sich in den letzten 10 bis 12 Zeilen des Programms. Assemblieren und testen Sie es. Kopieren Sie unter Verwendung von COPYS das eben erstellte Quellprogramm. Überprüfen Sie aber vorher mit STAT, ob noch genügend Speicherplatz auf der Diskette vorhanden ist. Geben Sie das Kommando

```
COPYS COPYS.ASM CRYPT.*
```

```
TITLE   'Kopiere Datei sektorweise'
;
;(Tagesdatum)
;
;Anwendung: COPYS QUELLE ZIEL
;
FALSE       EQU         0
TRUE        EQU         NOT FALSE
;
BOOT        EQU         0
BDOS        EQU         5           ;BDOS-Eingang
TPA         EQU         100H
;
FCB1        EQU         5CH         ;erster Dateiname
FCB2        EQU         6CH         ;zweiter Dateiname
DBUFF       EQU         80H         ;Standardpuffer
;
;Setze Flags im Hauptprogramm, damit von einigen
;Subroutines nur eine Kopie generiert wird.
;Die Flags müssen vor dem MACLIB-Call gesetzt werden.
;
CIFLAG      SET         FALSE       ;Eingabe Konsole Zeichen
CLFLAG      SET         FALSE       ;Datei schließen
CMFLAG      SET         FALSE       ;Vergleich
COFLAG      SET         FALSE       ;Ausgabe Konsole Zeichen
CRFLAG      SET         FALSE       ;neue Zeile
DEFLAG      SET         FALSE       ;lösche Datei
MKFLAG      SET         FALSE       ;erzeuge neue Datei
MVFLAG      SET         FALSE       ;Blockverschiebung
OPFLAG      SET         FALSE       ;eröffne Datei
PRFLAG      SET         FALSE       ;Ausgabe Konsole
RDFLAG      SET         FALSE       ;lies Sektor
RNFLAG      SET         FALSE       ;umbenenne Datei
S2FLAG      SET         FALSE       ;SETUP2 Makro
UNFLAG      SET         FALSE       ;Schutz aufheben
WRFLAG      SET         FALSE       ;schreibe Sektor
;Ende der Flags
;
            MACLIB      CPMMAC
;
```

Abb. 7.9: Das Programm COPYS zum Duplizieren einer Datei

```
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum).COPYS '
        SETUP2                  ;Eingabe- und Ausgabe-
                                Dateien
COPY:                           ;Datei 1 nach Datei 2
        READS   FCB1,'*'        ;lies Sektor
        JNZ     EOFILE          ;fertig
        ABORT   ESC             ;Abbruch?
        WRITES  DFCB,'#'        ;schreibe Sektor
        JMP     COPY            ;nächsten Sektor
EOFILE:
        CLOSE   DFCB            ;Ziel-Datei
DONE:
        EXIT
;
        END     START
```

Abb. 7.9: Das Programm COPYS zum Duplizieren einer Datei (Forts.)

Sehen Sie sich die Kopie mit SHOW an (wir haben dieses Programm in Kapitel 6 geschrieben) oder mit dem CP/M-Kommando TYPE. Löschen Sie die Kopie nicht; wir werden sie im nächsten Abschnitt benutzen.

Nach dem Start von COPYS wird ein Sektor gelesen und ein Stern (*) ausgegeben. Dieser Sektor wird dann in die neue Datei geschrieben, und ein #-Zeichen wird ausgegeben. Diese beiden Zeichen werden wechselweise auf dem Bildschirm erscheinen und dadurch den Fortgang des Programms andeuten. Wechselweises Lesen und Schreiben von Sektoren ist eine nicht besonders effektive Methode zum Kopieren, sie hat jedoch einen Vorteil – die Dateigröße ist nicht durch den verfügbaren Speicher begrenzt. Es wäre schneller, erst die ganze Datei zu lesen, um sie dann zu schreiben. Diese Methode werden wir in Kürze betrachten.

Da wir das Makro ABORT verwendet haben, können Sie jederzeit den Kopiervorgang durch Drücken der Escape-Taste abbrechen. Die neue Datei wird in diesem Fall jedoch nicht erstellt. Es gibt zwar einen Verzeichniseintrag, jedoch ist die Datei leer, da wir die CLOSE-Funktion nicht ausgeführt haben.

VERSCHLÜSSELUNG EINER ASCII-DATEI

Mit wenigen Modifikationen können wir das eben geschriebene Kopierprogramm in ein Codierprogramm verwandeln. So ein Programm kann sehr nützlich sein. Wenn Sie z.B. einen Computer in einem öffentlichen Raum aufgestellt haben, möchten Sie die Geheimhaltung einiger Dateien (wie Gehalts- oder Personaldateien) sicherstellen. Wenn diese Dateien verschlüsselt sind, können Sie von jemandem, der nicht weiß, wie sie entschlüsselt werden, nicht gelesen werden.

Benutzen Sie die Kopie des im vorigen Abschnitt erstellten Programms, und ändern Sie den Namen in CRYPT.ASM oder CRYPT.MAC. Ändern Sie das Programm nach Abb. 7.10 ab.

```
TITLE   'Encrypt file with XOR'
;
;Feb 8.0, 1982
;
;Usage ENCRYPT SOURCE DESTINATION
;
FALSE     EQU     0
TRUE      EQU     NOT FALSE
;
BOOT      EQU     0
BDOS      EQU     5           ;BDOS-Eingang
TPA       EQU     100H
FCB1      EQU     5CH         ;erster Dateiname
FCB2      EQU     6CH         ;zweiter Dateiname
DBUFF     EQU     80H         ;Standardpuffer
;
;Setze Flags im Hauptprogramm, damit von einigen
;Subroutines nur eine Kopie generiert wird.
;Die Flags müssen vor dem MACLIB-Call gesetzt werden.
;
CIFLAG    SET     FALSE       ;Eingabe Konsole Zeichen
CLFLAG    SET     FALSE       ;Datei schließen
CMFLAG    SET     FALSE       ;Vergleich
COFLAG    SET     FALSE       ;Ausgabe Konsole Zeichen
```

Abb. 7.10: Das Programm CRYPT zum Verschlüsseln einer Datei mit dem XOR-Befehl

```
CRFLAG    SET     FALSE     ;neue Zeile
DEFLAG    SET     FALSE     ;lösche Datei
FLFLAG    SET     FALSE     ;Füllzeichen
FNFLAG    SET     FALSE     ;lies Dateinamen
MKFLAG    SET     FALSE     ;erzeuge neue Datei
MVFLAG    SET     FALSE     ;Blockverschiebung
OPFLAG    SET     FALSE     ;eröffne Datei
PRFLAG    SET     FALSE     ;Ausgabe Konsole
RCFLAG    SET     FALSE     ;lies Konsole
RDFLAG    SET     FALSE     ;lies Sektor
RNFLAG    SET     FALSE     ;umbenenne Datei
S2FLAG    SET     FALSE     ;SETUP2 Makro
UNFLAG    SET     FALSE     ;setze Dateiattribute
WRFLAG    SET     FALSE     ;Sektor schreiben
;Ende der Flags
;
          MACLIB  CPMMAC
;
ORG       TPA
;
START:
          ENTER
          VERSN   '2.08.82.CRYPT '
          LDA     FCB1+1
          CPI     BLANK     ;erster Dateiname?
          JNZ     FIRN      ;ja
          GFNAME  FCB1      ;lies Dateinamen
FIRN:
          SETUP2            ;Eingabe- und Ausgabe-Dateien
;
;lies Schlüssel-Zeichen von Konsole
;
          PRINT   <CR,LF,' Zum Abbruch ESC druecken',
                  CR,LF,LF>
          PRINT   'Zeichen für Verschluesselung: '
          READCH            ;Konsole Zeichen
          ANI     7FH       ;lösche Parity
          CPI     ESC
          JZ      DONE
```

Abb. 7.10: Das Programm CRYPT zum Verschlüsseln einer Datei mit dem XOR-Befehl (Forts.)

```
            STA     KEY         ;speichern
            CRLF
COPY:                           ;Datei 1 nach Datei 2
            READS   FCB1,'*'    ;lies Sektor
            JNZ     EOFILE      ;fertig
            ABORT   ESC         ;Abbruch?
;
;Führe XOR aus mit dem Schlüssel für jedes Byte.
;HL ist Pointer zum Sektor-Puffer.
;
            PUSH    H           ;rette Pointer
            LXI     H,DBUFF     ;Sektor-Puffer
            LDA     KEY         ;Schlüssel
            MOV     B,A         ;nach B
            MVI     C,80H       ;Sektorlänge
CODE:
            MOV     A,M         ;lies Byte
            XRA     B           ;XOR mit Schlüssel
            MOV     M,A         ;speichere Byte
            INX     H           ;erhöhe Pointer
            DCR     C           ;zählen
            JNZ     CODE        ;weiter
            POP     H           ;wiederherstellen
            WRITES  DFCB,'#'    ;schreibe einen Sektor
            JMP     COPY        ;nächsten Sektor
EOFILE:
            CLOSE   DFCB        ;Ziel-Datei
            PRINT   <CR,LF,'    Originaldatei loeschen? '>
            READCH
            UCASE
            CPI     'J'
            JNZ     DONE
            DELETE  FCB1
DONE:
            EXIT
;
KEY:        DS      1           ;Schlüssel-Byte
;
            END     START
```

Abb. 7.10: Das Programm CRYPT zum Verschlüsseln einer Datei mit dem XOR-Befehl (Forts.)

Am Beginn des Befehlsteils haben wir den Aufruf des Makros GFNAME eingefügt. Hierdurch wird der Anwender nach einer Datei gefragt, falls im Kommando keine solche angegeben wurde. Das Makro SETUP2 bereitet zwei FCBs aus den Parametern der Kommandozeile vor. Mit den Makros PRINT und READCH wird das Schlüsselwort angefordert. Dies darf jedes beliebige Tastaturzeichen sein.

Ein Sektor der Ausgangsdatei wird gelesen. Dann wird jedes Byte dieses Sektors durch exklusives ODER mit dem eingegebenen Zeichen verschlüsselt. Dadurch wird die Datei unlesbar. Der Vorteil des exklusiven ODER ist die Einfachheit der Codierung. Durch ein weiteres exklusives ODER mit demselben Zeichen erhält man wieder das Original-Zeichen. Das Verschlüsselungsprogramm ist also gleichzeitig das Entschlüsselungsprogramm.

Nachdem jedes Byte des Sektors ver-(oder ent-)schlüsselt worden ist, wird der Sektor in die Zieldatei geschrieben. Der nächste Sektor der Ausgangsdatei wird gelesen. Das wird so lange wiederholt, bis die ganze Datei verschlüsselt ist oder das Programm durch Drücken der Escape-Taste abgebrochen wird.

Wenn zu Beginn des Programms nur ein Dateiname eingegeben wurde, erhält die Zieldatei den Namen der Ausgangsdatei, und der Dateityp der Ausgangsdatei wird in BAK geändert. Am Ende des Programms wird dem Anwender die Möglichkeit gegeben, die Ausgangsdatei zu löschen.

Verschlüsseln Sie eine Kopie der Quelldatei mit dem Buchstaben M. Der verschlüsselten Datei geben Sie den Namen CRYPT.COD. Die Verschlüsselung geht schneller, wenn die Zieldatei auf ein anderes Laufwerk geschrieben wird. Z.B.:

```
CRYPT CRYPT.ASM B:*.COD
```

Passen Sie auf, daß Sie nicht die Original-Datei löschen, obwohl Sie sie durch einen erneuten CRYPT-Lauf mit demselben Schlüsselzeichen wiederherstellen können:

```
CRYPT B:CRYPT.COD *.ASM
```

Wenn Sie die verschlüsselte Datei mit SHOW oder CP/M-TYPE inspizieren, wird der Bildschirm mit bedeutungsloser Information angefüllt sein. Mit dem Programm DUMP, das wir in Kapitel 6 geschrieben haben, können Sie jedoch das Ergebnis anschauen. Mit dem Kommando DUMP B:CRYPT.COD erhalten Sie etwa dies:

```
Space bar for next screen,<CR>next line,<ESC>to abort
0100 39243921 28446A08 232E3F34 3D396D2B   9$9!(Dj.#.?4=9m+
0110 2421286D 3A243925 6D15021F 6A404776   $!(m:$9%m...j§Gv
0120 4047766D 0B282F6D 6D75637D 616D7C74   §Gvm.(/mmucüamöt
0130 757F4047 76404776 6D183E2C 2A28776D   u.§Gv§Gvm.°,*(wm
0140 08030E1F 141D196D 6D1E0218 1F0E086D   .......mm......m
0150 6D09081E 1904030C 19040203 40477640   m..........§Gv§
0160 472B2C21 3E284428 3C38447D 4047393F   G+,!°(D(µ8Dü§G9?
0170 38284428 3C384423 22396D2B 2C213E28   ..1.....!
0180 40477640 472F2222 3944283C 38447D40   §Gv§G/""9D(µ8Dü§
```

Untersuchen Sie aber andererseits die Original-Datei mit dem Kommando DUMP CRYPT.ASM, sehen Sie etwa folgendes:

```
Space bar for next screen,<CR>next line,<FSC>to abort
0100 5449544C 45092745 6E637279 70742066   !IILL.'Encrypt f
0110 696C6520 77697468 20584F52 270D0A3B   ile with XOR'..;
0120 0D0A3B20 46656220 20382E30 2C203139   ..; Feb  8.0, 19
0130 38320D0A 3B0D0A3B 20557361 67653A20   82 ..;..; Usage:
0140 454E4352 59505420 20534F55 52434520   ENCRYPT  SOURCE
0150 20444553 54494E41 54494F4E 0D0A3B0D    DESTINATION..;.
0160 0A46414C 53450945 51550930 0D0A5452   .FALSE.EQU.0..TR
0170 55450945 5155094E 4F542046 414C5345   UE.EQU.NOT FALSE
0180 0D0A3B0D 0A424F4F 54094551 5509300D   ..;..BOOT.EQU.0.
```

Bei näherer Betrachtung der ASCII-Verschlüsselung der codierten Datei sehen Sie, daß das kleine m häufig erscheint. Sie erinnern sich, daß als Schlüsselbuchstabe der Buchstabe M verwendet wurde. Es ist offensichtlich nicht schwer, den Schlüsselbuchstaben durch Studium der verschlüsselten Datei herauszufinden.

Wenn größere Sicherheit erforderlich ist, kann man den Vorgang mit einem anderen Schlüsselbuchstaben wiederholen. Tun Sie dies z.B. mit dem großen A. Geben Sie das Kommando CRYPT CRYPT.COD, und betrachten Sie das Ergebnis mit dem Kommando

```
A:DUMP CRYPT.COD
```

Das Ergebnis wird sein:

```
Space bar for next screen,<CR>next line,<ESC>to abort
0100 78657860 69052B49 626F7E75 7C782C6A   xex`i.+Iboßuöx,j
0110 6560692C 7B657864 2C54435E 2B010637   e`i,äexd,TC^+..7
0120 0106372C 4A696E2C 2C34223C 202C3D35   ..7,Jin,,4"µ ,=5
0130 343E0106 37010637 2C597F6D 6B69362C   4°..7..7,Y.mki6,
```

```
0140 49424F5E 555C582C 2C5F4359 5E4F492C   IBO^UÖX,,_CY^OI,
0150 2C48495F 58454D4D 58454342 01063701   ,HI_XEBMXECB..7.
0160 066A6D60 7F690569 7D79053C 0106787E   .jm`.i.iüy.µ..x
0170 79690569 7D790562 63782C6A 6D607F69   ..1.............!
0180 01063701 066E6363 7805697D 79053C01   ..7..nccx.iüy.µ.
```

Diese Datei wurde das erste Mal mit dem Buchstaben M, das zweite Mal mit dem Buchstaben A verschlüsselt. Keines dieser beiden Zeichen dominiert in der ASCII-Darstellung. Diese Datei muß natürlich zweimal entschlüsselt werden: einmal mit dem Buchstaben M und dann mit dem Buchstaben A. Die Reihenfolge spielt keine Rolle.

Wenn Sie bei der Verschlüsselung einer Datei nur den Namen der Ausgangsdatei angeben, z.B. CRYPT.COD, zeigt das Programm alle seine Möglichkeiten. Am Ende des Programms gibt es nur eine Datei mit diesem Namen. Während der Ausführung wird auf der Konsole folgende Ausgabe erscheinen:

```
Zum Abbruch ESC drücken
Zeichen für Verschlüsselung: M
*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*
CRYPT COD ==> CRYPT BAK
CRYPT $$$ ==> CRYPT COD
Originaldatei loeschen? J
```

Sowohl CRYPT aus diesem Abschnitt als auch COPY aus dem vorigen Abschnitt geben wechselweise *- und #-Zeichen aus, während Sektoren gelesen und geschrieben werden. Da die Ausgabe dieser Zeichen bei großen Dateien zu zeitaufwendig ist, wollen Sie vielleicht den Parameter bei den Makros READS und WRITES entfernen, wenn Sie mit den Programmen vertraut sind. Dafür ändern Sie:

```
        READS   FCB1,'*'        ;lies einen Sektor
```

in

```
        READS   FCB1            ;lies einen Sektor
```

und

```
        WRITES  DFCB,'#'        ;schreibe eine Sektor
```

in

```
        WRITES  DFCB            ;schreibe einen Sektor
```

Betrachten wir jetzt eine effektivere Methode zum Lesen einer Datei.

KOPIEREN EINER DATEI MIT PUFFERUNG IM SPEICHER

Die eben geschriebenen Programme COPY und CRYPT benutzten die Makros READS zum Lesen eines Sektors und WRITES zum Schreiben eines Sektors. Wechselweises Lesen und Schreiben von einzelnen Sektoren ist eine einfache Methode, Diskettenoperationen durchzuführen, und benötigt keine großen Speicherbereiche. Schneller geht es jedoch, wenn man die Datei vollständig in den Speicher liest und sie dann aus dem Speicher auf die Diskette schreibt. Der Nachteil dieser Methode ist, daß sehr große Dateien nicht in den Speicher geladen werden können, wenigstens nicht vollständig in einem Schritt. Dies ist jedoch keine ernstliche Einschränkung. Die meisten kommerziellen Programme sind klein genug, um in einen Speicher mittlerer Größe zu passen. Weiterhin ist es besser, auch Textdateien auf eine Größe zu begrenzen, die dem Speicher entspricht, da dies die Verarbeitung beschleunigt.

Um ein effektives Kopieren von Dateien zu ermöglichen, müssen wir zwei weitere Makros in unsere Bibliothek aufnehmen. Mit dem einen wird eine Datei vollständig in den Speicher gelesen, mit dem anderen wird aus dem Speicher eine neue Datei erstellt.

Das Lesen einer ganzen Datei in den Speicher

Mit dem Makro LDFILE in Abb. 7.11 wird eine Datei vollständig in den Speicher gelesen. Tragen Sie es in die Bibliothek ein. Das Makro hat drei Parameter. Der erste gibt die Adresse des Speicher-FCB der zu lesenden Datei an. Der zweite ist die Adresse des Speicherpuffers. Der dritte ist das Zeichen, das auf der Konsole nach dem Lesen eines Sektors ausgegeben werden soll.

```
LDFILE    MACRO    FCB, POINTR, CHAR
;;(Tagesdatum)
;;Inline-Makro zum Einlesen einer Datei in den
;;Speicher ab der Adresse in POINTR.
;;POINTR zeigt auf den Anfang des Puffers.
;;Definieren Sie den Puffer am Programmende.
;;HL zeigt auf das Ende des geladenen Programms.
;;Der optionale Parameter CHAR wird nach
;;dem Lesen jedes Sektors ausgegeben.
;;CCP-Bereich kann überlagert werden,
;;FDOS ist aber geschützt.
```

Abb. 7.11: Das Makro LDFILE zum Lesen einer ganzen Datei in den Speicher

```
;;Carry-Flag ist gesetzt, wenn die Datei zu groß ist.
;;DMA Adresse steht am Ende wieder auf 80H.
;;Benötigte Makros: SETDMA, READS
;;
;;Anwendung: LDFILE FCB1, DBUFFP, '*'
;; LDFILE FCB1, BUFFP
;;
LOAD2?:
        LHLD    POINTR
        XCHG                    ;nach DE
        SETDMA                  ;nächsten Sektor
        READS   FCB, CHAR
        JNZ     LOAD3?          ;fertig, wenn nicht-null
        LHLD    POINTR
        LXI     D,80H           ;1 Sektor
        DAD     D               ;Pointer in DE
        SHLD    POINTR          ;rette Pointer
;
;Test, ob Datei den CCP überschreibt
;
        LDA     7               ;FDOS
        SUI     2               ;2 Blöcke zurück
        CMP     H               ;Datei zu groß?
        JNC     LOAD2?          ;nein, weiter
LOAD3?:                         ;fertig
        PUSH    PSW
        SETDMA  80H             ;zurücksetzen
        POP     PSW
                                ;;LDFILE
        ENDM
```

Abb. 7.11: Das Makro LDFILE zum Lesen einer ganzen Datei in den Speicher (Forts.)

Es scheint so, als ob der erste Parameter, FCB, erforderlich ist, aber tatsächlich ist er es nicht. Dieser Parameter wird nur an das Makro READS weitergereicht. Fehlt der aktuelle Parameter, nimmt das Makro READS an, daß das DE-Register bereits mit der Adresse des FCB geladen ist.

Der zweite Parameter des Makros LDFILE ist zwar erforderlich, aber Sie können das Makro ändern, um ihn wahlfrei zu machen. Der dritte Parameter wird auch nur an das Makro READS weitergereicht. Fehlt er, wird nach dem Lesen eines Sektors kein Zeichen ausgegeben.

Die Makros SETDMA und READS werden von dem Makro LDFILE benötigt. Wie wir wissen, liest CP/M Sektoren von der Diskette in einen durch die DMA-Adresse angegebenen Speicherbereich, und dieser Bereich ist nach einem Warmstart automatisch die Adresse 80 hex. Auch wissen wir, daß man die DMA-Adresse mit der BDOS-Funktion 26 auf jede gewünschte Speicheradresse setzen kann.

Ein Programm, das das Makro LDFILE benutzen will, muß den Speicherpuffer am Ende des Programms bereitstellen. Im Makro LDFILE wird die DMA-Adresse zunächst auf diesen Wert gesetzt. Nach dem Lesen eines Sektors wird vom Makro diese Adresse um 80 hex erhöht, die Länge eines Sektors. Dadurch wird die ganze Datei sequentiell in den Speicher gelesen. Am Ende des Lesevorgangs setzt LDFILE die DMA-Adresse wieder auf den Standardwert 80 hex.

Die meisten von uns geschriebenen Programme retten die übergebene Stapeladresse und setzen sie neu. Am Ende des Programms wird der Stapelzeiger wiederhergestellt und ein RET-Befehl ausgeführt. Dies geht schneller als ein Warmstart am Ende eines Programms. Bei größeren Programmen muß man anders vorgehen. Große Programme dürfen den vom CCP belegten Speicher benutzen. In diesem Fall muß jedoch ein Warmstart ausgeführt werden, wenn sich das Programm beendet. Dadurch werden CCP und BDOS erneut geladen. Diesen Weg werden wir hier gehen, wenn wir das Makro LDFILE benutzen, denn der CCP könnte überschrieben worden sein.

Die Anfangsadresse des BDOS finden wir bei Adresse 6 und 7. Z.B. beginnt das BDOS bei 3C00 hex in einem 20K-System, bei FA00 hex in einem 64K-System. Jedes Programm kann die Größe des gerade benutzten CP/M-Systems bestimmen. LDFILE liest das höherwertige Byte der BDOS-Adresse aus dem Speicher 7. Dieser Wert wird beim Lesen eines Sektors mit dem höherwertigen Byte des Speicherzeigers verglichen. LDFILE läßt ein Überschreiben des CCP zu, der Rest des CP/M-Systems wird aber geschützt.

Ist eine Datei so groß, daß das BDOS überschrieben werden würde, hört LDFILE auf zu lesen und setzt das Carry-Flag. Es wird keine Meldung ausgegeben, der Programmierer muß aber nach dem Laden das Carry-Flag testen, um herauszufinden, ob die Datei zu groß ist. Wenden wir uns nun dem gegenteiligen Makro WRFILE zu.

Das Schreiben einer ganzen Datei aus dem Speicher

Das Makro WRFILE in Abb. 7.12 ähnelt dem Makro LDFILE. Die drei Parameter sind auch dieselben wie bei LDFILE. Tragen Sie dieses Makro in Ihre Bibliothek ein.

Nach dem Laden einer Datei in den Speicher zeigt der Zeiger auf das Ende der geladenen Information. Das Makro WRFILE lädt als erstes diesen Wert in das DE-Register. Dann wird der Zeiger auf den Anfang des Speicherpuffers gesetzt. Durch Subtraktion dieser beiden Werte wird die Länge der Datei ermittelt. Für diese 16-Bit-Subtraktion wird das Makro SBC verwendet.

```
WRFILE   MACRO   FCB, POINTR, STAR
;;(Tagesdatum)
;;Inline-Makro zum Schreiben einer Datei aus
;;dem Speicher. Puffer beginnt bei POINTR+2.
;;POINTR markiert Dateiende.
;;Optionales Zeichen STAR wird nach jedem Sektor ausgegeben.
;;Benötigte Makros: WRITE, SBC, SETDMA, CRLF
;;
         LOCAL   WRFIL?,EVEN?
         LHLD    POINTR          ;Ende
         XCHG                    ;nach DE
         LXI     H,POINTR+2      ;Start
         SHLD    POINTR
         XCHG
         SBC     HL,DE           ;Programmlänge
         MOV     A,L
         MOV     L,H             ;oberer Teil
         MVI     H,0
         DAD     H               ;Anzahl Sektoren * 2
         ORA     A               ;ungerade Anzahl von
                                 Sektoren?
         JZ      EVEN?           ;nein
         INX     H
EVEN?:
         PUSH    B
         MOV     B,H
         MOV     C,L
WRFIL?:
         LHLD    POINTR
         XCHG                    ;nach DE
         SETDMA                  ;nächsten Sektor
         WRITES  FCB,STAR
```

Abb. 7.12: Das Makro WRFILE zum Schreiben einer ganzen Datei aus dem Speicher

```
        LHLD    POINTR
        LXI     D,80H           ;1 Sektor
        DAD     D               ;nächste Adresse
        SHLD    POINTR
        DCX     B               ;Anzahl Sektoren
        MOV     A,C
        ORA     B
        JNZ     WRFIL?
        POP     B
                                ;;WRFILE
        ENDM
```

Abb. 7.12: Das Makro WRFILE zum Schreiben einer ganzen Datei aus dem Speicher (Forts.)

Das Kopierprogramm, Version 2

Das Programm in Abb. 7.13 benutzt die Makros LDFILE und WRFILE, um Dateien schneller zu kopieren. Duplizieren Sie das Programm in Abb. 7.9 (COPYS), und geben Sie der neuen Version den Namen COPYB (für buffered copy - gepuffertes Kopieren). Das Kommando hierfür ist

```
COPYS COPYS.ASM COPYB.*
```

Ändern Sie das neue Programm nach Abb. 7.13 ab. Assemblieren und starten Sie es. Testen Sie es, indem Sie eine Kopie von ihm selbst machen.

Sie werden herausfinden, daß diese Version viel schneller läuft als die vorige, die sektorweise kopiert. Sie können die Geschwindigkeit noch steigern, wenn Sie die Zeichen * und # bei den Makros

```
LDFILE FCB1,BUFFP,'*'
```

und

```
WRFILE DFCB,BUFFP,'#'
```

entfernen.

Das Makro LDFILE beendet das Lesen, wenn eine Datei zu groß ist. Sie können dies folgendermaßen testen. Erzeugen Sie eine große Datei mit

```
SAVE 220 DUMMY
```

```
TITLE   'Kopiere Datei mit Pufferung'
;
;(Tagesdatum)
;
;Anwendung: COPYB QUELLE ZIEL
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
BOOT    EQU     0
BDOS    EQU     5               ;BDOS-Eingang
TPA     EQU     100H
;
FCB1    EQU     5CH             ;erster Dateiname
FCB2    EQU     6CH             ;zweiter Dateiname
DBUFF   EQU     80H             ;Standardpuffer
;
;Setze Flags im Hauptprogramm, damit von
;einigen Subroutines nur eine Kopie generiert wird.
;Die Flags müssen vor dem MACLIB-Call gesetzt werden.
;
CIFLAG  SET     FALSE           ;Eingabe Konsolzeichen
CLFLAG  SET     FALSE           ;Datei schließen
CMFLAG  SET     FALSE           ;Vergleich
COFLAG  SET     FALSE           ;Ausgabe Konsolzeichen
CRFLAG  SET     FALSE           ;neue Zeile
DEFLAG  SET     FALSE           ;Datei löschen
DMFLAG  SET     FALSE           ;DMA-Adresse setzen
MKFLAG  SET     FALSE           ;neue Datei erstellen
MVFLAG  SET     FALSE           ;Blockverschiebung
OPFLAG  SET     FALSE           ;Datei eröffnen
PRFLAG  SET     FALSE           ;Konsolausgabe
RDFLAG  SET     FALSE           ;Sektor lesen
RNFLAG  SET     FALSE           ;Datei umbenennen
S2FLAG  SET     FALSE           ;SETUP2 Makro
UNFLAG  SET     FALSE           ;Schutz aufheben
WRFLAG  SET     FALSE           ;Sektor schreiben
;Ende der Flags
;
        MACLIB  CPMMAC
```

Abb. 7.13: Das Programm COPYB zum Kopieren einer Datei mit Pufferung

```
;
;
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum).COPYB'
        SETUP2                  ;Eingabe- und Ausgabe-
                                Dateien
        LDFILE  FCB1,BUFFP,'*'
        JNC     EOFILE          ;Datei ok
        ERRORM  <CR,LF,'?Datei zu gross'>
EOFILE:
        LHLD    BUFFP           ;Pointer
        MVI     M,EOF
        ABORT   ESC
        WRFILE  DFCB,BUFFP,'#'
        CLOSE   DFCB            ;Ziel-Datei
DONE:
        JMP     BOOT            ;Warmstart
OLDSTK: DS      2
        DS      34
STACK:
BUFFP:  DW      BUFFER
BUFFER: DS      1
;
        END     START
```

Abb. 7.13: Das Programm COPYB zum Kopieren einer Datei mit Pufferung (Forts.)

(Die geschriebene Information ist der momentane Speicherinhalt.) Seien Sie sicher, daß auf der Diskette noch genügend Speicherplatz frei ist (etwa 55K Bytes). Versuchen Sie, diese Datei mit dem Kommando

```
COPYB DUMMY
```

zu kopieren. Das Kopierprogramm wird beginnen, die Datei zu lesen, wird jedoch mit der Meldung

```
?Datei zu gross
```

abbrechen.

EIN PROGRAMM ZUM GEPUFFERTEN KOPIEREN EINER DATEI MIT KONTROLLE

Unser Programm benötigt noch zwei weitere Funktionen, bevor wir es ernsthaft benutzen können. Nach dem Kopieren einer Datei sollten wir die neue Datei noch einmal lesen, um sicherzustellen, daß sie korrekt geschrieben wurde. Wir sollten auch die neue Datei gegen Überschreiben schützen, falls die Original-Datei schreibgeschützt war.

Vergleich zweier Dateien

Bevor wir die Kontrollfunktion in das Kopierprogramm einbauen, schreiben wir ein neues Programm. Duplizieren Sie das vorige Programm, und nennen Sie es VERIFY. Ändern Sie es nach Abb. 7.14 ab. Assemblieren und testen Sie es. Das Kommando ist wie bei dem Kopierprogramm, außer daß beide Parameter Quelldateien angeben. Die Reihenfolge spielt hier keine Rolle. Geben Sie ein Kommando mit zwei gleichen Parametern ein:

```
VERIFY VERIFY.ASM VERIFY.ASM
```

Sie sollten die Meldung

```
Dateien sind gleich
```

erhalten.

Dann geben Sie das Kommando

```
VERIFY VERIFY.ASM VERIFY.COM
```

ein. Sie werden die Meldung

```
?Dateien sind verschieden
```

erhalten.

Nach Start dieses Programms wird die erste Datei in den Speicher gelesen. Die zweite wird dann sektorweise in den Standardpuffer ab 80 hex gelesen. Das Programm vergleicht dann diesen Sektor mit dem entsprechenden Sektor der ersten Datei. Die TPA wird also nur für die erste Datei benötigt.

Stern und Fragezeichen können als Zeichen für nicht-eindeutige Dateinamen benutzt werden. Das folgende Kommando ist z.B. gültig:

```
VERIFY VERIFY.ASM *.BAK
```

```
TITLE   'Verifiziere zwei Dateien'
;
;(Tagesdatum)
;
;Anwendung: VERIFY QUELLE ZIEL
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
BOOT    EQU     0
BDOS    EQU     5               ;BDOS-Eingang
TPA     EQU     100H
;
FCB1    EQU     5CH             ;erster Dateiname
FCB2    EQU     6CH             ;zweiter Dateiname
DBUFF   EQU     80H             ;Standardpuffer
;
;Setze Flags im Hauptprogramm, damit von
;einigen Subroutines nur eine Kopie generiert wird.
;Die Flags müssen vor dem MACLIB-Call gesetzt werden.
;
CIFLAG  SET     FALSE           ;Eingabe Konsolzeichen
CMFLAG  SET     FALSE           ;Vergleich
COFLAG  SET     FALSE           ;Ausgabe Konsolzeichen
CRFLAG  SET     FALSE           ;neue Zeile
DMFLAG  SET     FALSE           ;DMA-Adresse setzen
MVFLAG  SET     FALSE           ;Blockverschiebung
OPFLAG  SET     FALSE           ;Datei eröffnen
PRFLAG  SET     FALSE           ;Konsolausgabe
RDFLAG  SET     FALSE           ;Sektor lesen
;Ende der Flags
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum).VERIFY '
        LDA     FCB2+1          ;zweiter Parameter
```

Abb. 7.14: Das Programm VERIFY zum Vergleichen zweier Dateien

```
        CPI     BLANK
        JZ      NOSEC
        AMBIG   FCB1,FCB2
        MOVE    FCB2,DFCB,16    ;Ziel
        OPEN    FCB1
        OPEN    DFCB
        LDFILE  FCB1,BUFFP
        JNC     EOFILE          ;Datei ok
        ERRORM  <CR,LF,'?Datei zu gross'>
EOFILE:
        LHLD    BUFFP           ;Pointer
        MVI     M,EOF
        LXI     H,BUFFER
NSECT:
        ABORT   ESC
        READS   DFCB
        ORA     A               ;weiter bei null
        JNZ     DONE2
        COMPAR  ,DBUFF,128      ;ein Sektor
        JNZ     DIFFER
        LXI     D,80H
        DAD     D               ;nächster Sektor
        JMP     NSECT
DONE2:
        PRINT   <CR,LF,'Dateien sind gleich'>
DONE:
        JMP     BOOT            ;Warmstart
NOSEC:
        ERRORM  <CR,LF,'?Zweite Datei nicht angegeben'>
DIFFER:
        ERRORM  <CR,LF,'?Dateien sind verschieden'>
;
OLDSTK: DS      2
        DS      34
STACK:
DFCB:   DS      33              ;zweite Datei
BUFFP:  DW      BUFFER
BUFFER: DS      1
;
        END     START
```

Abb. 7.14: Das Programm VERIFY zum Vergleichen zweier Dateien (Forts.)

Ein Makro zum Schützen von Dateien

Vielleicht haben Sie bemerkt, daß nach Benutzung unseres Kopierprogramms für eine schreibgeschützte Datei die Kopie nicht geschützt ist. D.h. die neue Datei ist nicht mit „Nur-Lesen" gekennzeichnet. In der nächsten Version werden wir dieses Problem beheben, so daß die neue Datei dieselben Attribute hat wie die alte.

Das Makro PROTEC in Abb. 7.15 kann eine Datei unter Zuhilfenahme der BDOS-Funktion 30 schützen. Wir haben bereits das Makro UNPROT geschrieben, mit dem man, auch mit der BDOS-Funktion 30, den Schutz einer Datei aufheben kann. Sie erinnern sich, daß das höchstwertige Bit des ersten Zeichens des Dateityps das Schutzattribut angibt. Ist es gesetzt, ist die Datei geschützt. Ist es dagegen nicht gesetzt, kann man die Datei ändern oder löschen. Tragen Sie das Makro PROTEC in die Bibliothek ein.

```
PROTEC   MACRO    POINTR
;;(Tagesdatum)
;;Inline-Makro zum Schützen der Datei in FCB bei POINTR.
;;Benötigte Makros: SYSF
;;
         LOCAL    AROUND,PROT2?
         LXI      D,POINTR
         LDA      POINTR+9          ;;Dateityp
         ORI      80H               ;;setze auf R/O
         STA      POINTR+9
         CALL     PROT2?
         JMP      AROUND
PROT2?:
         SYSF     30
AROUND:                            ;;PROTEC
         ENDM
```

Abb. 7.15: Das Makro PROTEC zum Schützen einer Datei

Das Kopierprogramm, Version 3

In der endgültigen Fassung des Kopierprogramms wird eine Datei vollständig in den Speicher gelesen. Dann wird die neue Datei aus dem Speicher geschrieben. Die neue Datei wird sektorweise wieder gelesen und mit dem Speicherinhalt verglichen. Wird ein Unterschied festgestellt,

erscheint die Meldung

```
?Dateien sind verschieden
```

auf der Konsole, und die Konsolglocke ertönt.

In dieser Version wird auch das Schutzattribut der alten Datei auf die neue übertragen. Im Speicher-FCB wird geprüft, ob die Datei geschützt ist. Ist dies der Fall, wird das Schutzattribut der neuen Datei mit dem Makro PROTEC entsprechend gesetzt.

Kopieren Sie das Programm COPYB, und geben Sie ihm den Namen COPYV (copy with verification – Kopieren mit Vergleich). Prüfen Sie mit dem Programm VERIFY, ob die Kopie korrekt ist. Ändern Sie COPYV nach Abb. 7.16 ab. Assemblieren und testen Sie das Programm.

```
TITLE   'Kopiere und verifiziere Datei'
;
;(Tagesdatum)
;
;Anwendung: COPYV QUELLE ZIEL
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
BOOT    EQU     0
BDOS    EQU     5               ;BDOS-Eingang
TPA     EQU     100H
;
FCB1    EQU     5CH             ;erster Dateiname
FCB2    EQU     6CH             ;zweiter Dateiname
DBUFF   EQU     80H             ;Standardpuffer
BEL     EQU     7
;
;Setze Flags im Hauptprogramm, damit von
;einigen Subroutines nur eine Kopie generiert wird.
;Die Flags müssen vor dem MACLIB-Call gesetzt werden.
;
CIFLAG  SET     FALSE           ;Eingabe Konsolzeichen
CLFLAG  SET     FALSE           ;Datei schließen
CMFLAG  SET     FALSE           ;Vergleich
```

Abb. 7.16: Das Programm COPYV zum Kopieren von Dateien mit Vergleich

```
COFLAG  SET     FALSE           ;Ausgabe Konsolzeichen
CRFLAG  SET     FALSE           ;neue Zeile
DEFLAG  SET     FALSE           ;Datei löschen
DMFLAG  SET     FALSE           ;DMA-Adresse setzen
MKFLAG  SET     FALSE           ;neue Datei erstellen
MVFLAG  SET     FALSE           ;Blockverschiebung
OPFLAG  SET     FALSE           ;Datei eröffnen
PRFLAG  SET     FALSE           ;Konsolausgabe
RDFLAG  SET     FALSE           ;Sektor lesen
RNFLAG  SET     FALSE           ;Datei umbenennen
S2FLAG  SET     FALSE           ;SETUP2 Makro
UNFLAG  SET     FALSE           ;Schutz aufheben
WRFLAG  SET     FALSE           ;Sektor schreiben
;Ende der Flags
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum).COPYV '
        SETUP2                  ;Eingabe- und Ausgabe-
                                Dateien
        LDA     FCB1+9
        ANI     80H             ;geschützt
        STA     PROTFL          ;Schutzflag
        LDFILE  FCB1,BUFFP
        JNC     EOFILE          ;Datei ok
        ERRORM  <CR,LF,'?Datei zu gross'>
EOFILE:
        LHLD    BUFFP           ;Pointer
        MVI     M,EOF
        ABORT   ESC
        WRFILE  DFCB,BUFFP
        CLOSE   DFCB            ;Ziel-Datei
;Vergleiche beide Dateien
        OPEN    DFCB
        LXI     H,BUFFER
        SETDMA  DBUFF
```

Abb. 7.16: Das Programm COPYV zum Kopieren von Dateien mit Vergleich (Forts.)

```
NSECT:
        ABORT   ESC
        READS   DFCB
        ORA     A               ;weiter bei null
        JNZ     DONE2
        COMPAR  ,DBUFF,128      ;1 Sektor
        JNZ     DIFFER
        LXI     D,80H
        DAD     D               ;nächsten Sektor
        JMP     NSECT
DONE2:
        LDA     PROTFL          ;geschützt?
        ORA     A
        JZ      DONE            ;nein
        PROTEC  DFCB
DONE:
        JMP     BOOT            ;Warmstart
DIFFER:
        ERRORM  <BEL,'?Dateien sind verschieden'>
PROTFL: DS      1               ;Schutzflag
OLDSTK:
        DS      2
        DS      34
STACK:
BUFFP:  DW      BUFFER
BUFFER: DS      1
;
        END     START
```

Abb. 7.16: Das Programm COPYV zum Kopieren von Dateien mit Vergleich (Forts.)

EIN PROGRAMM ZUM UMBENENNEN EINER DATEI

Dateien kann man mit dem CP/M-Kommando REN umbenennen. Bei diesem Kommando sind jedoch keine nicht-eindeutigen Dateinamen zugelassen. Daher müssen Sie, wenn Sie alle BASIC-Dateien zu Backup-Dateien machen – d.h. den Dateityp von BAS in BAK ändern – wollen, jede Datei einzeln umbenennen.

Mit dem Programm in Abb. 7.17 kann man CP/M-Dateien umbenennen, einzeln oder gruppenweise. Das Kommando ist ähnlich wie bei den ande-

ren Programmen in diesem Kapitel. Z.B. ändert das Kommando

```
RENAME OLDNAME NEWNAME
```

den Namen OLDNAME in NEWNAME. Sollte eine Datei mit dem neuen Namen schon existieren, fragt das Programm, ob es diese Datei löschen soll. Ist diese Datei schreibgeschützt, fragt das Programm noch einmal, ob der Schutz aufgehoben werden darf.

Der eigentliche Nutzen dieses Programms liegt in der Möglichkeit, mehrere Programme mit einem einzelnen Kommando umzubenennen. Z.B. ändert das Kommando

```
RENAME *.BAS *.BAK
```

den Dateityp aller BASIC-Dateien in BAK um. Wenn Sie während dieses Kommandos einen Fehler entdecken, können Sie es mit der Taste ESC abbrechen.

In einem einzelnen RENAME-Kommando können eine Löschfunktion und eine Umbenennfunktion kombiniert werden. Wenn Sie nämlich die Backup-Kopie löschen und die Datei zur Backup-Kopie machen wollen, können Sie dies mit den beiden CP/M-Kommandos

```
ERA MAIN.BAK
REN MAIN.BAK=MAIN.ASM
```

machen. Dasselbe Ergebnis erreichen Sie mit einem einzigen Kommando, wenn Sie unser RENAME-Programm verwenden. Geben Sie das Kommando

```
RENAME MAIN.ASM *.BAK
```

Natürlich fragt RENAME, ob es die Datei MAIN.BAK löschen darf.

Da in diesem Programm das Makro RENAME verwendet wird, wird jede Umbenennung auf der Konsole angezeigt. Mit der OPEN-Funktion wird festgestellt, ob die Original-Datei existiert. Dann wird mit OPEN geprüft, ob die neue Datei nicht existiert. Nach der Umbenennung wird mit der OPEN-Funktion der Name der nächsten Datei festgestellt. Diese Methode ist recht wirksam. Sie versagt jedoch, wenn Sie eine Datei einer Gruppe nicht umbenennen wollen, da jedes nachfolgende OPEN dieselbe Datei findet. Daher beendet sich das RENAME-Programm in diesem Fall.

```
TITLE   'RENAME benennt eine Datei um mit nicht-eindeutigen Angaben'
;
;(Tagesdatum)
;Abbruch des Programms mit ESC.
;Programm beendet sich, wenn es eine Systemdatei findet.
;
;Anwendung:  RENAME OLD NEW
;           RENAME OLD.EXT *.BAK
;           RENAME OLD.EXT NEW.*
;           RENAME OLD.* NEW.*
;           RENAME *.EXT *.BAK
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
BOOT    EQU     0
BDOS    EQU     5               ;BDOS-Eingang
TPA     EQU     100H
FCB     EQU     5CH             ;Dateisteuerblock
;
FCB1    EQU     5CH             ;erster Dateiname
FCB2    EQU     6CH             ;zweiter Dateiname
DBUFF   EQU     80H             ;Standardpuffer
;
;Setze Flags im Hauptprogramm, damit von
;einigen Subroutines nur eine Kopie generiert wird.
;Die Flags müssen vor dem MACLIB-Call gesetzt werden.
;
CIFLAG  SET     FALSE           ;Eingabe Konsolzeichen
CMFLAG  SET     FALSE           ;Vergleich
CRFLAG  SET     FALSE           ;neue Zeile
COFLAG  SET     FALSE           ;Ausgabe Konsolzeichen
DEFLAG  SET     FALSE           ;Datei löschen
MVFLAG  SET     FALSE           ;Blockverschiebung
OPFLAG  SET     FALSE           ;Datei eröffnen
PRFLAG  SET     FALSE           ;Konsolausgabe
RNFLAG  SET     FALSE           ;Datei umbenennen
UNFLAG  SET     FALSE           ;setze Dateiattribute
;Ende der Flags
;
```

Abb. 7.17: Das Programm RENAME zum Umbenennen von Dateien

```
            MACLIB    CPMMAC
;
ORG         TPA
;
START:
            ENTER
            VERSN     '(Tagesdatum).RENAME '
            LDA       FCB1+1
            CPI       BLANK
            JZ        NOSOUR
            LDA       FCB2+1
            CPI       BLANK
            JZ        NODEST
            COMPAR    FCB1+1,FCB2+1,11
            JZ        SAMEN
            COMPAR    '???????????',FCB1+1
            JZ        IMPROP
            COMPAR    '???????????',FCB2+1
            JZ        IMPROP
            XRA       A                    ;null
            STA       FIRSTF               ;lösche Flag
            PRINT     <LF,'Zum Abbruch ESC druecken',CR,LF>
;
;rette Originalparameter
;
            MOVE      FCB1,FCOPY,20H
            MOVE      FCB1,OFCB,20H
NEXTN:                                     ;nächsten Namen
            OPEN      FCB1,FPASS           ;Quelldatei
            ABORT     ESC                  ;Ende?
            MVI       A,0FFH
            STA       FIRSTF               ;Wiederholung
            LDA       FCB1+10              ;System-Datei
            ANI       80H                  ;Bit 7
            JNZ       SYSFIL               ;überspringen
            UNPROT    FCB1
            MOVE      FCB1,OFCB,12         ;Original
            LDA       FCB1                 ;Laufwerknummer
            STA       DFCB
;
```

Abb. 7.17: Das Programm RENAME zum Umbenennen von Dateien (Forts.)

```
;Test auf nicht-eindeutigen Originaldateinamen
;
            COMPAR  FCB1+1,FCOPY+1,8
            JZ      NOQ1                ;nein
            MOVE    FCB1+1,OFCB+1,11    ;alter Name
            MOVE    FCB1+1,DFCB+1,8     ;neuer Name
            MOVE    F2COPY+9,DFCB+9,3   ;Dateityp
            JMP     CHEK2
NOQ1:
;
;Test auf nicht-eindeutigen Originaldateityp
;
            COMPAR  FCB1+9,FCOPY+9,3
            JZ      NOQ3                ;nein
            MOVE    FCB1+1,OFCB+1,11    ;neuer Name
            MOVE    FCB1+9,DFCB+9,3     ;neuer Typ
            MOVE    F2COPY+1,DFCB+1,8   ;Name
            JMP     CHEK2
;
;Test auf nicht-eindeutigen neuen Namen
;
NOQ3:
            AMBIG   FCB1, DFCB
CHEK2:
            OPEN    DFCB, RENAM
            CRLF
            PFNAME  DFCB
            PRINT   ' Existiert bereits. Loeschen? '
            READCH
            UCASE
            CPI     'J'
            JNZ     DONE
            LDA     FCB1                ;Laufwerk
            STA     DFCB
            DELETE  DFCB
RENAM:
            RENAME  OFCB
            MOVE    FCOPY+1,FCB+1,11
            JMP     NEXTN
```

Abb. 7.17: Das Programm RENAME zum Umbenennen von Dateien (Forts.)

```
FPASS:                                          ;Fehlermeldung
                                                falls 1. Durchgang
        LDA     FIRSTF                          ;lies Durchgangs-Flag
        ORA     A                               ;erster Durchgang
        JNZ     DONE                            ;nein
        ERRORM  'Datei nicht gefunden', DONE
NOSOUR:
        ERRORM  'Keine Quell-Datei', DONE
NODEST:
        ERRORM  'Keine Ziel-Datei', DONE
SAMEN:
        ERRORM  'Gleicher Name', DONE
IMPROP:
        ERRORM  'Ungueltiger Name', DONE
;
FIRSTF: DB      0                               ;1. Durchgang
;
SYSFIL:                                         ;Systemdatei gefunden
        CRLF
        PFNAME  FCB
        PRINT   ' ist eine Systemdatei'
DONE:
        EXIT
;
FCOPY:  DS      10H                             ;Originalkommando
F2COPY: DS      10H                             ;mit zweitem Dateinamen
OFCB:   DS      10H                             ;Originalname
DFCB:   DS      10H                             ;neuer Name
;
        END     START
```

Abb. 7.17: Das Programm RENAME zum Umbenennen von Dateien (Forts.)

EIN PROGRAMM ZUM LÖSCHEN EINER DATEI

Mit dem Programm in Abb. 7.18 kann man Dateien löschen. Nichtgeschützte Dateien kann man zwar mit dem CP/M-Kommando ERA löschen, geschützte Dateien jedoch nicht. Mit DELETE kann man auch geschützte Dateien löschen, allerdings wird die Erlaubnis dazu erst eingeholt. Der angegebene Dateiname darf auch Sterne oder Fragezeichen enthalten.

Gibt man als zweiten Parameter Q (für query – nachfragen) an, fragt DELETE bei jeder Datei, ob sie gelöscht werden soll, bei geschützten und bei ungeschützten Dateien. Dies ist sehr nützlich, wenn man ungültige Verzeichniseinträge löschen will. Manchmal enthält ein Dateiname versehentlich nichtdruckbare Zeichen oder Kleinbuchstaben. So eine Datei kann man mit dem CP/M-Kommando ERA nicht löschen. Die ungültige Datei kann man aber mit dem Kommando

```
DELETE *.* Q
```

löschen. Dies ist ein Kommando zum Löschen aller Dateien einer Diskette, allerdings nur mit besonderer Erlaubnis. Ihnen wird jeder Dateiname einzeln ausgegeben, ob die entsprechende Datei geschützt ist oder nicht. Wenn Sie dann mit einem Zeichen ungleich J antworten, wird die Datei nicht gelöscht, und der nächste Dateiname erscheint. Ist Ihre Antwort J, und die Datei ist geschützt, wird um Erlaubnis gebeten, bei dieser Datei den Schutz aufzuheben.

In diesem Programm benutzen wir die BDOS-Funktion 17, um die erste Datei zu finden, und die BDOS-Funktion 18, um die folgenden zu finden. Wenn man die Funktion 18 benutzt, muß man alle Dateinamen in einen separaten Puffer umspeichern. Dann kann das Programm die Dateinamen einzeln verarbeiten.

```
TITLE   'Loesche Datei mit nicht-eindeutigem Namen'
;
;(Tagesdatum)
;
;Anwendung:  DELETE NAME
;           DELETE NAME.EXT
;           DELETE NAME.*
;           DELETE *.EXT
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
BOOT    EQU     0
BDOS    EQU     5               ;BDOS-Eingang
TPA     EQU     100H
```

Abb. 7.18: Das Programm DELETE zum Löschen von Diskettendateien

```
;
FCB1    EQU     5CH             ;erster Dateiname
FCB2    EQU     6CH             ;zweiter Dateiname
DBUFF   EQU     80H             ;Standardpuffer
;
;Setze Flags im Hauptprogramm, damit von
;einigen Subroutines nur eine Kopie generiert wird.
;Die Flags müssen vor dem MACLIB-Call gesetzt werden.
;
CIFLAG  SET     FALSE           ;Eingabe Konsolzeichen
CMFLAG  SET     FALSE           ;Vergleich
CRFLAG  SET     FALSE           ;neue Zeile
COFLAG  SET     FALSE           ;Ausgabe Konsolzeichen
DEFLAG  SET     FALSE           ;Datei löschen
MVFLAG  SET     FALSE           ;Blockverschiebung
OPFLAG  SET     FALSE           ;Datei eröffnen
PRFLAG  SET     FALSE           ;Konsolausgabe
UNFLAG  SET     FALSE           ;Dateischutz aufheben
;Ende der Flags
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum).DELETE '
        LDA     FCB1+1
        CPI     BLANK
        JZ      NOSOUR
        PRINT   <LF,' Zum Abbruch ESC druecken',CR,LF>
        LDA     FCB2+1
        STA     QUERY           ;nachfragen zum Löschen
        COMPAR  '???????????', FCB1+1
        JNZ     ALLNAM
        PRINT   'Alles Loeschen? (J/N) '
        READCH
        UCASE
        CPI     'J'
```

Abb. 7.18: Das Programm DELETE zum Löschen von Diskettendateien (Forts.)

```
        JNZ     DONE
ALLNAM:                         ;lies ersten Dateinamen
        LXI     D,FCB1
        MVI     C,17            ;suche Dateinamen
        CALL    BDOS
        CPI     0FFH            ;gefunden?
        JZ      NOSOUR          ;nein
        CALL    GETNAM
NNAME:                          ;lies nächsten Dateinamen
        LXI     D,FCB1
        MVI     C,18
        CALL    BDOS
        CPI     0FFH            ;mehr?
        JZ      NNAM2           ;nein
        CALL    GETNAM
        JMP     NNAME
NNAM2:
        LXI     H,FNAMES-12
        SHLD    FPNTR
NEXTN:                          ;nächsten Namen
        LHLD    FPNTR           ;Pointer
        LXI     D,12
        DAD     D
        SHLD    FPNTR           ;rette
        MOV     A,M
        CPI     BLANK
        JZ      DONE
        MOVE    , FCB1, 12
        OPEN    FCB1            ;Quell-Datei
        ABORT   ESC             ;Ende?
        LDA     QUERY           ;nachfragen
        CPI     'Q'
        JNZ     NOASK
        PRINT   <CR,LF,' Loeschen '>
        PFNAME  FCB1
        PCHAR   '?'
        PCHAR   BLANK
        READCH
        UCASE
```

Abb. 7.18: Das Programm DELETE zum Löschen von Diskettendateien (Forts.)

```
        CPI     'J'
        JNZ     NEXTN
NOASK:
        DELETE  FCB1,NEXTN
        CRLF
        PFNAME  FCB1
        PRINT   ' geloescht'
        JMP     NEXTN
GETNAM:                         ;Kopiere Dateinamen in
                                Zwischenbereich
        RRC                     ;3 Bits rechts = 5 links
        RRC                     ;0 = 0, 1 = 20H
        RRC                     ;2 = 40H, 3 = 60H
        ANI     60H             ;Maske
        MOV     E,A
        MVI     D,0
        LXI     H,DBUFF
        DAD     D
        XCHG
        LHLD    FPNTR           ;Ziel
        XCHG                    ;nach DE
        MOVE    , ,12
        LXI     H,12
        DAD     D
        SHLD    FPNTR           ;nächsten Namen
        MVI     M,BLANK         ;markiere Ende
        RET
NOSOUR:
        ERRORM  'Keine Quell-Datei', DONE
;
FPNTR:  DW      FNAMES          ;Namen Pointer
QUERY:  DS      1               ;wenn Q, vor Löschen fragen
;
DONE:
        EXIT
;
FNAMES: DS      1               ;Dateinamenstapel
;
        END     START
```

Abb. 7.18: Das Programm DELETE zum Löschen von Diskettendateien (Forts.)

SICHERUNG EINES SPEICHERBEREICHS AUF DIE DISKETTE

In Kapitel 3 haben wird das BIOS so abgeändert, daß die Druckausgabe in einen Speicherbereich geschrieben werden konnte. Wir haben dann diesen Bereich in die TPA umgespeichert und eine Datei mit dem Kommando SAVE erstellt. Wir schreiben nun ein Programm, das diesen Vorgang erleichert. Das in Abb. 7.19 gezeigte Programm kann die Information aus dem Speicherbereich direkt in eine Datei speichern. Der Hauptteil des Programms ist sehr kurz. Es benutzt die Makros ENTER, VERSN und EXIT in der üblichen Weise. Zusätzlich kann mit dem Makro GFNAME ein Dateiname angefordert werden.

Sie erinnern sich, daß wir zwei Zeiger am Beginn des Speicherbereichs hatten. Der erste zeigte auf den Beginn, der zweite auf das Ende des Textes. Der erste Zeiger hat den Wert F000 hex. Mit dem Makro WRFILE wird eine Datei direkt aus dem Speicher erstellt. Das Kommando

```
CACHE Dateiname
```

erzeugt eine Datei mit dem angegebenen Dateinamen, die die Information aus dem Speicherbereich enthält.

```
TITLE   'CACHE zum Schreiben des Speichers auf die Diskette'
;
;(Tagesdatum)
;
;Anwendung: CACHE DISKETTENDATEI
;
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
MPOINT  EQU     0F000H      ;Haupt-Pointer
MMAX    EQU     MPOINT+2    ;Ende des Textes
MBUFF   EQU     MPOINT+2    ;Puffer
;
BOOT    EQU     0           ;Warmstart
BDOS    EQU     5           ;BDOS-Eingang
FCB1    EQU     5CH         ;Eingabe-FCB
DBUFF   EQU     80H         ;Standardpuffer
```

Abb. 7.19: Das Programm CACHE zum Erstellen einer Datei aus dem Speicherpuffer

```
TPA     EQU     100H            ;Anwenderbereich
;
;Setze Flags im Hauptprogramm, damit von
;einigen Subroutines nur eine Kopie generiert wird.
;Die Flags müssen vor dem MACLIB-Call gesetzt werden.
;
CIFLAG  SET     FALSE           ;Eingabe Konsolzeichen
CLFLAG  SET     FALSE           ;Datei schließen
COFLAG  SET     FALSE           ;Ausgabe Konsolzeichen
CRFLAG  SET     FALSE           ;neue Zeile
DEFLAG  SET     FALSE           ;Datei löschen
DMFLAG  SET     FALSE           ;DMA setzen
FLFLAG  SET     FALSE           ;Füllzeichen
FNFLAG  SET     FALSE           ;lies Dateiname
MKFLAG  SET     FALSE           ;neue Datei erstellen
MVFLAG  SET     FALSE           ;Blockverschiebung
OPFLAG  SET     FALSE           ;Datei eröffnen
PRFLAG  SET     FALSE           ;Konsolausgabe Puffer
RCFLAG  SET     FALSE           ;lies Konsol-Puffer
RNFLAG  SET     FALSE           ;Datei umbenennen
S2FLAG  SET     FALSE           ;SETUP2 Makro nicht benutzt
UNFLAG  SET     FALSE           ;Schutz aufheben
WRFLAG  SET     FALSE           ;Datei schreiben
;Ende der Flags
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum).CACHE '
        LDA     FCB1+1
        CPI     BLANK           ;Dateiname?
        JNZ     OP3             ;ja
        GFNAME  FCB1            ;lies Dateiname
OP3:
        DELETE  FCB1            ;existierende Datei
        MAKE    FCB1            ;erstelle neue
```

Abb. 7.19: Das Programm CACHE zum Erstellen einer Datei aus dem Speicherpuffer (Forts.)

```
;
;Erstelle Datei beginnend bei MMAX
;
            WRFILE      FCB1,MMAX
            CLOSE       FCB1
DONE:
            EXIT
;
            END         START
```

Abb. 7.19: Das Programm CACHE zum Erstellen einer Datei aus dem Speicherpuffer (Forts.)

ZUSAMMENFASSUNG

In diesem Kapitel haben wir unsere Makrobibliothek um mehrere wichtige Makros erweitert: MAKE, UNPROT, PFNAME, DELETE, SETUP2, RENAME, CLOSE, WRITES, LDFILE, WRFILE und PROTEC. Des weiteren haben wir mehrere Programme zum Kopieren, Codieren, Kontrollieren, Umbenennen und Löschen von Dateien geschrieben.

Das Verzeichnis unserer Makrobibliothek sollte jetzt so aussehen:

```
;;Makros in dieser Bibliothek                    Flags
;;ABORT   MACRO   CHAR                           CIFLAG,COFLAG
;;AMBIG   MACRO   OLD,NEW                        (keine)
;;CLOSE   MACRO   POINTR                         CLFLAG,COFLAG,CRFLAG,
;;                                               PRFLAG,OPFLAG,MVFLAG,
;;                                               DEFLAG,CIFLAG,UNFLAG,
;;                                               RNFLAG,S2FLAG
;;COMPAR MACRO   FIRST,SECOND,BYTES              CMFLAG
;;COMPRA MACRO   FIRST,SECOND,BYTES              CMFLAG
;;CPMVER MACRO                                   (keine)
;;CRLF    MACRO                                  CRFLAG, COFLAG
;;DELETE  MACRO   POINTR,WHERE                   DEFLAG,CIFLAG,
;;                                               COFLAG,PRFLAG,UNFLAG
;;ENTER   MACRO                                  (keine)
;;ERRORM MACRO   TEXT,WHERE                      COFLAG,CRFLAG,PRFLAG
;;EXIT    MACRO   SPACE?                         (keine)
;;FILL    MACRO   ADDR,BYTES,CHAR                FLFLAG
;;GFNAME MACRO   FCB                             FNFLAG,FLFLAG,RCFLAG,
```

```
;;                                          COFLAG,CRFLAG,PRFLAG
;;HEXHL   MACRO   POINTR                    HXFLAG,RCFLAG
;;LCHAR   MACRO   PAR                       LOFLAG
;;LDFILE  MACRO   FCB,POINTR,CHAR           COFLAG,DMFLAG,RDFLAG
;;MAKE    MACRO   POINTR                    MKFLAG,COFLAG,CRFLAG,
;;                                          PRFLAG
;;MOVE    MACRO   FROM,TO,BYTES             MVFLAG
;;OPEN    MACRO   POINTR,WHERE              OPFLAG,COFLAG,PRFLAG,
;;                                          CRFLAG
;;OUTHEX  MACRO   REG                       CXFLAG,COFLAG
;;PCHAR   MACRO   PAR                       COFLAG
;;PFNAME  MACRO   FCB                       COFLAG,PRFLAG
;;PRINT   MACRO   TEXT,BYTES                PRFLAG,COFLAG
;;PROTEC  MACRO   POINTR                    (keine)
;;READB   MACRO   BUFFR                     RCFLAG
;;READCH  MACRO   REG                       CIFLAG,COFLAG
;;READS   MACRO   POINTR,STAR               RDFLAG,COFLAG
;;RENAME  MACRO   POINTR                    RNFLAG,COFLAG,
;;                                          PRFLAG,CRFLAG
;;SBC     MACRO                             (keine)
;;SETDMA  MACRO   POINTR                    DMFLAG
;;SETUP2  MACRO                             S2FLAG,CIFLAG,COFLAG,
;;                                          CRFLAG,CMFLAG,DEFLAG,
;;                                          MKFLAG,MVFLAG,OPFLAG,
;;                                          PRFLAG,UNFLAG
;;SYSF    MACRO   FUNC,AE                   (keine)
;;UCASE   MACRO   REG                       (keine)
;;UNPROT  MACRO   POINTR                    UNFLAG
;;UPPER   MACRO   REG                       (keine)
;;VERSN   MACRO   NUM                       (keine)
;;WRITES  MACRO   POINTR,STAR               WRFLAG,COFLAG,PRFLAG
;;WRFILE  MACRO   FCB,POINTR                COFLAG,CRFLAG,DMFLAG,
;;                                          WRFLAG
```

Kapitel 8

Das Inhaltsverzeichnis einer CP/M-Diskette

EINFÜHRUNG

In Kapitel 6 haben wir bereits einen kurzen Blick auf das Inhaltsverzeichnis einer CP/M-Diskette geworfen. Wir werden uns nun dieses Verzeichnis näher anschauen, indem wir ein Programm entwickeln, mit dem wir mehrere Verzeichnisangaben anzeigen können. Wir werden die Diskettenparameter, eine erweiterte Verzeichnisliste und die Blockzuordnungstabelle ausgeben können.

DIE DISKETTENPARAMETER

CP/M war ursprünglich für die Arbeit mit Standard-IBM-8-Zoll-Disketten entwickelt worden. Das Format dieser Diskette ist: einfache Schreibdichte, einseitig und soft-sektoriert. Sie hat 77 Spuren mit jeweils 26 Sektoren. Jeder Sektor umfaßt 128 Bytes. Die Blockgröße, d. h. die kleinste auf der Diskette vergebbare Speichermenge, ist 1024 (1K) Bytes. Diese Diskettenparameter waren im CP/M Version 1.4 im BDOS codiert. Daher konnte man bei dieser Version nur schwierig Disketten mit verschiedenen Parametern verwenden.

CP/M Version 2 ist anders organisiert. Die Diskettenparameter stehen im BIOS und nicht im BDOS. Daher kann man bei dieser Version leicht beliebige Diskettentypen verarbeiten. Die Charakteristika jedes unterschiedlichen Laufwerks stehen in einem Bereich, den man Disketten-Parameter-Block (DPB) nennt. Diesen Bereich kann man mit der BDOS-Funktion 31 finden. Lassen Sie uns diesen Bereich näher untersuchen.

Schalten Sie auf das Laufwerk A, und machen Sie mit Control-C einen Warmstart. Starten Sie den Debugger DDT (oder SID), und schreiben Sie mit dem A-Kommando folgendes kurzes Programm:

```
A100
0100 MVI C,1F
0102 CALL 5
0105 RST 7
```

Der erste Befehl lädt eine 31 (1F hex) in das C-Register. Dies ist die BDOS-Funktion, mit der die Diskettenparameter gefunden werden. Der zweite Befehl ruft das BDOS auf, und der dritte Befehl bewirkt die Rückkehr zum Debugger. Starten Sie dieses Programm mit dem Kommando G100. Zeigen Sie nun die Registerinhalte mit dem Debuggerkommando X an. Die Ausgabe ist dann etwa:

```
-Z-E- A=8A B=D400 D=0000 H=D48A S=0100 P=0105 RST 07
```

Der erste Teil zeigt die CPU-Flags. Hier sind das Z-Flag (Z) und das Parity-Flag (E) gesetzt. Die drei Minuszeichen zeigen, daß die übrigen Flags (Carry, Hilfs-Carry und Vorzeichen) gelöscht sind. Die nächsten sechs Daten geben die Inhalte der CPU-Register wieder, einschließlich des Stapelzeigers (S) und des Programmzählers (P). Die letzte Angabe zeigt den letzten ausgeführten Befehl vor Rückkehr zum Debugger an.

Wir interessieren uns hier für das H-Register, da es die Adresse des Anfangs des Disketten-Parameter-Blocks enthält. In diesem Beispiel ist das die Adresse D48A hex für das momentane Standardlaufwerk. Bevor wir uns diese Parameter ansehen, lassen Sie uns noch andere Diskettenformate betrachten. Man kann unterschiedliche Laufwerke gleichzeitig betreiben. Ein Laufwerk kann einseitig, das andere zweiseitig sein. Es gibt auch Laufwerke, die sowohl einfache als auch doppelte Schreibdichte verarbeiten können. Bei jedem dieser Fälle gibt es einen Parametersatz für ein bestimmtes Diskettenformat.

Angenommen, Laufwerk A liest nur doppelte, Laufwerk B jedoch einfache oder doppelte Schreibdichte. Legen Sie eine Diskette mit einfacher Dichte in das Laufwerk B. Schreiben Sie mit dem A-Kommando folgendes Programm ab 200 hex:

```
A200
0200 MVI E,1
0202 MVI C,E
0204 CALL 5
0207 RST 7
```

Mit diesem Programm wird mit Hilfe der BDOS-Funktion 14 (0E hex) das Standardlaufwerk geändert. Register E bezeichnet das neue Laufwerk. Der Wert 0 bezeichnet Laufwerk A, 1 Laufwerk B etc. Laden wir Register E mit 1, schalten wir auf Laufwerk B um. Register C wird mit 0E hex geladen. Der dritte Befehl ruft BDOS auf, und der letzte kehrt zum Debugger zurück.

Starten Sie das Programm mit dem Kommando G200. Der Kopf von Laufwerk B sollte geladen werden, und die Anzeigelampe sollte angehen. Starten Sie das erste Programm erneut mit G100, und zeigen Sie die Register mit dem X-Kommando an. Das sollte dann etwa so aussehen:

```
-Z-E- A=7B B=D43F D=003F H=D478 S=0100 P=0105 RST 07
```

Diesmal enthält Register H eine andere Adresse, weil ein anderer Parametersatz angesprochen wurde. Man sieht, daß die Adresse des ersten DPB 15 Bytes höher als die des zweiten ist. Man kann die DPBs im BIOS verteilen, aber normalerweise stehen sie zusammen. Da jeder DPB 15 Bytes lang ist, werden die Adressen von zwei aufeinanderfolgenden DPBs sich um 15 unterscheiden.

Studieren wir nun den DPB-Bereich. Wir haben DPBs bei den Adressen D478 hex und D48A hex gefunden. Es mag noch weitere geben für andere Diskettenformate. Man findet sie gewöhnlich auch in diesem Bereich. Daher beginnen wir die Ausgabe einige Zeilen vor dem gefundenen DPB. Geben Sie das Kommando

```
DD450
```

Die Ausgabe mag aussehen wie in Abb. 8.1. In der Abbildung sind fünf verschiedene DPBs zu sehen. Wir werden sie bald genauer studieren. Doch zunächst wollen wir einen einzelnen DPB betrachten.

```
D450: 00 00 00 00 00 00 DC 5D D4 08 DD F2 DC 28 00 04 .......Ü.....(..
D460: 0F 01 A9 00 3F 00 80 00 10 00 02 00 14 00 03 07 ....?...........
D470: 00 4F 00 3F 00 C0 00 10 00 03 00 28 00 03 07 00 .O.?.......(....
D480: A4 00 3F 00 C0 00 10 00 02 00 28 00 04 0F 01 51 ..?.......(....Q
D490: 00 3F 00 80 00 10 00 02 00 1A 00 03 07 00 F2 00 .?..............
D4A0: 3F 00 C0 00 10 00 02 00                         ?.......
```

Abb. 8.1: Fünf verschiedene Disketten-Parameter-Blöcke

DER DISKETTEN-PARAMETER-BLOCK

Neun Angaben im Parameter-Block beschreiben das Format einer Diskette. Jede dieser Angaben besteht aus einem oder zwei Bytes. Sie sind in Tabelle 8.1 zusammengefaßt. Der Wert in der ersten Spalte ist eine Relativadresse innerhalb eines DPBs. D. h. die Adresse einer Angabe setzt sich aus der von der BDOS-Funktion 31 erhaltenen Adresse im HL-Register und der Relativadresse zusammen.

Stelle	Symbol	Bytes	Erklärung
0	SPT	2	Logische Sektoren per Spur
2	BSH	1	Block-Shift
3	BLM	1	Block-Maske
4	EXM	1	Extent-Maske
5	DSM	2	Maximale Blocknummer
7	DRM	2	Maximale Zahl der Einträge im Inhaltsverzeichnis
9	AL0,1	2	Verzeichnisgröße
11	CKS	2	Anzahl der im Verzeichnis zu überprüfenden Einträge
13	OFF	2	Spur-Offset

Tabelle 8.1: Der Aufbau des Disketten-Parameter-Blocks

Lassen Sie uns nun die Diskettenparameter genauer betrachten. Der erste Eintrag, SPT, gibt die Anzahl der logischen 128-Byte-Sektoren einer Spur an. Es ist ein Zwei-Byte-Wert; das niederwertige Byte ist zuerst gespeichert. Viele Disketten-Steuerungen verarbeiten Sektoren mit mehr als 128 Bytes. Das North-Star-Format für doppelte Dichte benutzt 512-Byte-Sektoren; 10 Sektoren sind in einer Spur. Der SPT-Wert ist jedoch 40 und nicht 10, da die Anzahl der logischen, 128-Byte-langen Sektoren angegeben wird.

Der zweite und dritte Wert, BSH und BLM, hängen von der Blockgröße ab. BSH ist der Logarithmus zur Basis 2 der Anzahl von 128-Byte-Sektoren in einem Block. Bei der IBM-Standard-Diskette (8 Zoll) mit einfacher Schreibdichte ist daher BSH 3, denn sie hat acht Sektoren in einem Block (2 hoch 3 ist 8). BLM ist um eins kleiner als die Anzahl der Sektoren in einem Block. Mögliche Werte für BSH und BLM findet man in Tabelle 8.2.

Block Größe	Anzahl von Sectoren	BSH	BLM
1K	8	3	7
2K	16	4	15
4K	32	5	31
8K	64	6	63
16K	128	7	127

Tabelle 8.2: Erlaubte Werte für BSH und BLM

Wir haben gesehen, daß Dateien durch einen 32-Byte langen FCB beschrieben sind. Die ersten 16 Bytes enthalten den Namen und die Größe der Datei. Die übrigen 16 Bytes geben die Nummern der belegten Blöcke auf der Diskette an. Eine 8-Zoll-Diskette mit einfacher Dichte hat eine Blockgröße von 1K. Jeder FCB auf der Diskette kann daher maximal 16K Daten bezeichnen, da jede Nummer ein Byte groß ist.

Bei Disketten mit doppelter Schreibdichte ist es anders. Sie kann eine Blockgröße von 2K, 4K, 8K oder 16K haben. Nehmen wir einmal eine Diskette mit einer Blockgröße von 2K an. Die 16 Nummern umfassen nun 32K Bytes. Da CP/M 16K-Bereiche bearbeitet, wird jeder FCB in zwei 16K-Bereiche unterteilt. Bei einer Blockgröße von 4K haben wir vier 16K-Bereiche in einem FCB. Die Bezeichnung ist manchmal etwas verwirrend, wenn man einen FCB als Bereich bezeichnet. Heißt es z. B., daß ein physikalischer Bereich vier logische Bereiche hat, ist gemeint, daß mit einem FCB 64K Speicher bezeichnet sind.

Die Sache wird noch komplizierter, wenn eine Diskette oder Platte mehr als 255 Blöcke hat. Dann sind nämlich die Blocknummern zwei Bytes lang, und ein FCB enthält nur acht Nummern. Eine Diskette oder Platte mit 2K-Blöcken und zwei-Byte-langen Nummern kann dann nur 16K Bytes mit einem FCB beschreiben.

Die vielen verschiedenen Formate werden mit dem vierten Eintrag, EXM, im DPB beschrieben. Dieser Wert hängt von der Blockgröße und der Anzahl der Blöcke auf der Diskette ab. Er ist um eins kleiner als die Anzahl der Bereiche in einem FCB. Tabelle 8.3 zeigt diesen Zusammenhang. Kleine Disketten haben weniger als 255 Blöcke, größere Disketten oder Platten haben mehr.

Block Größe	Extent-Maske: Kleine Diskette	Extent-Maske: Große Platte
1K	0	--
2K	1	0
4K	3	1
8K	7	3
16K	15	7

Tabelle 8.3: Erlaubte Werte für EXM

Der fünfte Eintrag, DSM, gibt die größte Blocknummer auf der Diskette an. Von diesem Zwei-Byte-Wert ist wieder das niederwertige Byte zuerst

gespeichert. Da die Blocknummern bei null beginnen, ist die tatsächliche Anzahl der Blöcke um eins größer als der Wert DSM.

Der sechste Eintrag, DRM, gibt die um eins verminderte Anzahl der Einträge im Inhaltsverzeichnis an. Das niederwertige Byte dieses Zwei-Byte-Wertes ist zuerst gespeichert. Da ein Verzeichniseintrag 32 Bytes lang ist, passen vier Einträge in einen logischen 128-Byte-Sektor.

Das CP/M-Disketten-Inhaltsverzeichnis belegt die ersten Datenblöcke der Diskette. Daher müssen diese Blöcke stets als belegt gekennzeichnet werden, damit sie nicht überschrieben werden. Dies wird mit dem siebten Eintrag im DPB gemacht. Die beiden Bytes werden als ein 16-Bit-Vektor betrachtet. Beginnend auf der linken Seite belegt jedes gesetzte Bit einen Block für das Verzeichnis. Tabelle 8.4 zeigt die Belegung für das Verzeichnis.

Wenn eine Diskette aus dem Laufwerk genommen und durch eine andere ersetzt wird, muß man einen Warmstart durchführen, bevor man Daten auf die neue Diskette schreiben kann. Immer wenn eine Schreiboperation erforderlich ist, prüft CP/M das Verzeichnis, um herauszufinden, ob die Diskette gewechselt wurde. Der achte Eintrag im DPB, CKS, gibt die Anzahl der vor einer Schreiboperation zu überprüfenden Verzeichniseinträge an.

Anzahl von Verzeichnis Blöcken	Binär-Wert AL0	AL1	Hex Wert AL0	AL1
1	10000000	00000000	80	0
2	11000000	00000000	C0	0
3	11100000	00000000	E0	0
4	11110000	00000000	F0	0

Tabelle 8.4: 16-Bit-Tabelle für die Verzeichnis-Belegung

Bei Disketten oder anderen auswechselbaren Speichermedien ist CKS die Anzahl der Verzeichniseinträge DRM plus eins, geteilt durch vier (die Anzahl der Einträge in einem Sektor). Kann das Speichermedium nicht ausgewechselt werden, wie bei einer Festplatte, ist diese Prüfung nicht nötig. Dann wird dieser Wert auf Null gesetzt, wodurch der Warmstart beschleunigt wird.

Der neunte und letzte Eintrag im DPB ist der Track Offset OFF. Diese Zwei-Byte-Größe wird auf die vom BDOS verlangte Spurnummer (die

logische Spurnummer) addiert. Mit diesem Parameter kann man auch eine große Platte in mehrere logische Platten unterteilen. Jede logische Platte hat dann einen eigenen Track Offset. Angenommen, eine große Platte wird in die logischen Platten A, B und C unterteilt, dann könnte der Wert für OFF 0, 100 und 200 sein für die „Laufwerke" A, B und C.

Der DPB für eine Standard-8-Zoll-Diskette mit einfacher Schreibdichte ist in Tabelle 8.5 wiedergegeben.

Adresse	Symbol	Hex	Dezimal	Bedeutung
D499	SPT	1A	26	Logische Sektoren pro Spur
D49B	BSH	3	3	Block-Shift
D49C	BLM	7	7	Block-Maske
D49D	EXM	0	0	Extent-Maske
D49E	DSM	F2	242	Anzahl der Blöcke-1 (243 tatsächlich)
D4A0	DRM	3F	63	Verzeichniseinträge-1 (64 tatsächlich)
D4A2	AL0	C0		Verzeichnisbelegung (11000000)
D4A4	CKS	10	16	Verzeichnissektoren zu prüfen
D4A6	OFF	2	2	Spur-Offset

Tabelle 8.5: Der DPB für eine Standard-8-Zoll-Diskette

Sehen Sie sich den DPB bei Adresse D48A (Zeile 4 in Abb. 8.1) für Laufwerk A an. Der DPB beschreibt eine 5-Zoll-Diskette mit doppelter Schreibdichte und 40 Sektoren pro Spur (28 hex). Sie hat 82 Blöcke (51 hex + 1) zu 2K Bytes (BLM ist 15). Es gibt maximal 64 Verzeichniseinträge (3F hex + 1), und ein Block ist für das Verzeichnis belegt (AL1 ist 80 hex oder 10000000 bin). Der Track Offset ist 2.

UNTERSUCHUNG DER DISKETTEN-PARAMETER

Bevor wir die Diskettenparameter mit einem Programm anzeigen können, schreiben wir noch fünf Makros für die Ausgabe von Binärzahlen, für eine 16-Bit-Zahlen-Konvertierung, Multiplikation und Division.

Ein Makro zur binären Anzeige einer Binärzahl

Jede Information in einem Computer, ob Zeichen oder Zahl, wird als Folge von Bits gespeichert – binäre Null oder Eins. Wir haben bereits Routinen für die Konvertierung von binären Zahlen nach ASCII oder hexadezimal geschrieben. Manchmal wollen wir aber die Bits selbst

untersuchen. Um so ein Bitmuster eines Bytes darzustellen, muß man eine Folge von acht ASCII-Nullen und Einsen ausgeben. Diese Konvertierung nennen wir binär nach ASCII-binär.

In Kapitel 3 benutzten wir diese Routine, um herauszufinden, ob unser Drucker das DTR-Bit unterstützt. Wir werden jetzt diese Konvertierung in einem Makro verwirklichen. Tragen Sie das Makro BINBIN (Abb. 8.2) in Ihre Bibliothek ein. Beachten Sie das Flag BNFLAG.

```
BINBIN      MACRO
;;(Tagesdatum)
;;Inline-Makro zum Konvertieren einer Binärzahl in A
;;in eine Folge von ASCII-Binär-Ziffern.
;;
            LOCAL   BIT2,AROUND
            CALL    BINB2?
            IF      NOT BNFLAG
            JMP     AROUND
BINB2?:
            PUSH    B
            MOV     C,A
            MVI     B,8
BIT2:
            MOV     A,C
            ADD     A               ;;setze Carry
            MOV     C,A
            MVI     A,'0'/2
            ADC     A
            PCHAR
            DCR     B
            JNZ     BIT2
            POP     H
            RET
BNFLAG      SET     TRUE
            ENDIF
AROUND:
                                    ;;BINBIN
            ENDM
```

Abb. 8.2: Das Makro BINBIN für die Anzeige einer Binärzahl durch eine Folge von ASCII-Nullen und -Einsen

Ein Makro zur dezimalen Anzeige einer 16-Bit-Binärzahl

Manchmal will man das dezimale Äquivalent einer Binärzahl wissen Die größte 8-Bit-Zahl ist 255. Bei der Konvertierung einer 8-Bit-Zahl haben wir es also mit den Zehnerpotenzen 1, 10 und 100 zu tun. Konvertieren wir dagegen eine 16-Bit-Zahl nach dezimal, sind es die Zehnerpotenzen 1, 10, 100, 1000 und 10000, da die größte Zahl jetzt 65535 ist.

Bei dem hier benutzten Algorithmus subtrahieren wir Zehnerpotenzen von der Ausgangszahl, bis sie negativ wird. Die letzte Subtraktion machen wir dann wieder rückgängig. Die Anzahl der Subtraktionen entspricht einer Dezimalziffer. Das wird mit 1000, 100 und 10 wiederholt.

Wie wir wissen, kann eine 8080-CPU eine 16-Bit-Subtraktion nicht direkt ausführen. Dafür haben wir ein Makro geschrieben. Wir wissen auch, daß man statt einer Subtraktion auch das Komplement addieren kann. Das nutzen wir bei diesem Algorithmus aus. Wir addieren wiederholt −10000, das Komplement von 10000. Wenn das Ergebnis negativ wird, machen wir die letzte Subtraktion durch Subtraktion des Komplements rückgängig. Dafür benutzen wir dann das Makro SBC.

Das zweite Problem sind führende Nullen. Ist eine Zahl kleiner als 10000, ist die linke Ziffer null. Ist sie auch kleiner als 1000, ist die nächste Ziffer auch null. Gewöhnlich unterdrückt man jedoch bei Dezimalzahlen diese führenden Nullen, und daher werden wir das hier auch tun.

```
HLDEC       MACRO
;;(Tagesdatum)
;;Inline-Makro für die Ausgabe von HL als Dezimalzahl
;;benötigte Makros: SBC, PCHAR
;;
            LOCAL   AROUND,SUBTR,SUBT2,NZERO
            CALL    HLDC2?
            IF      NOT DEFLAG
            JMP     AROUND
HLDC2?:
            PUSH    H
            PUSH    D
            PUSH    B
            MVI     B,0             ;;führende Nullen-Flag
            LXI     D,-10000        ;Zweierkomplement
            CALL    SUBTR           ;Zehntausender
            LXI     D,-1000
```

Abb. 8.3: Das Makro HLDEC zur Ausgabe einer 16-Bit-Binärzahl in dezimaler Form

```
                CALL    SUBTR           ;Tausender
                LXI     D,-100
                CALL    SUBTR           ;;Hunderter
                LXI     D,-10
                CALL    SUBTR           ;;Zehner
                MOV     A,L
                ADI     '0'             ;;ASCII-Grundwert
                PCHAR
                POP     B
                POP     D
                POP     H
                RET
;;
;;subtrahiere Zehnerpotenz und zähle
;;
SUBTR:          MVI     C,'0'-1         ;;ASCII-Zähler
SUBT2:          INR     C
                DAD     D               ;;addiere negative Zahl
                JC      SUBT2           ;;weiter
;;einmal zu oft, eine Rückaddition
;;durch Subtraktion des Zweierkomplements
                SBC     HL,DE
                MOV     A,C             ;;Zähler
                CPI     '1'             ;;< 1?
                JNC     NZERO           ;;nein
                MOV     A,B             ;;teste Nullflag
                ORA     A               ;;gesetzt?
                MOV     A,C             ;;Zähler
                RZ                      ;;übergehe führende 0
                PCHAR
                RET
;;setze Flag für Zeichen ungleich 0
NZERO:
                MVI     B,0FFH
                PCHAR
                RET
DEFLAG          SET     TRUE
                ENDIF
AROUND:                                 ;;HLDEC
                ENDM
```

Abb. 8.3: Das Makro HLDEC zur Ausgabe einer 16-Bit-Binärzahl in dezimaler Form (Forts.)

Ein Makro zur hexadezimalen Anzeige einer 16-Bit-Binärzahl

In Kapitel 5 haben wir mit dem Makro OUTHEX einen 8-Bit-Wert mit zwei Hexadezimalziffern auf der Konsole angezeigt. Für das Programm in diesem Kapitel müssen wir eine 16-Bit-Zahl im HL-Register in eine Folge von vier Hexadezimalziffern umwandeln. Wenn der Wert im HL-Register größer als 255 ist, rufen wir das Makro OUTHEX zweimal auf. Dadurch erhalten wir vier ASCII-Zeichen. Wenn die Zahl in HL kleiner als 256 ist, ist das H-Register null. Dann rufen wir das Makro OUTHEX nur einmal auf. Tragen Sie das Makro OUTHL in Abb. 8.4 in Ihre Bibliothek ein.

```
OUTHL     MACRO
;;(Tagesdatum)
;;Inline-Makro für die Anzeige von HL in hex.
;;benötigtes Makro: OUTHEX
;;
          LOCAL     OVER
          MOV       A,H
          ORA       A
          JZ        OVER
          OUTHEX    H
OVER:
          OUTHEX    L
                                ;;OUTHL
          ENDM
```

Abb. 8.4: Das Makro OUTHL für die Anzeige einer 16-Bit-Binärzahl in hexadezimaler Form

Ein Makro zur Multiplikation einer 16-Bit-Zahl mit einer Zweierpotenz

Die CPUs 8080 und Z80 haben zwar Befehle für die Addition und Subtraktion, jedoch keine Befehle für die Multiplikation und Division. Wir wollen jetzt ein Makro für die Multiplikation einer 16-Bit-Zahl im HL-Register mit einer Zweierpotenz schreiben. Die Beschränkung des Multiplikators auf Zweierpotenzen vereinfacht unseren Algorithmus erheblich, ohne unsere Anwendung einzuschränken, da wir immer einen Multiplikator dieses Typs benötigen.

In unserem Makro betrachten wir zwei Sonderfälle am Beginn des Makros. Ist ein Faktor 0, wird das Ergebnis im HL-Register 0. Ist ein Fak-

tor 1, bleibt der Wert im HL-Register unverändert. Wir stellen den einen Faktor im B-Register, den anderen im HL-Register bereit und addieren den Wert im HL-Register zu sich selbst. Dadurch wird der Multiplikand verdoppelt. Dann rotieren wir den Multiplikator im B-Register und prüfen das Carry-Flag. Wenn das Flag gesetzt ist, war der Multiplikator 2, und die Operation ist beendet. Ist es nicht gesetzt, war der Multiplikator größer als 2, und die letzten Schritte werden wiederholt.

Tragen Sie das Makro MULT (Abb. 8.4) in die Bibliothek ein. Es hat einen optionalen Parameter – den Multiplikator. Fehlt er beim Aufruf, wird angenommen, daß der Multiplikator bereits im A-Register steht.

```
MULT        MACRO   TIMES
;;(Tagesdatum)
;;Inline-Makro für die Multiplikation des Wertes in HL mit TIMES.
;;Parameter sollte eine 2er-Potenz sein.
;;0 und 1 sind gültige Operanden.
;;Parameter fehlt, wenn A den Faktor enthält
;;
            LOCAL   LOOP, AROUND, NOTZ
            PUSH    B
            IF      NUL TIMES
            MOV     B,A
            ELSE
            MVI     B,TIMES
            ENDIF
            CALL    MULT2?
            POP     B
            IF      NOT MLFLAG
            JMP     AROUND
MULT2?:
            MOV     A,B
            ORA     A               ;null
            JNZ     NOTZ            ;nein
            MOV     L,A
            MOV     H,A             ;HL=0
            RET
NOTZ:
            RAR
            RC                      ;mal 1
```

Abb. 8.5: Das Makro MULT für die Multiplikation einer 16-Bit-Zahl in HL mit einer Zweierpotenz

```
        MOV     B,A
LOOP:
        DAD     H           ;mal 2
        MOV     A,B
        RAR
        MOV     B,A
        JNC     LOOP
        RET
MLFLAG  SET     TRUE        ;;nur eine Kopie
        ENDIF
AROUND:                     ;;MULT
        ENDM
```

Abb. 8.5: Das Makro MULT für die Multiplikation einer 16-Bit-Zahl in IIL mit einer Zweierpotenz (Forts.)

Ein Makro zur Division einer 16-Bit-Zahl durch eine Zweierpotenz

Das Gegenteil des vorigen Makros ist eine Routine für die Division einer 16-Bit-Zahl im HL-Register durch eine Zweierpotenz. Die Verdoppelung des Wertes im HL-Register ist gleichbedeutend mit einer Verschiebung des Doppelregisters nach links. Dividieren bedeutet eine Verschiebung nach rechts. Leider gibt es keinen 16-Bit-Verschiebe-Befehl; also müssen wir ersatzweise zwei 8-Bit-Verschiebungen durchführen.

Das Makro DIVIDE ist in Abb. 8.6 gezeigt; tragen Sie es in Ihre Bibliothek ein. Es hat einen optionalen Parameter, den Divisor. Fehlt er, wird 2 angenommen. Eine Division durch null ist undefiniert, das Makro läßt den Dividenden jedoch unverändert. Das Ergebnis bleibt auch unverändert, wenn der Divisor 1 ist.

```
DIVIDE      MACRO   DENOM
;;(Tagesdatum)
;;Inline-Makro für die Division des HL-Registers durch DENOM.
;;DENOM sollte eine Zweierpotenz sein (2, 4, 8, 16).
;;HL unverändert, falls DENOM = 0 oder 1.
;;
            LOCAL   AROUND,SHFTR?,DIV3?
```

Abb. 8.6: Das Makro DIVIDE für die Division einer 16-Bit-Zahl in HL durch eine Zweierpotenz

```
        PUSH    B
        IF      NUL DENOM
        MVI     B,2             ;Standard
        ELSE
        MVI     B,DENOM
        ENDIF
        CALL    DIV2?
        POP     B
        IF      NOT DVFLAG
        JMP     AROUND
DIV2?:
        MOV     A,B
        ORA     A               ;lösche Carry
        RZ                      ;div durch 0?
        RAR
        RC                      ;div durch 1?
        MOV     B,A
DIV3?:
        CALL    SHFTR?          ;schiebe HL nach rechts
        MOV     A,B             ;Divisor
        RAR
        MOV     B,A
        JNC     DIV3?
        RET
SHFTR?:                         ;16-Bit-Verschiebung
                                nach rechts
        XRA     A
        MOV     A,H
        RAR
        MOV     H,A
        MOV     A,L
        RAR
        MOV     L,A
        RET
DVFLAG  SET     TRUE            ;;nur eine Kopie
        ENDIF
AROUND:                         ;;DIVIDE
        ENDM
```

Abb. 8.6: Das Makro DIVIDE für die Division einer 16-Bit-Zahl in HL durch eine Zweierpotenz (Forts.)

Nachdem wir nun die nötigen Makros erstellt haben, können wir ein Programm für die Ausgabe der Diskettenparameter schreiben.

Ein Programm zur Ausgabe der Disketten-Parameter

Mit dem Programm in Abb. 8.7 kann man die Parameter einer beliebigen Diskette bestimmen. Die CP/M-Version muß bei diesem Programm 2.0 oder größer sein. Das Programm beginnt wie gewöhnlich mit den Makros ENTER und VERSN. Mit dem Makro CPMVER wird dann die CP/M-Version bestimmt. Ist sie kleiner als 2, wird das Programm mit einer Fehlermeldung beendet.

Im Speicher-FCB bei 5C hex wird geprüft, ob in der Kommandozeile ein Laufwerk angegeben wurde. War dies der Fall, wird die Subroutine SETDSK aufgerufen. In dieser Routine wird mit Hilfe der BDOS-Funktion 14 das gewünschte Laufwerk selektiert. War kein Laufwerk angegeben, wird das Standardlaufwerk benutzt. Die Subroutine SETDSK beendet sich etwas ungewöhnlich – mit einem Sprung zu BDOS statt dem normalen CALL. Die etwas durchsichtigere Folge

```
CALL    BDOS
RET
```

bewirkt das gleiche. Es wird jedoch mehr Code und mehr Stapel-Speicher benötigt als bei dem Befehl

```
JMP     BDOS
```

Im nächsten Schritt wird mit der BDOS-Funktion 31 die Adresse des Disketten-Parameter-Blocks bestimmt. Die Parameter werden dann an das Ende des Programms umgespeichert und dort geringfügig modifiziert. Die größte Blocknummer (DSM) wird z. B. um eins erhöht und ergibt dadurch die Anzahl der Blöcke. Die Anzahl der Verzeichniseinträge wird um eins erhöht und durch vier geteilt und ergibt dann die Anzahl der Sektoren für das Verzeichnis. Dieser Wert wird als DIRMAX gespeichert.

Die Beleg-Bytes (ALO und AL1) werden vertauscht, so daß das niederwertige Byte nach einem LHLD-Befehl im H-Register steht. Wiederholte DAD-Befehle verschieben dann das HL-Register nach links in das Carry-Flag. Das Flag wird dadurch immer gesetzt, wenn das entsprechende Bit in ALO oder AL1 gesetzt ist. Sie erinnern sich, daß ein gesetztes Bit die Belegung des entsprechenden Blocks anzeigt. Die Anzahl der für das Verzeichnis reservierten Blöcke wird dadurch ermittelt. Das Ergebnis wird von der Gesamtzahl der Blöcke subtrahiert, was die Zahl der Daten-

blöcke ergibt. Der Wert wird in NETBL gespeichert. Die Anzahl der Verzeichnisblöcke wird nach ASCII konvertiert und in ALLOCA für späteren Gebrauch gespeichert.

Geben Sie das in Abb. 8.7 gezeigte Programm ein, und assemblieren Sie es. Die bisher entwickelte Makrobibliothek wird hierfür benötigt. Testen Sie das Programm zunächst für das Standardlaufwerk mit dem Kommando

```
DIREC
```

Dann testen Sie es mit einem anderen Laufwerk mit dem Kommando

```
DIREC B:
```

Im zweiten Beispiel werden die Diskettenparameter des Laufwerks B angezeigt.

Das Ergebnis für ein 8-Zoll-Laufwerk mit einfacher Schreibdichte ist in Abb. 8.8 wiedergegeben. Die Ausgabe zeigt 26 Sektoren pro Spur und 8 Sektoren für einen 1K-Block. Es gibt 64 Verzeichniseinträge, gespeichert in zwei Blöcken. Die Diskette kann exklusive dem Verzeichnis maximal 241K Bytes an Daten aufnehmen. Der Track Offset ist 2; d. h. zwei Spuren sind reserviert.

Die Ausgabe für ein 5-Zoll-Laufwerk mit doppelter Schreibdichte, zweiseitig und hart-sektoriert, ist in Abb. 8.9 gezeigt. Hier sind 40 logische Sektoren pro Spur und 16 Sektoren pro 2K-Block ausgegeben. Die Diskette kann maximal 338K Bytes, exklusive einem Block für das Verzeichnis aufnehmen.

```
TITLE   'DIREC, Verzeichnis-Utility'
;
;(Tagesdatum)
;
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
BOOT    EQU     0
BDOS    EQU     5               ;BDOS-Eingang
TPA     EQU     100H
```

Abb. 8.7: Das Programm zum Anzeigen der Disk-Parameter

```
FCB     EQU     5CH             ;Dateisteuerblock
FCB1    EQU     5CH             ;1. Dateiname
FCB2    EQU     6CH             ;2. Dateiname
DBUFF   EQU     80H             ;Standardpuffer
ABUFF   EQU     DBUFF           ;Puffer
UNUSED  EQU     0E5H            ;unbenutzter Verz. Eintrag
LMAX    EQU     24              ;max Zeilen/Schirm
;
;Setze Flags im Hauptprogramm, damit von
;einigen Subroutines nur eine Kopie generiert wird.
;Die Flags müssen vor dem MACLIB-Call gesetzt werden.
;
BNFLAG  SET     FALSE           ;binär nach ASCII bin
COFLAG  SET     FALSE           ;Konsolausgabe
CRFLAG  SET     FALSE           ;Neue Zeile
CXFLAG  SET     FALSE           ;binär nach hex
DEFLAG  SET     FALSE           ;binär nach dezimal
DVFLAG  SET     FALSE           ;16-bit Division
MLFLAG  SET     FALSE           ;16-bit Multiplikation
MVFLAG  SET     FALSE           ;Blockverschiebung
PRFLAG  SET     FALSE           ;Konsolausgabe
;Ende der Flags
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum).DIREC.1'
        CPMVER                  ;teste Version
        CPI     20H
        JC      ERRVER          ;falsche Version
        LDA     FCB1
        ORA     A               ;Laufwerk angegeben?
        CNZ     SETDSK          ;ja
        CALL    GETDPH          ;Disk-Parameter
        CALL    XAMINE
        JMP     DONE
;
```

Abb. 8.7: Das Programm zum Anzeigen der Disk-Parameter (Forts.)

```
;Block-Move Disk-Parameter zum Ende des Programms
;
GETDPH:
        MVI     C,31            ;Disk-Param-Adresse
        CALL    BDOS
        MOVE    ,DPARM,15       ;kopiere ans Ende
        LHLD    BLKMAX          ;max. Zahl der Blöcke
        INX     H
        SHLD    BLKMAX          ;Beginn bei Null
        LHLD    DIRENT          ;Anzahl der Verzeichniseinträge
        INX     H               ;Beginn bei Null
        DIVIDE  4               ;konvertiere in # Sektoren
;Rette Anzahl der Verzeichnis-Sektoren als 16 bits
        SHLD    DIRMAX
        SHLD    DIRMX2
;
;Verzeichnis-Zuordnung wird gespeichert als
;1000 0000 für 1 Block, 1100 0000 für 2, etc.
;Das linke Byte soll in H stehen
;
        LHLD    ALLOC           ;vertausche Bytes
        MOV     A,L
        MOV     L,H
        MOV     H,A
        SHLD    ALLOC
;Anzahl der Verzeichnisblöcke in ASCII
XRA     A
XAM3:
        DAD     H               ;mal 2
        JNC     XAM4
        INR     A
        JMP     XAM3
XAM4:
        MOV     E,A             ;# Verz.Blöcke
        MVI     D,0
        LHLD    BLKMAX          ;Blöcke
        SBC     HL,DE           ;für Verzeichnis
        SHLD    NETBL           ;netto Datenblöcke
        MOV     A,E
        ORI     '0'
```

Abb. 8.7: Das Programm zum Anzeigen der Disk-Parameter (Forts.)

```
        STA     ALLOCA
        RET
;
;Anzeige der Disk-Parameter
;
XAMINE:
        PRINT   <CR,LF,'Sektoren/Spur: '>
        LHLD    NUMSEC
        HLDEC                   ;dezimal
        PCHAR   BLANK
        PCHAR   '('
        OUTHL
        PRINT   <' hex)',CR,LF,'Sektoren/Block: '>
        LDA     BLM
        INR     A
        MOV     L,A
        MVI     H,0
        HLDEC
        PCHAR   BLANK
        PCHAR   '('
        OUTHEX  L
        PRINT   ' hex)'
        PRINT   <CR,LF,'Blockgroesse: '>
        DIVIDE  8
        MOV     B,L             ;rette Blockgröße
        HLDEC
        PRINT   'K Bytes'
        LHLD    NETBL           ;# Daten-Blöcke
        MOV     A,B             ;Blockgröße
        MULT
        PRINT   <CR,LF,'Diskettengroesse: '>
        HLDEC
        PRINT   'K Bytes'
        PRINT   <CR,LF,'Extents/Eintrag: '>
        LDA     EMASK
        INR     A
        MOV     L,A
        MVI     H,0
        HLDEC
        PRINT   <CR,LF,'Anzahl der Bloecke: '>
```

Abb. 8.7: Das Programm zum Anzeigen der Disk-Parameter (Forts.)

```
        LHLD    BLKMAX
        HLDEC
        PCHAR   BLANK
        PCHAR   '('
        OUTHL
        PRINT   <'hex)',CR,LF,'Max. Dir. Eintraege: '>
        LHLD    DIRENT
        INX     H
        HLDEC
        PCHAR   BLANK
        PCHAR   '('
        OUTHL
        PRINT   <'hex)',CR,LF,'Dir. Bloecke: '>
        LDA     ALLOCA
        PCHAR
        PCHAR   BLANK
        PCHAR   '('
        LDA     ALLOC+1
        BINBIN                  ;ALLOC in binär
        LDA     ALLOC
        ORA     A
        JZ      XAM2
        BINBIN                  ;2. falls benötigt
XAM2:
        PRINT   <')',CR,LF,'Spur-Offset: '>
        LHLD    TRKOFF
        HLDEC
        MOV     A,H
        ORA     A
        JZ      XAM5            ;überspringe hex
        PCHAR   BLANK
        PCHAR   '('
        OUTHL
        PRINT   'hex)'
XAM5:
        CRLF
        RET
SETDSK:                         ;setze Laufwerk
        DCR     A               ;0=A, 1=B
        MOV     E,A
```

Abb. 8.7: Das Programm zum Anzeigen der Disk-Parameter (Forts.)

```
        MVI     C,14            ;selektiere neues Laufwerk
        JMP     BDOS
ERRVER:
        ERRORM  'CP/M-Version muss 2 oder groesser sein'
DONE:
        EXIT
DPARM:                          ;Kopie der Disk-Parameter
NUMSEC: DS      2               ;Sektoren per Spur
BSHIFT: DS      1               ;Block-Shift
BLM:    DS      1               ;Block-Maske
EMASK:  DS      1               ;Extent-Maske
BLKMAX: DS      2               ;max # Blöcke auf Diskette
DIRENT: DS      2               ;max # Verz.Einträge
ALLOC:  DS      2               ;AL1, AL0 vertauscht
CKS:    DS      2               ;Check Größe
TRKOFF: DS      2               ;Spur-Offset
;
DIRMAX: DS      2               ;max # Verzeichnis-Sektoren
NETBL:  DS      2               ;Anzahl von Daten-Blöcken
ALLOCA: DS      1               ;Verzeichnis-Blöcke (ASCII)
DIRMX2: DS      2               ;verbleibende Verz.Sektoren
;
        END     START
```

Abb. 8.7: Das Programm zum Anzeigen der Disk-Parameter (Forts.)

```
Sektoren/Spur: 26 (1A hex)
Sektoren/Block: 8 (8 hex)
Blockgroesse: 1K bytes
Diskettengroesse: 241K bytes
Extents/Eintrag: 1
Anzahl der Bloecke: 243 (F3 hex)
Max. Dir. Eintraege: 64 (40 hex)
Dir. Bloecke: 2 (11000000)
Spur-Offset: 2
```

Abb. 8.8: Die Disk-Parameter einer 8-Zoll-Diskette

```
Sektoren/Spur: 40 (28 hex)
Sektoren/Block: 16 (10 hex)
Blockgroesse: 2K bytes
Diskettengroesse: 338K bytes
Extents/Eintrag: 2
Anzahl der Bloecke: 170 (AA hex)
Max. Dir. Eintraege: 64 (40 hex)
Dir. Bloecke: 1 (10000000)
Spur-Offset: 2
```

Abb. 8.9: Die Disk-Parameter einer 5-Zoll-Diskette

DIE BLÖCKE FÜR DAS INHALTSVERZEICHNIS DER DISKETTE

Der erste oder die ersten Datenblöcke einer CP/M-Diskette enthalten ein Verzeichnis aller Dateien dieser Diskette. Wie wir in Kapitel 6 gesehen haben, ist jeder dieser Einträge 32 Bytes lang. Ein logischer Sektor kann also maximal vier Einträge enthalten.

Wie wir wissen, bezeichnet das erste Byte den Benutzer, der die Datei erstellt hat. Dies ist eine Binärzahl von 0–15. Der Wert E5 hex an dieser Stelle bedeutet, daß die Datei gelöscht wurde. Dateiname und Dateityp stehen als ASCII-Zeichen in den nächsten 11 Bytes. Dann folgen vier Bytes, die die Bereichsnummer und die Anzahl der Sätze enthalten.

Die tatsächliche Position der Datei auf der Diskette wird durch die folgenden 16 Bytes gekennzeichnet. Bei kleineren Disketten ist die Blocknummer ein Byte groß; bei größeren ist sie zwei Bytes lang, und das niederwertige Byte ist zuerst gespeichert.

Untersuchen wir nun, wie eine Gruppe von Dateien auf drei verschiedenen Diskettentypen gespeichert ist. Die drei Typen sind:

1K-Byte Blockgröße, 1-Byte-Blocknummer

2K-Byte Blockgröße, 1-Byte-Blocknummer

2K-Byte Blockgröße, 2-Byte-Blocknummer

Unser letztes Programm in diesem Buch dient der Untersuchung eines FCB. Bei einer Blockgröße von 1K sieht die Ausgabe etwa so aus:

```
00 CPMIO    ASM 00000055 02030405060708090A0B0C00
00 DUMP     COM 00000007 0D00
00 GO       COM 00000002 0E00
00 LOAD     COM 0000000E 0F1000
00 CPMIO    HEX 00000007 1100
00 WSOVLY1 OVR 00000080 12131415161718191A1B1C1D1E1F2021
00 WSOVLY1 OVR 01000080 22232425262728292A2B2C2D2E2F3031
00 WSOVLY1 OVR 0200000A 323300
01 TIME     COM 0000000A 343500
02 SORT     BAS 00000009 363700
02 SORT     BAK 0000000F 383900
E5SORTA     BAS 00000008 3A00
E5PRIN      STR 0000000C 3B3C00
```

Der erste FCB, CPMIO.ASM, gehört dem Benutzer 0. Er umfaßt 55 (85 dez) Sätze, die in den Blöcken 02–0C gespeichert sind. Die Blocknummer bezeichnet den eigentlichen Platz, an dem die Datei liegt. Die nächste Datei, DUMP.COM, umfaßt sieben Sätze; sie passen alle in den Block 0D. Die Zeilen 6, 7 und 8 des Verzeichnisses beziehen sich auf dieselbe Datei, WSOVLY1.OVR. Diese Datei ist so groß, daß sie drei FCBs erfordert. Der erste und der zweite FCB (numeriert mit 0 und 1) enthalten jeweils 80 hex Sätze. Der dritte FCB beschreibt den Bereich 2 und umfaßt 0A Sätze. Bei diesem Diskettenformat kan jeder FCB maximal einen Bereich von 80 hex Sätzen (16K Bytes) beschreiben.

Die Blockgröße in dem vorigen Beispiel betrug 1K Bytes. Bei CP/M sind jedoch auch Blockgrößen von 2K, 4K, 8K und 16K möglich. Das nächste Diskettenformat, das wir beschreiben, hat eine Blockgröße von 2K. Bei diesem Format kann ein FCB maximal zwei Bereiche von 16K umfassen. Sehen wir, wie die obigen Dateien dann gespeichert werden:

```
00 CPMIO    ASM 00000055 01020304050600
00 DUMP     COM 00000007 0700
00 GO       COM 00000002 0800
00 LOAD     COM 0000000E 0900
00 CPMIO    HEX 00000007 0A00
00 WSOVLY1 OVR 01000080 0B0C0D0E0F10111213141516171819 1A
00 WSOVLY1 OVR 0200000A 1B00
01 TIME     COM 0000000A 1C00
02 SORT     BAS 00000009 1D00
02 SORT     BAK 0000000F 1E00
```

Die Dateien haben in diesem Beispiel dieselben Größen wie in dem vorigen. Es werden jedoch weniger Blöcke benötigt, da die Blöcke doppelt so

groß sind. Auch benötigt WSOVLY1.OVR statt drei nur zwei FCBs. Die Bereichsnummer im ersten dieser beiden FCBs ist allerdings 1 und gibt damit an, daß die Bereiche 0 und 1 durch diesen FCB zusammengefaßt sind. Jeder dieser Bereiche hat 80 hex Sätze. Der dritte Bereich, Bereich 2, ist im zweiten FCB enthalten und hat 0A Sätze. Die Blocknummern laufen von 0B bis 1B.

Als drittes Beispiel betrachten wir eine Diskette (Platte) hoher Schreibdichte, die zwar 2K-Blöcke hat, aber 2-Byte-Blocknummern benutzt. Es werden wieder dieselben Dateien gezeigt. Die Blocknummern erscheinen jedoch als 0200, 0300, 0400 etc.

```
00 CPMIO    ASM 00000055 020003000400050006000700
00 DUMP     COM 00000007 0800
00 GO       COM 00000002 0900
00 LOAD     COM 0000000E 0A00
00 CPMIO    HEX 00000007 0B00
00 WSOVLY1 OVR 00000080 0C000D000E000F001000110012001300
00 WSOVLY1 OVR 01000080 14001500160017001800190001A001B00
00 WSOVLY1 OVR 0200000A 1C00
```

DIE BLOCK-ZUORDNUNGS-TABELLE

Wenn mit Control-C ein Warmstart ausgeführt wird, werden alle Laufwerke in den Grundzustand versetzt, und das Verzeicnis von Laufwerk A wird gelesen. Ist ein anderes Laufwerk als Laufwerk A das Standardlaufwerk, so wird auch das Verzeichnis dieses Laufwerks gelesen. Eine Block-Zuordnungs-Tabelle wird in diesem Schritt aufgebaut. In dieser Tabelle repräsentiert ein Bit einen Block. Bei benutzten Blöcken ist dies Bit 1, bei unbenutzten Blöcken 0. CP/M sucht in dieser Tabelle nach unbenutzten Blöcken, wenn eine neue Datei eingerichtet werden soll.

Der dritte Teil unseres letzten Programms erstellt eine Block-Zuordnungs-Tabelle nicht aus der CP/M-Version, sondern durch direkten Zugriff auf das Inhaltsverzeichnis der Diskette. In diesem Beispiel steht der erste Block, Block 0, in der oberen linken Ecke. Die Eins bedeutet, daß der Block belegt ist. Die übrigen Stellen sind Null und zeigen dadurch an, daß die Blöcke unbenutzt sind. Der erste Block dieser Diskette ist für das Verzeichnis reserviert. Das Verzeichnis belegt immer den oder die ersten Blöcke am Beginn des Datenbereichs. Daher gibt es immer Einsen am Anfang der Block-Zuordnungs-Tabelle. Die Anzahl der Blöcke in der Zusammenfassung unten in der Ausgabe schließt die für das Verzeichnis belegten Blöcke nicht mit ein.

Immer wenn eine Datei erstellt wird, werden die den Blöcken entspre-

```
     01234567  89ABCDEF 01234567  89ABCDEF
00:  10000000  00000000 00000000  00000000
20:  00000000  00000000 00000000  00000000
40:  00000000  00000000 00000000  00000000
60:  00000000  00000000 00000000  00000000
80:  00000000  00000000 00000000  00000000
A0:  00000000  00
169 Bloecke total, 0 benutzt, 169 uebrig
```

Abb. 8.10: Block-Zuordnungs-Tabelle einer frisch formatierten Diskette

chenden Bits auf eins gesetzt. Beim Speichern weiterer Dateien wird die Block-Zuordnungs-Tabelle weiter gefüllt. Beim Löschen von Dateien entstehen Lücken in der Tabelle. Nach einer Weile könnte sie dann so aussehen wie in Abb. 8.11.

Die Block-Zuordnungs-Tabelle zeigt auch an, ob es mehrere Verweise auf eine Datei gibt. Betrachten Sie die Block-Zuordnungs-Tabelle in Abb. 8.12. Hier sind mehrere Blöcke (41, 42, 47 und andere) mit dem Wert 2 gekennzeichnet. Das bedeutet, daß zwei verschiedene Dateien diese Blöcke verwenden. Das kann mit einem Disketten-Dienstprogramm wie BADLIM oder RECLAIM auftreten. Diese Programme lesen die gesamte Diskette und suchen dabei nach defekten Sektoren. Wenn solche Sektoren auftreten, werden sie in einer speziellen Datei gesammelt, damit sie nicht mehr benutzt werden können. Wenn das Original-Programm allerdings noch existiert, bezieht es sich auch auf diese Bereiche.

```
     01234567  89ABCDEF 01234567  89ABCDEF
00:  11111111  11111111 11111111  11111111
20:  11111111  11111000 01111111  11111111
40:  11111111  11111111 10000001  11111111
60:  11111111  00000000 00000000  00000000
80:  00000000  00000000 00000000  11111111
A0:  11111000  00
169 Bloecke total, 63 benutzt, 106 uebrig
```

Abb. 8.11: Block-Zuordnungs-Tabelle einer Diskette mit Dateien

```
     01234567  89ABCDEF 01234567  89ABCDEF
00:  11111111  11111111 11111111  11111111
20:  11111111  11111111 11111111  11111111
40:  12211112  21111111 11111111  11111111
60:  22111111  11111110 11111111  11111121
80:  11101222  21111111 11111111  11111111
A0:  11111101  11
169 Bloecke total, 166 benutzt, 3 uebrig
```

Abb. 8.12: Block-Zuordnungs-Tabelle mit Mehrfach-Bezügen auf eine Datei

UNTERSUCHUNG DER BLÖCKE FÜR DAS INHALTSVERZEICHNIS UND DER BLOCK-ZUORDNUNGS-TABELLE

In diesem Abschnitt wollen wir unser Verzeichnis-Programm erweitern, so daß es die tatsächlichen Verzeichniseinträge ausgibt. Die Benutzernummer und die Blocknummern werden angezeigt. Außerdem wird für die Diskette eine Block-Zuordnungs-Tabelle aufgebaut. Doch bevor wir dieses Programm entwickeln, müssen wir noch ein weiteres Makro in unsere Bibliothek eintragen.

Ein Makro zum Füllen großer Bereiche

In Kapitel 4 haben wir ein Makro geschrieben, mit dem man einen Speicherbereich mit einer Konstanten füllen konnte. Die Länge dieses Bereichs war auf 256 Bytes beschränkt, da zum Zählen ein einfaches Register verwendet wurde. Im nächsten Programm müssen wir einen Bereich füllen, der länger als 256 Bytes ist. Wir ändern daher unser Makro FILL, so daß ein Doppelregister zum Zählen benutzt wird.

Kopieren Sie das Makro FILL, und nennen Sie die Kopie FILLD (für doppelt). Ändern Sie das neue Makro nach Abb. 8.13 ab. Beachten Sie, daß in dieser Version dasselbe Flag, FLFLAG, verwendet wird. Das bedeutet, daß Sie in diesem Programm nur eines der beiden Makros benutzen dürfen.

Nun können wir zwei weitere Funktionen in unser Programm einbauen.

Das Programm DIREC, Version 2

Machen Sie eine Kopie der ersten Version des Verzeichnis-Programms in Abb. 8.7, und ändern Sie es nach Abb. 8.14 ab. Das Makro FILLD wird

```
FILLD       MACRO   ADDR, BYTES, CHAR
;;(Tagesdatum)
;;(für große Bereiche)
;;Inline-Makro zum Füllen von BYTES Bytes
;;mit dem Zeichen CHAR beginnend bei ADDR.
;;Anwendung: FILL     FCB+1, 8, blank
;;          FILL     FCB+9, 3, '?'
;;
            LOCAL    AROUND,FILL3?
            PUSH     H
            PUSH     B
            IF       NOT NUL ADDR
            LXI      H,ADDR
            ENDIF
            IF       NOT NUL BYTES
            LXI      B,BYTES
            ENDIF
            MVI      A,CHAR
            CALL     FILL2?
            POP      B
            POP      H
            IF       NOT FLFLAG
            JMP      AROUND
FILL2?:
            PUSH     D
            MOV      D,A
FILL3?:
            MOV      M,D
            INX      H
            DCX      B
            MOV      A,C
            ORA      B
            JNZ      FILL3?
            POP      D
            RET
FLFLAG      SET      TRUE
            ENDIF
AROUND:               ;;FILLD
            ENDM
```

Abb. 8.13: Das Makro FILLD zum Füllen eines großen Speicherbereichs mit einer Konstanten

benutzt. Assemblieren und testen Sie es. Wenn Sie es ohne Parameter starten, wird, wie bei der vorigen Version, das Standardlaufwerk benutzt. Wollen Sie jedoch ein anderes Laufwerk ansprechen, geben Sie seinen Namen und einen Doppelpunkt an.

Die neue Version gibt, wie die vorige, als erstes die Diskettenparameter aus. Nach dem Drücken einer Taste wird dieses Programm jedoch fortgesetzt. Jeder Verzeichniseintrag wird in einer separaten Zeile ausgegeben. Die Benutzernummer, der Bereich, die Anzahl der Sätze und die Blocknummern werden angezeigt. Nach nochmaligem Drücken einer Taste wird im dritten Teil des Programms eine Block-Zuordnungs-Tabelle für diese Diskette ausgegeben. Belegte Blöcke sind mit einer Eins, unbelegte Blöcke mit einer Null bezeichnet.

Das Programm DIREC beginnt wie das vorige mit den Makros ENTER, VERSN und CPMVER. In der Subroutine CDISK wird mit Hilfe der BDOS-Funktion 25 das Standardlaufwerk bestimmt. Es wird geprüft, ob in der Kommandozeile ein Laufwerk angegeben wurde. War das der Fall, wird auch hier die Subroutine SETDSK aufgerufen. War kein Laufwerk angegeben, steht das Standardlaufwerk im FCB1. Der Laufwerksname wird diesmal auch ausgegeben.

Wie in der vorigen Version werden die Diskettenparameter ausgegeben. Dann wird auf das Drücken einer beliebigen Taste gewartet. Dadurch wird das vollständige Disketten-Inhaltsverzeichnis ausgegeben. Nach dem Drücken einer weiteren Taste wird die Block-Zuordnungs-Tabelle angezeigt.

In diesem Programm werden zahlreiche Diskettenoperationen durch Aufruf des BIOS statt des BDOS durchgeführt. Sie erinnern sich, daß das BIOS mit einer Sequenz von Sprungvektoren beginnt. Hier interessieren wir uns für die Vektoren in den Positionen 8, 9, 10, 12, 13 und 15. Es sind dies die Vektoren:

```
JMP   SELDSK    Selektieren des Laufwerks in C
JMP   SETTRK    Setzen Spurnummer auf den Wert in BC
JMP   SETSEC    Setzen Sektornummer auf den Wert in BC
JMP   READ      Lesen eines Sektors
JMP   WRITE     Schreiben eines Sektors
JMP   SECTRN    Übersetzung logischen in physikalischen Sektor
```

Die Lage des BIOS ändert sich mit den verschiedenen Größen von CP/M. Daher kann man seine exakte Adresse im Programm noch nicht angeben.

Die Adresse des Warmstartvektors steht ab Adresse 1. Wir können sie

auslesen, die Differenz zum gewünschten Vektor addieren und dahin springen. Z. B. ist der Vektor SELDSK der achte hinter dem Warmstartvektor, und jeder Vektor ist 3 Bytes lang. Man muß also 8 mal 3 zur Adresse des Warmstartvektors addieren. Die Befehle dafür sind:

```
SELDSK:                 ;selektiere Laufwerk in C
        LHLD  BOOT+1    ;Warmstartvektoradresse nach HL
        PUSH  D         ;retten DE
        LXI   D,3*8     ;Differenz nach DE
        DAD   D         ;addiere zu HL
        POP   D         ;wiederladen DE
        PCHL            ;Sprung zur Adresse in HL
```

Bei den anderen fünf Routinen ist es analog.

Immer wenn ein Laufwerk selektiert wird, baut CP/M eine Tabelle der benutzten Sektoren auf. Wir werden diese aber in diesem Programm nicht verwenden, sondern eine separate Tabelle aufbauen. Dafür haben wir einen Bereich ab BMAP am Ende des Programms reserviert. CP/M benutzt zwar ein Bit für jeden Block, wir werden hier jedoch ein Byte verwenden. Der Bereich wird mit Nullen initialisiert. Jede gefundene Blocknummer bewirkt eine Erhöhung um eins an der entsprechenden Stelle der Tabelle.

```
TITLE   'DIREC, Verzeichnis-Utility'
;
;(Tagesdatum)
;
;
FALSE   EQU     0
TRUE    EQU     NOT FALSE
;
BOOT    EQU     0
BDOS    EQU     5               ;BDOS-Eingang
TPA     EQU     100H
FCB     EQU     5CH             ;Dateisteuerblock
FCB1    EQU     5CH             ;1. Dateiname
FCB2    EQU     6CH             ;2. Dateiname
DBUFF   EQU     80H             ;Standardpuffer
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle

```
ABUFF   EQU     DBUFF           ;Puffer
UNUSED  EQU     0E5H            ;unbenutzter Verz.Eintrag
;
;Setze Flags im Hauptprogramm, damit von
;einigen Subroutines nur eine Kopie generiert wird.
;Die Flags müssen vor dem MACLIB-Call gesetzt werden.
;
BNFLAG  SET     FALSE           ;binär nach ASCII bin
CIFLAG  SET     FALSE           ;Konsoleingabe
COFLAG  SET     FALSE           ;Konsolausgabe
CRFLAG  SET     FALSE           ;Neue Zeile
CXFLAG  SET     FALSE           ;binär nach hex
DEFLAG  SET     FALSE           ;binär nach dezimal
DVFLAG  SET     FALSE           ;16-bit Division
MLFLAG  SET     FALSE           ;16-bit Multiplikation
MVFLAG  SET     FALSE           ;Blockverschiebung
PRFLAG  SET     FALSE           ;Konsolausgabe
FLFLAG  SET     FALSE           ;Löschen
;
;Ende der Flags
;
        MACLIB  CPMMAC
;
ORG     TPA
;
START:
        ENTER
        VERSN   '(Tagesdatum).DIREC.2'
        CPMVER                  ;teste Version
        CPI     20H
        JC      ERRVER          ;falsche Version
        PRINT   'Fuer Laufwerk'
        CALL    CDISK           ;momentanes Laufwerk
        LDA     FCB1
        ORA     A               ;Laufwerk angegeben?
        CNZ     SETDSK          ;ja
        LDA     CURD2           ;angegebenes Laufwerk
        STA     FCB1            ;binär
        ADI     'A'             ;konvertiere nach ASCII
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
            PCHAR
            CALL    GETDPH          ;Disk-Parameter
            CALL    XAMINE          ;anzeigen
            PRINT   'Bitte eine Taste druecken:'
            READCH                  ;warte auf Eingabe
            CALL    REPEAT          ;init. Parameter
            CALL    BLOCK           ;Zuordnungs-Tabelle
            JMP     DONE
;
;Blockverschiebung der Disk-Parameter zum Ende des Programms
;
GETDPH:
            MVI     C,31            ;Disk-Param.-Adresse
            CALL    BDOS
            MOVE    ,DPARM,15       ;Kopie ans Ende
            LHLD    BLKMAX          ;max # Blöcke
            INX     H
            SHLD    BLKMAX          ;Beginn bei Null
            LHLD    DIRENT          ;# Verzeichnis-Einträge
            INX     H               ;Beginn bei Null
            DIVIDE  4               ;konvertiere # Sektoren
;Rette Anzahl Verzeichnis-Sektoren als 16 Bits
            SHLD    DIRMAX
            SHLD    DIRMX2
;
;Verzeichnis-Zuordnung wird gespeichert als
;1000 0000 für 1 Block, 1100 0000 für 2, etc.
;Das linke Byte soll aber in H stehen.
;
            LHLD    ALLOC           ;vertausche Bytes
            MOV     A,L
            MOV     L,H
            MOV     H,A
            SHLD    ALLOC
;Anzahl Verzeichnis-Blöcke in ASCII
            XRA     A
XAM3:
            DAD     H               ;mal 2
            JNC     XAM4
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
            INR     A
            JMP     XAM3
XAM4:
            MOV     E,A             ;# Verz.Blöcke
            MVI     D,0
            LHLD    BLKMAX          ;Blöcke
            SBC     HL,DE           ;für Verzeichnis
            SHLD    NETBL           ;netto Daten-Blöcke
            MOV     A,E
            ORI     '0'
            STA     ALLOCA          ;save
;
;selektiere Laufwerk und init. Disk-Parameter-Kopf
;
            LDA     FCB
            MOV     C,A
            CALL    SELDSK          ;selektiere Laufwerk
            MOV     A,H             ;HL enthält DPH
            ORA     L
            JZ      ILDISK          ;Fehler, kein Laufwerk
            MOV     E,M             ;Übersetzungstabelle
            INX     H               ;Adresse
            MOV     D,M
            XCHG
            SHLD    DPH
            RET
;
;Anzeige der Disk-Parameter
;
XAMINE:
            PRINT   <CR,LF,'Sektoren/Spur: '>
            LHLD    NUMSEC
            HLDEC                   ;dezimal
            PCHAR   BLANK
            PCHAR   '('
            OUTHL
            PRINT   <' hex)',CR,LF,'Sektoren/Block: '>
            LDA     BLM
            INR     A
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
        MOV     L,A
        MVI     H,0
        HLDEC
        PCHAR   BLANK
        PCHAR   '('
        OUTHEX  L
        PRINT   ' hex)'
        PRINT   <CR,LF,'Blockgroesse: '>
        DIVIDE  8
        MOV     B,L             ;Blockgröße
        HLDEC
        PRINT   'K Bytes'
        LHLD    NETBL           ;# Daten-Blöcke
        MOV     A,B             ;Blockgröße
        MULT
        PRINT   <CR,LF,'Diskettengroesse: '>
        HLDEC
        PRINT   'K Bytes'
        PRINT   <CR,LF,'Extents/Eintrag: '>
        LDA     EMASK
        INR     A
        MOV     L,A
        MVI     H,0
        HLDEC
        PRINT   <CR,LF,'Anzahl der Bloecke: '>
        LHLD    BLKMAX
        HLDEC
        PCHAR   BLANK
        PCHAR   '('
        OUTHL
        PRINT   <' hex)',CR,LF,'Max. Dir. Eintraege: '>
        LHLD    DIRENT
        INX     H
        HLDEC
        PCHAR   BLANK
        PCHAR   '('
        OUTHL
        PRINT   <' hex)',CR,LF,'Dir. Bloecke: '>
        LDA     ALLOCA
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
        PCHAR
        PCHAR   BLANK
        PCHAR   '('
        LDA     ALLOC+1
        BINBIN                  ;ALLOC in binär
        LDA     ALLOC
        ORA     A
        JZ      XAM2
        BINBIN                  ;2. falls benötigt
XAM2:
        PRINT   <')',CR,LF,'Spur-Offset:'>
        LHLD    TRKOFF
        HLDEC
        MOV     A,H
        ORA     A
        JZ      XAM5            ;überspringe hex
        PCHAR   BLANK
        PCHAR   '('
        OUTHL
        PRINT   'hex)'
XAM5:
        CRLF
        RET
SETDSK:                         ;selektiere Laufwerk
        DCR     A               ;0=A, 1=B
        STA     CURD2
        MOV     E,A
        MVI     C,14            ;selektiere neues Laufwerk
        JMP     BDOS
;
REPEAT:                         ;zurücksetzen Parameter
        FILLD   SECTOR, HERE-SECTOR,0
        LHLD    TRKOFF
        SHLD    TRACK           ;Spur-Offset zurücksetzen
        LHLD    DIRMAX          ;# Verzeichnis-Sektoren
        SHLD    DIRMX2
        RET
;
;Anzeige der Block-Zuordnungs-Tabelle
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
;
BLOCK:
;
;Setze reservierte Verzeichnis-Blöcke in Tabelle
;durch Shift von ALLOC nach links
;
            PRINT   <CR,LF,LF,' Dateizuordnungstabelle',CR,LF>
            LHLD    BLKMAX          ;Anzahl Blöcke
            MOV     B,H
            MOV     C,L
            FILLD   BMAP,,0         ;löschen des Tabellenbereichs
            LHLD    ALLOC
            LXI     D,BMAP
C14A:
            XCHG
            INR     M               ;setze Bit
            INX     H
            XCHG
            DAD     H               ;mal 2
            MOV     A,L
            ORA     H               ;null?
            JNZ     C14A            ;nein
BLOCK3:
            CALL    NXTSEC
            JZ      BLOCK4          ;fertig
            CALL    BPROG
            JMP     BLOCK3
;
;Anzeige der Zuordnungstabelle
;
BLOCK4:
            PRINT   'Bitte eine Taste druecken: '
            READCH                  ;warte auf Eingabe
            PRINT   <CR,LF,LF,' 01234567 89ABCDEF'>
            PRINT   ' 01234567 89ABCDEF'
            LHLD    BLKMAX
            MOV     B,H
            MOV     C,L
            LXI     H,BMAP          ;Beginn der Tabelle
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
BMAP2:
        MOV     A,L
        ANI     0FH             ;lösche obere 4 Bits
        JNZ     BMAP6
        MOV     A,L
        ANI     1FH
        JZ      BMAP7           ;lösche obere 3 Bits
        PCHAR   BLANK           ;gerade
        JMP     BMAP5
BMAP7:
        ABORT   ESC
        CRLF                    ;neue Zeile
        OUTHEX  L               ;Adresse
        PCHAR   ':'
        PCHAR   BLANK
        JMP     BMAP5
BMAP6:
        CPI     8
        JNZ     BMAP5
        PCHAR   BLANK
BMAP5:
        MOV     A,M             ;Eintrag
        ORA     A               ;null?
        JZ      BMAP8           ;ja
        XCHG                    ;rette HL in DE
        LHLD    BLKCNT
        INX     H               ;Zähler
        SHLD    BLKCNT
        XCHG
BMAP8:
        CPI     10
        JNC     BMAP3           ;<9
        ORI     '0'             ;mache ASCII
        JMP     BMAP4
BMAP3:
        ADI     'A'-10          ;mache hex
BMAP4:
        PCHAR                   ;ausgeben
        INX     H
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
            DCX     B               ;zählen
            MOV     A,C             ;fertig?
            ORA     B
            JNZ     BMAP2           ;nein
;
;Anzeige der Gesamtblockzahl und Zahl der benutzten Blöcke
;
            CRLF
            LHLD    NETBL           ;netto # Bloecke
            HLDEC
            PUSH    H
            PRINT   ' Bloecke total, '
            LHLD    BLKCNT
            LDA     ALLOCA          ;Verz.Blöcke
            SUI     '0'             ;mache binär
            MOV     E,A
            MVI     D,0
            SBC     HL,DE
            HLDEC
            PRINT   ' benutzt, '
            XCHG
            POP     H
            SBC     HL,DE           ;Differenz
            HLDEC
            PRINT   <' uebrig',CR,LF>
            RET
;
;Aufbau der Block-Zuordnungs-Tabelle
;
BPROG:
            CALL    E5AREA          ;suche E5
            MOV     A,M             ;1. Byte
            CPI     17              ;Benutzer > 16?
            JNC     BKINCD          ;ja
            PUSH    H
;
            OUTHEX                  ;Benutzernummer
            PCHAR   BLANK
            INX     H               ;Dateiname
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
            PRINT   ,11             ;Anzeige Dateiname
            PCHAR   BLANK
            LXI     D,11            ;hinter Dateiname
            DAD     D               ;1. Eintrag
            MVI     C,4             ;nächste 4 Bytes
;
;Die nächsten 4 Bytes enthalten Bereich und Anzahl der Sektoren
;
LOOP2:
            OUTHEX  M
            INX     H
            DCR     C
            JNZ     LOOP2
            PCHAR   BLANK
            MVI     C,16            ;16 Blöcke/Bereich
;
;Test auf mehr als 255 Blöcke.
;Wenn ja, wird eine 16-Bit-Blockadresse benutzt
;
            LDA     BLKMAX+1        ;obere Hälfte
            ORA     A               ;null?
            JNZ     BNEXT6          ;nein, 16 Bits
;
;8-bit Blockadressen
BNEXT8:
            MOV     A,M
            OUTHEX                  ;Anzeige Blocknummer
            ORA     A               ;null?
            JZ      BPRT2           ;letzter Block
            PUSH    H
            LXI     H,BMAP          ;Start
            MOV     E,A
            MVI     D,0
            DAD     D               ;Offset
            INR     M               ;kennzeichne als benutzt
            POP     H
            INX     H
            MOV     A,L
            ANI     0FH             ;Ende der Zeile?
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
            JNZ     BNEXT8      ;nein
            JMP     BPRT2
;
;16-bit Blockadressen
;
BNEXT6:
            MOV     E,M         ;niederes Byte
            OUTHEX  E           ;Blocknummer
            INX     H
            MOV     A,M         ;oberes Byte
            OUTHEX              ;Blocknummer
            ORA     E           ;null?
            JZ      BPRT2       ;ja, fertig
            MOV     D,M
            PUSH    H
            LXI     H,BMAP      ;Tabellenbeginn
            DAD     D           ;addiere Adresse
            INR     M           ;kennzeichne als benutzt
            POP     H
            INX     H           ;nächste Adresse
            MOV     A,L
            ANI     0FH         ;Zeilenende?
            JNZ     BNEXT6      ;nein
BPRT2:
            POP     H           ;Beginn des FCB
            CRLF                ;neue Zeile
BKINCD:
            CALL    DECCNT
            JZ      CKDONE
            LXI     D,32        ;FCB-Länge
            DAD     D           ;nächster Eintrag
            JMP     BPROG
;
;erhöhe Zähler, vermindere Sektornummer
;
CKDONE:
            LHLD    DIRMX2
            DCX     H           ;Sektorzähler
            SHLD    DIRMX2
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
            LHLD    SECTOR      ;16 Bits
            INX     H
            SHLD    SECTOR
            XCHG
            LHLD    NUMSEC      ;Sektoren/Spur
;teste, ob eine neue Spur gelesen werden muß
;
            SBC     HL,DE       ;Differenz
            MOV     A,L
            ORA     H           ;null?
            RNZ
            SHLD    SECTOR      ;null setzen
            LHLD    TRACK
            INX     H           ;erhöhe Spur
            SHLD    TRACK
            RET
;
;Lies nächsten Sektor (4 Verzeichnis-Einträge).
;Setze Zero-Flag, falls letzter Sektor
;
NXTSEC:
            LDA     E5FLAG      ;uninitialisierter gefunden?
            CPI     1
            RZ                  ;ja
NXTSF:
            LHLD    DIRMX2      ;mehr Sektoren?
            MOV     A,L
            ORA     H           ;setze Flags
            RZ                  ;nein
            CALL    SETTRK      ;setze Spur
            LHLD    SECTOR      ;16 Bits
            MOV     B,H
            MOV     C,L
            CALL    TRANSL
            CALL    SETSEC      ;setze Sektor
            CALL    READ
            MVI     A,4         ;Einträge/Sektor
            STA     ECOUNT
            LXI     H,ABUFF     ;DMA-Adresse
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
            ANI     1
            XRI     1           ;invertiere Fehler-Flag
            RET                 ;Null, wenn Fehler
;
;Vermindere Anzahl der verbleibenden Einträge im Sektor
;(4 maximal). Zero-Flag gesetzt, falls letzter
DECCNT:
            LDA     ECOUNT      ;Einträge/Sektor
            DCR     A
            STA     ECOUNT
            RET
;
;Test auf E5 uninitialisierter Bereich, setze E5FLAG = 1, wenn ja
;
E5AREA:
            INX     H           ;1. Zeichen
            INX     H           ;2. Zeichen
            MOV     A,M
            CPI     UNUSED
            DCX     H
            DCX     H
            RNZ                 ;nicht gefunden
            MVI     A,1
            STA     E5FLAG      ;setze Flag
            RET
;
;suche momentanes Standardlaufwerk
;
CDISK:
            MVI     C,25
            CALL    BDOS
            STA     CURD2       ;A=0, B=1
            ADI     'A'         ;konvertiere nach ASCII
            STA     CURDSK
            RET
;
;übersetze BC von logischer in physikalische
;Sektornummer BC => HL => BC
;
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
TRANSL:
        LHLD    DPH             ;Translate-Tabelle
        XCHG
        CALL    SECTRN
        MOV     B,H
        MOV     C,L
        RET
;
;setze Spur auf 16-bit Wert in BC
;
SETTRK:
        LHLD    TRACK           ;16 Bits
        MOV     B,H             ;kann null sein
        MOV     C,L
        LHLD    BOOT+1          ;Warmstart
        PUSH    D
        LXI     D,3*9           ;Offset
        DAD     D
        POP     D
        PCHL
SETSEC:                         ;selektiere Sektor in BC
        LHLD    BOOT+1          ;Warmstart
        PUSH    D
        LXI     D,3*10          ;Offset
        DAD     D
        POP     D
        PCHL
SELDSK:                         ;selektiere Disk in C
        LHLD    BOOT+1          ;Warmstart
        PUSH    D
        LXI     D,3*8           ;Offset
        DAD     D
        POP     D
        PCHL
;Lies Sektor, A=0, wenn erfolgreich
READ:
        LHLD    BOOT+1          ;Warmstart
        PUSH    D
        LXI     D,3*12          ;Offset
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
            DAD     D
            POP     D
            PCHL
;Schreibe Sektor, A=0, wenn erfolgreich
WRITE:
            LHLD    BOOT+1          ;Warmstart
            PUSH    D
            LXI     D,3*13          ;Offset
            DAD     D
            POP     D
            PCHL
;
;Sektor-Übersetzung von logischem Sektor in BC
;in physikalischen Sektor in HL, DE enthält Translate-Tabelle
SECTRN:
            LHLD    BOOT+1          ;Warmstart
            PUSH    D
            LXI     D,3*15          ;Offset
            DAD     D
            POP     D
            PCHL
;
ERRVER:
            ERRORM  '?CP/M-Version muss 2 oder groesser sein'
ILDISK:
            ERRORM  '?Illegales Laufwerk'
DONE:
            EXIT
;
DPARM:                              ;Kopie der Disk-Parameter
NUMSEC:     DS      2               ;Sektoren per Spur
BSHIFT:     DS      1               ;Block-Shift
BLM:        DS      1               ;Block-Maske
EMASK:      DS      1               ;Extent-Maske
BLKMAX:     DS      2               ;max # Blöcke auf Diskette
DIRENT:     DS      2               ;max # Verz.Einträge
ALLOC:      DS      2               ;A11,A10 vertauscht
CKS:        DS      2               ;Check Size
TRKOFF:     DS      2               ;Spur-Offset
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

```
DPH:      DS      2               ;Disk-Parameter-Kopf
;
DIRMAX:   DS      2               ;max # Verzeichnis-Sektoren
NETBL:    DS      2               ;Anzahl der Daten-Blöcke
ALLOCA:   DS      1               ;Verzeichnis-Blöcke (ASCII)
DIRMX2:   DS      2               ;verbleibende Verz.Sektoren
CURDSK:   DS      1               ;Laufwerk (ASCII)
CURD2:    DS      1               ;Laufwerk (binär)
ECOUNT:   DS      1               ;Einträge im Sektor (0-3)
;
SECTOR:   DS      2               ;laufender Sektor
TRACK:    DS      2               ;laufende Spur
E5FLAG:   DS      1               ;nicht-initialisiert, wenn 1
BLKCNT:   DS      2               ;Blöcke benutzt
;
HERE:                             ;ersetze durch ASEG:
          ORG     (HERE AND 0FF00H) + 100H ;ORG 0A00H
                  fuer Microsoft
BMAP:     DS      1               ;Block-Zuordnungs-Tabelle
;
          END     START
```

Abb. 8.14: Das Programm DISK zum Anzeigen der Disk-Parameter und der Block-Zuordnungs-Tabelle (Forts.)

ZUSAMMENFASSUNG

In diesem Kapitel haben wir das CP/M-Disketten-Inhaltsverzeichnis genauer untersucht. Wir haben ein Programm geschrieben, das die Diskettenparameter und die Verzeichniseinträge inklusive der Blocknummern ausgibt und eine Block-Zuordnungs-Tabelle aufbaut. Man kann dieses Programm auch noch weiter ausbauen. Z. B. kann man eine gelöschte Datei wiedergewinnen, indem man den Wert am Beginn des FCB von E5 in 0 ändert.

Mehrfachbezüge auf eine einzelne Datei können nützlich sein, wenn mehr als ein Benutzerbereich aktiv ist. Normalerweise müßte man ein Programm, das von mehreren Benutzern gebraucht wird, für jeden Benutzer kopieren. Das aber erfordert zusätzlichen Diskettenspeicher. Andererseits kann man Platz sparen, wenn man mehrere FCBs für eine Datei erzeugt. Ein FCB ist anhand des ersten Bytes des FCB zugewiesen,

der übrige Teil der FCB ist jedoch derselbe. So bezeichnen alle diese FCBs dieselbe Datei. (Diese Funktionen sind in einem Dienstprogramm für Disketten namens FILEFIX enthalten, das man käuflich erwerben kann.)

Das Verzeichnis unserer Makrobibliothek sollte jetzt so aussehen:

```
;;Makros in dieser Bibliothek                    Flags
;;ABORT   MACRO  CHAR                            CIFLAG,COFLAG
;;AMBIG   MACRO  OLD,NEW                         (keine)
;;BINBIN  MACRO                                  BNFLAG
;;CLOSE   MACRO  POINTR                          CLFLAG,COFLAG,CRFLAG,
;;                                               PRFLAG,OPFLAG,MVFLAG,
;;                                               DEFLAG,CIFLAG,UNFLAG,
;;                                               RNFLAG,S2FLAG
;;COMPAR  MACRO  FIRST,SECOND,BYTES              CMFLAG
;;COMPRA  MACRO  FIRST,SECOND,BYTES              CMFLAG
;;CPMVER  MACRO                                  (keine)
;;CRLF    MACRO                                  CRFLAG,COFLAG
;;DELETE  MACRO  POINTR,WHERE                    DEFLAG,CIFLAG,
;;                                               COFLAG,PRFLAG,UNFLAG
;;DIVIDE  MACRO  DENOM                           DVFLAG
;;ENTER   MACRO                                  (keine)
;;ERRORM  MACRO  TEXT,WHERE                      COFLAG,CRFLAG,PRFLAG
;;EXIT    MACRO  SPACE?                          (keine)
;;FILL    MACRO  ADDR,BYTES,CHAR                 FLFLAG
;;FILLD   MACRO  ADDR,BYTES,CHAR                 FLFLAG
;;GFNAME  MACRO  FCB                             FNFLAG,FLFLAG,RCFLAG,
;;                                               COFLAG,CRFLAG,PRFLAG
;;HEXHL   MACRO  POINTR                          HXFLAG,RCFLAG
;;HLDEC   MACRO                                  DEFLAG,COFLAG
;;LCHAR   MACRO  PAR                             LOFLAG
;;LDFILE  MACRO  FCB,POINTR,CHAR                 COFLAG,DMFLAG,RDFLAG
;;MAKE    MACRO  POINTR                          MKFLAG,COFLAG,CRFLAG,
;;                                               PRFLAG
;;MOVE    MACRO  FROM,TO,BYTES                   MVFLAG
;;MULT    MACRO  TIMES                           MLFLAG
;;OPEN    MACRO  POINTR,WHERE                    OPFLAG,COFLAG,PRFLAG,
;;                                               ORFLAG
;;OUTHEX  MACRO  REG                             CXFLAG,COFLAG
;;OUTHL   MACRO                                  CXFLAG,COFLAG
;;PCHAR   MACRO  PAR                             COFLAG
;;PFNAME  MACRO  FCB                             COFLAG,PRFLAG
;;PRINT   MACRO  TEXT,BYTES                      PRFLAG,COFLAG
```

```
;;PROTEC  MACRO  POINTR          (keine)
;;READB   MACRO  BUFFR           RCFLAG
;;READCH  MACRO  REG             CIFLAG,COFLAG
;;READS   MACRO  POINTR,STAR     RDFLAG,COFLAG
;;RENAME  MACRO  POINTR          RNFLAG,COFLAG
;;                               PRFLAG,CRFLAG
;;SBC     MACRO                  (keine)
;;SETDMA  MACRO  POINTR          DMFLAG
;;SETUP2  MACRO                  S2FLAG,CIFLAG,COFLAG,
;;                               CRFLAG,CMFLAG,DEFLAG,
;;                               MKFLAG,MVFLAG,OPFLAG,
;;                               PRFLAG,UNFLAG
;;SYSF    MACRO  FUNC,AE         (keine)
;;UCASE   MACRO  REG             (keine)
;;UNPROT  MACRO  POINTR          UNFLAG
;;UPPER   MACRO  REG             (keine)
;;VERSN   MACRO  NUM             (keine)
;;WRITES  MACRO  POINTR,STAR     WRFLAG,COFLAG,PRFLAG
;;WRFILE  MACRO  FCB,POINTR      COFLAG,CRFLAG,DMFLAG,
;;                               WRFLAG
```

Anhang A

Der ASCII-Zeichensatz

Der ASCII-Zeichensatz ist hier in numerischer Folge mit den entsprechenden dezimalen, hexadezimalen und oktalen Werten gelistet. Die Control-Zeichen sind mit einem Dach (^) gekennzeichnet.
Z.B. ist der Tabulator TAB (HT) als Control-I (^I) dargestellt.

ASCII Symbol	Dezimal Wert	Hex Wert	Oktal Wert	Control Zeichen	Bedeutung
NUL	0	00	000	^§	Null
SOH	1	01	001	^A	Start of heading
STX	2	02	002	^B	Start of text
ETX	3	03	003	^C	End of text
EOT	4	04	004	^D	End of transmission
ENQ	5	05	005	^E	Inquiry
ACK	6	06	006	^F	Acknowledge
BEL	7	07	007	^G	Bell
BS	8	08	010	^H	Backspace
HT	9	09	011	^I	Horizontal tab
LF	10	0A	012	^J	Line feed
VT	11	0B	013	^K	Vertical tab
FF	12	0C	014	^L	Form feed
CR	13	0D	015	^M	Carriage return
SO	14	0E	016	^N	Shift out
SI	15	0F	017	^O	Shift in
DLE	16	10	020	^P	Data link escape
DC1	17	11	021	^Q	Device control 1
DC2	18	12	022	^R	Device control 2
DC3	19	13	023	^S	Device control 3
DC4	20	14	024	^T	Device control 4
NAK	21	15	025	^U	Negative acknowledge
SYN	22	16	026	^V	Synchronous idle
ETB	23	17	027	^W	End of transmission block

ASCII Symbol	Dezimal Wert	Hex Wert	Oktal Wert	Control Zeichen	Bedeutung
CAN	24	18	030	^X	Cancel
EM	25	19	031	^Y	End of medium
SUB	26	1A	032	^Z	Substitute
ESC	27	1B	033	^[	Escape
FS	28	1C	034	^\	File separator
GS	29	1D	035	^]	Group separator
RS	30	1E	036	^^	Record separator
US	31	1F	037	^_	Unit separator
SP	32	20	040		Leerzeichen
!	33	21	041		
"	34	22	042		
#	35	23	043		
$	36	24	044		
%	37	25	045		
&	38	26	046		
'	39	27	047		
(	40	28	050		
)	41	29	051		
*	42	2A	052		
+	43	2B	053		
,	44	2C	054		
--	45	2D	055		Minus
.	46	2E	056		
/	47	2F	057		
0	48	30	060		
1	49	31	061		
2	50	32	062		
3	51	33	063		
4	52	34	064		
5	53	35	065		
6	54	36	066		
7	55	37	067		
8	56	38	070		
9	57	39	071		
:	58	3A	072		
;	59	3B	073		
<	60	3C	074		

ASCII Symbol	Dezimal Wert	Hex Wert	Oktal Wert	Control Zeichen	Bedeutung
=	61	3D	075		
>	62	3E	076		
?	63	3F	077		
§	64	40	100		
A	65	41	101		
B	66	42	102		
C	67	43	103		
D	68	44	104		
E	69	45	105		
F	70	46	106		
G	71	47	107		
H	72	48	110		
I	73	49	111		
J	74	4A	112		
K	75	4B	113		
L	76	4C	114		
M	77	4D	115		
N	78	4E	116		
O	79	4F	117		
P	80	50	120		
Q	81	51	121		
R	82	52	122		
S	83	53	123		
T	84	54	124		
U	85	55	125		
V	86	56	126		
W	87	57	127		
X	88	58	130		
Y	89	59	131		
Z	90	5A	132		
[	91	5B	133		
\	92	5C	134		
]	93	5D	135		
^	94	5E	136		
_	95	5F	137		Unterstrich
`	96	60	140		
a	97	61	141		

ASCII Symbol	Dezimal Wert	Hex Wert	Oktal Wert	Control Zeichen	Bedeutung
b	98	62	142		
c	99	63	143		
d	100	64	144		
e	101	65	145		
f	102	66	146		
g	103	67	147		
h	104	68	150		
i	105	69	151		
j	106	6A	152		
k	107	6B	153		
l	108	6C	154		
m	109	6D	155		
n	110	6E	156		
o	111	6F	157		
p	112	70	160		
q	113	71	161		
r	114	72	162		
s	115	73	163		
t	116	74	164		
u	117	75	165		
v	118	76	166		
w	119	77	167		
x	120	78	170		
y	121	79	171		
z	122	7A	172		
{	123	7B	173		
|	124	7C	174		
}	125	7D	175		
~	126	7E	176		
DEL	127	7F	177		Löschzeichen

Anhang B
Eine 64K-Speicher-Liste

Die Prozessoren 8080 und Z80 können 64K Bytes direkt adressieren. Der Speicherbereich ist in der folgenden Tabelle dargestellt. Jeder Eintrag repräsentiert einen 256-Byte-Block. Das höhere Byte der Adresse ist hexadezimal als erster Wert angegeben, dann oktal. Z.B. repräsentiert der Eintrag

Hex	*Oct*	*K*	*Bl*
20	040		32

eine Adresse im Bereich von 2000 bis 20FF hex, oder 040-000 bis 040-777 oktal. Die dritte Spalte zeigt die dezimale Zahl der 1K-Blöcke. Die vierte Spalte ist die dezimale Zahl von 256-Byte-Blöcken ab Adresse 100 hex. Z.B. angenommen, ein CP/M-Programm läuft von 100 hex bis 3035 hex. Der Eintrag 30 hex in der Tabelle sagt aus, daß das Programm 48 dezimale Blöcke zu 256 Bytes enthält. Das Programm kann mit dem CP/M-Kommando

```
A>SAVE 48 (Dateiname)
```

gespeichert werden. Oder wenn man z.B. zwei 16K-Speicherkarten hat ab Adresse 0, dann ist die höchste Speicheradresse 7FFF hex.

Hex	Okt	K	Bl
00	000		0
01	001		1
02	002		2
03	003	1	3
04	004		4
05	005		5
06	006		6
07	007	2	7

Hex	Okt	K	Bl
08	010		8
09	011		9
0A	012		10
0B	013	3	11
0C	014		12
0D	015		13
0E	016		14
0F	017	4	15

Hex	Okt	K	Bl
10	020		16
11	021		17
12	022		18
13	023	5	19
14	024		20
15	025		21
16	026		22
17	027	6	23
18	030		24
19	031		25
1A	032		26
1B	033	7	27
1C	034		28
1D	035		29
1E	036		30
1F	037	8	31
20	040		32
21	041		33
22	042		34
23	043	9	35
24	044		36
25	045		37
26	046		38
27	047	10	39
28	050		40
29	051		41
2A	052		42
2B	053	11	43
2C	054		44
2D	055		45
2E	056		46
2F	057	12	47
30	060		48
31	061		49
32	062		50
33	063	13	51
34	064		52
35	065		53

Hex	Okt	K	Bl
36	066		54
37	067	14	55
38	070		56
39	071		57
3A	072		58
3B	073	15	59
3C	074		60
3D	075		61
3E	076		62
3F	077	16	63
40	100		64
41	101		65
42	102		66
43	103	17	67
44	104		68
45	105		69
46	106		70
47	107	18	71
48	110		72
49	111		73
4A	112		74
4B	113	19	75
4C	114		76
4D	115		77
4E	116		78
4F	117	20	79
50	120		80
51	121		81
52	122		82
53	123	21	83
54	124		84
55	125		85
56	126		86
57	127	22	87
58	130		88
59	131		89
5A	132		90
5B	133	23	91

Hex	Okt	K	Bl
5C	134		92
5D	135		93
5E	136		94
5F	137	24	95
60	140		96
61	141		97
62	142		98
63	143	25	99
64	144		100
65	145		101
66	146		102
67	147	26	103
68	150		104
69	151		105
6A	152		106
6B	153	27	107
6C	154		108
6D	155		109
6E	156		110
6F	157	28	111
70	160		112
71	161		113
72	162		114
73	163	29	115
74	164		116
75	165		117
76	166		118
77	167	30	119
78	170		120
79	171		121
7A	172		122
7B	173	31	123
7C	174		124
7D	175		125
7E	176		126
7F	177	32	127
80	200		128
81	201		129
82	202		130
83	203	33	131
84	204		132
85	205		133
86	206		134
87	207	34	135
88	210		136
89	211		137
8A	212		138
8B	213	35	139
8C	214		140
8D	215		141
8E	216		142
8F	217	36	143
90	220		144
91	221		145
92	222		146
93	223	37	147
94	224		148
95	225		149
96	226		150
97	227	38	151
98	230		152
99	231		153
9A	232		154
9B	233	39	155
9C	234		156
9D	235		157
9E	236		158
9F	237	40	159
A0	240		160
A1	241		161
A2	242		162
A3	243	41	163
A4	244		164
A5	245		165
A6	246		166
A7	247	42	167

Hex	Okt	K	Bl
A8	250		168
A9	251		169
AA	252		170
AB	253	43	171
AC	254		172
AD	255		173
AE	256		174
AF	257	44	175
B0	260		176
B1	261		177
B2	262		178
B3	263	45	179
B4	264		180
B5	265		181
B6	266		182
B7	267	46	183
B8	270		184
B9	271		185
BA	272		186
BB	273	47	187
BC	274		188
BD	275		189
BE	276		190
BF	227	48	191
C0	300		192
C1	301		193
C2	302		194
C3	303	49	195
C4	304		196
C5	305		197
C6	306		198
C7	307	50	199
C8	310		200
C9	311		201
CA	312		202
CB	313	51	203
CC	314		204
CD	315		205

Hex	Okt	K	Bl
CE	316		206
CF	317	52	207
D0	320		208
D1	321		209
D2	322		210
D3	323	53	211
D4	324		212
D5	325		213
D6	326		214
D7	327	54	215
D8	330		216
D9	331		217
DA	332		218
DB	333	55	219
DC	334		220
DD	335		221
DE	336		222
DF	337	56	223
E0	340		224
E1	341		225
E2	342		226
E3	343	57	227
E4	344		228
E5	345		229
E6	346		230
E7	347	58	231
E8	350		232
E9	351		233
EA	352		234
EB	353	59	235
EC	354		236
ED	355		237
EE	356		238
EF	357	60	239
F0	360		240
F1	361		241
F2	362		242

Hex	Okt	K	Bl
F3	363	61	243
F4	364		244
F5	365		245
F6	366		246
F7	367	62	247
F8	370		248
F9	371		249
FA	372		250
FB	373	63	251
FC	374		252
FD	375		253
FE	376		254
FF	377	64	255

Anhang C
Der 8080-Befehlssatz *(alphabetisch)*

Der 8080-Befehlssatz ist alphabetisch mit dem entsprechenden hexadezimalen Code gelistet. Es gilt:

nn 8-bit-Parameter
nnnn 16-bit-Parameter

Hex		Befehl		Hex		Befehl	
CE	nn	ACI	nn	A3		ANA	E
8F		ADC	A	A4		ANA	H
88		ADC	B	A5		ANA	L
89		ADC	C	A6		ANA	M
8A		ADC	D	E6	nn	ANI	nn
8B		ADC	E	CD	nnnn	CALL	nnnn
8C		ADC	H	DC	nnnn	CC	nnnn
8D		ADC	L	FC	nnnn	CM	nnnn
8E		ADC	M	2F		CMA	
87		ADD	A	3F		CMC	
80		ADD	B	BF		CMP	A
81		ADD	C	B8		CMP	B
82		ADD	D	B9		CMP	C
83		ADD	E	BA		CMP	D
84		ADD	H	BB		CMP	E
85		ADD	L	BC		CMP	H
86		ADD	M	BD		CMP	L
C6	nn	ADI	nn	BE		CMP	M
A7		ANA	A	D4	nnnn	CNC	nnnn
A0		ANA	B	C4	nnnn	CNZ	nnnn
A1		ANA	C	F4	nnnn	CP	nnnn
A2		ANA	D	EC	nnnn	CPE	nnnn

Hex		Befehl		Hex		Befehl	
FE	nn	CPI	nn	C2	nnnn	JNZ	nnnn
E4	nnnn	CPO	nnnn	F2	nnnn	JP	nnnn
CC	nnnn	CZ	nnnn	EA	nnnn	JPE	nnnn
27		DAA		E2	nnnn	JPO	nnnn
09		DAD	B	CA	nnnn	JZ	nnnn
19		DAD	D	3A	nnnn	LDA	nnnn
29		DAD	H	0A		LDAX	B
39		DAD	SP	1A		LDAX	D
3D		DCR	A	2A	nnnn	LHLD	nnnn
05		DCR	B	01	nnnn	LXI	B,nnnn
0D		DCR	C	11	nnnn	LXI	D,nnnn
15		DCR	D	21	nnnn	LXI	H,nnnn
1D		DCR	E	31	nnnn	LXI	SP,nnnn
25		DCR	H	7F		MOV	A,A
2D		DCR	L	78		MOV	A,B
35		DCR	M	79		MOV	A,C
0B		DCX	B	7A		MOV	A,D
1B		DCX	D	7B		MOV	A,E
2B		DCX	H	7C		MOV	A,H
3B		DCX	SP	7D		MOV	A,L
F3		DI		7E		MOV	A,M
FB		EI		47		MOV	B,A
76		HLT		40		MOV	B,B
DB	nn	IN	nn	41		MOV	B,C
3C		INR	A	42		MOV	B,D
04		INR	B	43		MOV	B,E
0C		INR	C	44		MOV	B,H
14		INR	D	45		MOV	B,L
1C		INR	E	46		MOV	B,M
24		INR	H	4F		MOV	C,A
2C		INR	L	48		MOV	C,B
34		INR	M	49		MOV	C,C
03		INX	B	4A		MOV	C,D
13		INX	D	4B		MOV	C,E
23		INX	H	4C		MOV	C,H
33		INX	SP	4D		MOV	C,L
DA	nnnn	JC	nnnn	4E		MOV	C,M
FA	nnnn	JM	nnnn	57		MOV	D,A
C3	nnnn	JMP	nnnn	50		MOV	D,B
D2	nnnn	JNC	nnnn	51		MOV	D,C

Hex		Befehl		Hex		Befehl	
52		MOV	D,D	1E	nn	MVI	E,nn
53		MOV	D,E	26	nn	MVI	H,nn
54		MOV	D,H	2E	nn	MVI	L,nn
55		MOV	D,L	36	nn	MVI	M,nn
56		MOV	D,M	00		NOP	
5F		MOV	E,A	B7		ORA	A
58		MOV	E,B	B0		ORA	B
59		MOV	E,C	B1		ORA	C
5A		MOV	E,D	B2		ORA	D
5B		MOV	E,E	B3		ORA	E
5C		MOV	E,H	B4		ORA	H
5D		MOV	E,L	B5		ORA	L
5E		MOV	E,M	B6		ORA	M
67		MOV	H,A	F6	nn	ORI	nn
60		MOV	H,B				
61		MOV	H,C	D3	nn	OUT	nn
62		MOV	H,D	E9		PCHL	
63		MOV	H,E	C1		POP	B
64		MOV	H,H	D1		POP	D
65		MOV	H,L	E1		POP	H
66		MOV	H,M	F1		POP	PSW
6F		MOV	L,A	C5		PUSH	B
68		MOV	L,B	D5		PUSH	D
69		MOV	L,C	E5		PUSH	H
6A		MOV	L,D	F5		PUSH	PSW
6B		MOV	L,E	17		RAL	
6C		MOV	L,H	1F		RAR	
6D		MOV	L,L	D8		RC	
6E		MOV	L,M	C9		RET	
77		MOV	M,A	07		RLC	
70		MOV	M,B	F8		RM	
71		MOV	M,C	D0		RNC	
72		MOV	M,D	C0		RNZ	
73		MOV	M,E	F0		RP	
74		MOV	M,H	E8		RPE	
75		MOV	M,L	E0		RPO	
3E	nn	MVI	A,nn	0F		RRC	
06	nn	MVI	B,nn	C7		RST	0
0E	nn	MVI	C,nn	CF		RST	1
16	nn	MVI	D,nn	D7		RST	2

Hex		Befehl		Hex		Befehl	
DF		RST	3	97		SUB	A
E7		RST	4	90		SUB	B
EF		RST	5	91		SUB	C
F7		RST	6	92		SUB	D
FF		RST	7	93		SUB	E
C8		RZ		94		SUB	H
9F		SBB	A	95		SUB	L
98		SBB	B	96		SUB	M
99		SBB	C	D6	nn	SUI	nn
9A		SBB	D	EB		XCHG	
9B		SBB	E	AF		XRA	A
9C		SBB	H	A8		XRA	B
9D		SBB	L	A9		XRA	C
9E		SBB	M	AA		XRA	D
DE	nn	SBI	nn	AB		XRA	E
22	nnnn	SHLD	nnnn	AC		XRA	H
F9		SPHL		AD		XRA	L
32	nnnn	STA	nnnn	AE		XRA	M
02		STAX	B	EE	nn	XRI	nn
12		STAX	D	E3		XTHL	
37		STC					

Anhang D
Der 8080-Befehlssatz *(numerisch)*

Der 8080-Befehlssatz ist in numerischer Folge mit dem entsprechenden hexadezimalen Code gelistet. Es gilt:

nn 8-bit-Parameter
nnnn 16-bit-Parameter

Hex		Befehl		Hex		Befehl	
00		NOP		16	nn	MVI	D,nn
01	nnnn	LXI	B,nnnn	17		RAL	
02		STAX	B	18		<unbenutzt>	
03		INX	B	19		DAD	D
04		INR	B	1A		LDAX	D
05		DCR	B	1B		DCX	D
06	nn	MVI	B,nn	1C		INR	E
07		RLC		1D		DCR	E
08		<unbenutzt>		1E	nn	MVI	E,nn
09		DAD	B	1F		RAR	
0A		LDAX	B	20		<unbenutzt>	
0B		DCX	B	21	nnnn	LXI	H,nnnn
0C		INR	C	22	nnnn	SHLD	nnnn
0D		DCR	C	23		INX	H
0E	nn	MVI	C,nn	24		INR	H
0F		RRC		25		DCR	H
10		<unbenutzt>		26	nn	MVI	H,nn
11	nnnn	LXI	D,nnnn	27		DAA	
12		STAX	D	28		<unbenutzt>	
13		INX	D	29		DAD	H
14		INR	D	2A	nnnn	LHLD	nnnn
15		DCR	D	2B		DCX	H

Hex		Befehl		Hex	Befehl	
2C		INR	L	54	MOV	D,H
2D		DCR	L	55	MOV	D,L
2E	nn	MVI	L,nn	56	MOV	D,M
2F		CMA		57	MOV	D,A
30		<unbenutzt>		58	MOV	E,B
31	nnnn	LXI	SP,nnnn	59	MOV	E,C
32	nnnn	STA	nnnn	5A	MOV	E,D
33		INX	SP	5B	MOV	E,E
34		INR	M	5C	MOV	E,H
35		DCR	M	5D	MOV	E,L
36	nn	MVI	M,nn	5E	MOV	E,M
37		STC		5F	MOV	E,A
38		<unbenutzt>		60	MOV	H,B
39		DAD	SP	61	MOV	H,C
3A	nnnn	LDA	nnnn	62	MOV	H,D
3B		DCX	SP	63	MOV	H,E
3C		INR	A	64	MOV	H,H
3D		DCR	A	65	MOV	H,L
3E	nn	MVI	A,nn	66	MOV	H,M
3F		CMC		67	MOV	H,A
40		MOV	B,B	68	MOV	L,B
41		MOV	B,C	69	MOV	L,C
42		MOV	B,D	6A	MOV	L,D
43		MOV	B,E	6B	MOV	L,E
44		MOV	B,H	6C	MOV	L,H
45		MOV	B,L	6D	MOV	L,L
46		MOV	B,M	6E	MOV	L,M
47		MOV	B,A	6F	MOV	L,A
48		MOV	C,B	70	MOV	M,B
49		MOV	C,C	71	MOV	M,C
4A		MOV	C,D	72	MOV	M,D
4B		MOV	C,E	73	MOV	M,E
4C		MOV	C,H	74	MOV	M,H
4D		MOV	C,L	75	MOV	M,L
4E		MOV	C,M	76	HLT	
4F		MOV	C,A	77	MOV	M,A
50		MOV	D,B	78	MOV	A,B
51		MOV	D,C	79	MOV	A,C
52		MOV	D,D	7A	MOV	A,D
53		MOV	D,E	7B	MOV	A,E

Hex	Befehl		Hex	Befehl	
7C	MOV	A,H	A4	ANA	H
7D	MOV	A,L	A5	ANA	L
7E	MOV	A,M	A6	ANA	M
7F	MOV	A,A	A7	ANA	A
80	ADD	B	A8	XRA	B
81	ADD	C	A9	XRA	C
82	ADD	D	AA	XRA	D
83	ADD	E	AB	XRA	E
84	ADD	H	AC	XRA	H
85	ADD	L	AD	XRA	L
86	ADD	M	AE	XRA	M
87	ADD	A	AF	XRA	A
88	ADC	B	B0	ORA	B
89	ADC	C	B1	ORA	C
8A	ADC	D	B2	ORA	D
8B	ADC	E	B3	ORA	E
8C	ADC	H	B4	ORA	H
8D	ADC	L	B5	ORA	L
8E	ADC	M	B6	ORA	M
8F	ADC	A	B7	ORA	A
90	SUB	B	B8	CMP	B
91	SUB	C	B9	CMP	C
92	SUB	D	BA	CMP	D
93	SUB	E	BB	CMP	E
94	SUB	H	BC	CMP	H
95	SUB	L	BD	CMP	L
96	SUB	M	BE	CMP	M
97	SUB	A	BF	CMP	A
98	SBB	B	C0	RNZ	
99	SBB	C	C1	POP	B
9A	SBB	D	C2 nnnn	JNZ	nnnn
9B	SBB	E	C3 nnnn	JMP	nnnn
9C	SBB	H	C4 nnnn	CNZ	nnnn
9D	SBB	L	C5	PUSH	B
9E	SBB	M	C6 nn	ADI	nn
9F	SBB	A	C7	RST	0
A0	ANA	B	C8	RZ	
A1	ANA	C	C9	RET	
A2	ANA	D	CA nnnn	JZ	nnnn
A3	ANA	E	CB	<unbenutzt>	

Hex		Befehl		Hex		Befehl	
CC	nnnn	CZ	nnnn	E6	nn	ANI	nn
CD	nnnn	CALL	nnnn	E7		RST	4
CE	nn	ACI	nn	E8		RPE	
CF		RST	1	E9		PCHL	
D0		RNC		EA	nnnn	JPE	nnnn
D1		POP	D	EB		XCHG	
D2	nnnn	JNC	nnnn	EC	nnnn	CPE	nnnn
D3	nn	OUT	nn	ED		<unbenutzt>	
D4	nnnn	CNC	nnnn	EE	nn	XRI	nn
D5		PUSH	D	EF		RST	5
D6	nn	SUI	nn	F0		RP	
D7		RST	2	F1		POP	PSW
D8		RC		F2	nnnn	JP	nnnn
D9		<unbenutzt>		F3		DI	
DA	nnnn	JC	nnnn	F4	nnnn	CP	nnnn
DB	nn	IN	nn	F5		PUSH	PSW
DC	nnnn	CC	nnnn	F6	nn	ORI	nn
DD		<unbenutzt>		F7		RST	6
DE	nn	SBI	nn	F8		RM	
DF		RST	3	F9		SPHL	
E0		RPO		FA	nnnn	JM	nnnn
E1		POP	H	FB		EI	
E2	nnnn	JPO	nnnn	FC	nnnn	CM	nnnn
E3		XTHL		FD		<unbenutzt>	
E4	nnnn	CPO	nnnn	FE	nn	CPI	nn
E5		PUSH	H	FF		RST	7

Anhang E

Der Z80-Befehlssatz *(alphabetisch)*

Der Befehlssatz Zilog Z80 ist in alphabetischer Folge mit den entsprechenden hexadezimalen Werten gelistet. Es gilt:

nn	8-Bit-Parameter
nnnn	16-Bit-Parameter
dd	8-Bit-Distanz mit Vorzeichen
*	Mit dem 8080 gemeinsame Befehle

Hex			Befehl		Hex			Befehl	
8E		*	ADC	A,(HL)	81		*	ADD	A,C
DD	8Edd		ADC	A,(IX+dd)	82		*	ADD	A,D
FD	8Edd		ADC	A,(IY+dd)	83		*	ADD	A,E
8F		*	ADC	A,A	84		*	ADD	A,H
88		*	ADC	A,B	85		*	ADD	A,L
89		*	ADC	A,C	C6	nn	*	ADD	A,nn
8A		*	ADC	A,D	09		*	ADD	HL,BC
8B		*	ADC	A,E	19		*	ADD	HL,DE
8C		*	ADC	A,H	29		*	ADD	HL,HL
8D		*	ADC	A,L	39		*	ADD	HL,SP
CE	nn	*	ADC	A,nn	DD	09		ADD	IX,BC
ED	4A		ADC	HL,BC	DD	19		ADD	IX,DE
ED	5A		ADC	HL,DE	DD	29		ADD	IX,IX
ED	6A		ADC	HL,HL	DD	39		ADD	IX,SP
ED	7A		ADC	HL,SP	FD	09		ADD	IY,BC
86		*	ADD	A,(HL)	FD	19		ADD	IY,DE
DD	86dd		ADD	A,(IX+dd)	FD	29		ADD	IY,IY
FD	86dd		ADD	A,(IY+dd)	FD	39		ADD	IY,SP
87		*	ADD	A,A	A6		*	AND	(HL)
80		*	ADD	A,B	DD	A6dd		AND	(IX+dd)

Hex		Befehl		Hex		Befehl	
FD	A6dd	AND	(IY+dd)	DD	CBdd5E	BIT	3,(IX+dd)
A7		* AND	A	FD	CBdd5E	BIT	3,(IY+dd)
A0		* AND	B	CB	5F	BIT	3,A
A1		* AND	C	CB	58	BIT	3,B
A2		* AND	D	CB	59	BIT	3,C
A3		* AND	E	CB	5A	BIT	3,D
A4		* AND	H	CB	5B	BIT	3,E
A5		* AND	L	CB	5C	BIT	3,H
E6	nn	* AND	nn	CB	5D	BIT	3,L
CB	46	BIT	0,(HL)	CB	66	BIT	4,(HL)
DD	CBdd46	BIT	0,(IX+dd)	DD	CBdd66	BIT	4,(IX+dd)
FD	CBdd46	BIT	0,(IY+dd)	FD	CBdd66	BIT	4,(IY+dd)
CB	47	BIT	0,A	CB	67	BIT	4,A
CB	40	BIT	0,B	CB	60	BIT	4,B
CB	41	BIT	0,C	CB	61	BIT	4,C
CB	42	BIT	0,D	CB	62	BIT	4,D
CB	43	BIT	0,E	CB	63	BIT	4,E
CB	44	BIT	0,H	CB	64	BIT	4,H
CB	45	BIT	0,L	CB	65	BIT	4,L
CB	4E	BIT	1,(HL)	CB	6E	BIT	5,(HL)
DD	CBdd4E	BIT	1,(IX+dd)	DD	CBdd6E	BIT	5,(IX+dd)
FD	CBdd4E	BIT	1,(IY+dd)	FD	CBdd6E	BIT	5,(IY+dd)
CB	4F	BIT	1,A	CB	6F	BIT	5,A
CB	48	BIT	1,B	CB	68	BIT	5,B
CB	49	BIT	1,C	CB	69	BIT	5,C
CB	4A	BIT	1,D	CB	6A	BIT	5,D
CB	4B	BIT	1,E	CB	6B	BIT	5,E
CB	4C	BIT	1,H	CB	6C	BIT	5,H
CB	4D	BIT	1,L	CB	6D	BIT	5,L
CB	56	BIT	2,(HL)	CB	76	BIT	6,(HL)
DD	CBdd56	BIT	2,(IX+dd)	DD	CBdd76	BIT	6,(IX+dd)
FD	CBdd56	BIT	2,(IY+dd)	FD	CBdd76	BIT	6,(IY+dd)
CB	57	BIT	2,A	CB	77	BIT	6,A
CB	50	BIT	2,B	CB	70	BIT	6,B
CB	51	BIT	2,C	CB	71	BIT	6,C
CB	52	BIT	2,D	CB	72	BIT	6,D
CB	53	BIT	2,E	CB	73	BIT	6,E
CB	54	BIT	2,H	CB	74	BIT	6,H
CB	55	BIT	2,L	CB	75	BIT	6,L
CB	5E	BIT	3,(HL)	CB	7E	BIT	7,(HL)

Hex		Befehl		Hex		Befehl	
DD	CBdd7E	BIT	7,(IX+dd)	05		* DEC	B
FD	CBdd7E	BIT	7,(IY+dd)	0B		* DEC	BC
CB	7F	BIT	7,A	0D		* DEC	C
CB	78	BIT	7,B	15		* DEC	D
CB	79	BIT	7,C	1B		* DEC	DE
CB	7A	BIT	7,D	1D		* DEC	E
CB	7B	BIT	7,E	25		* DEC	H
CB	7C	BIT	7,H	2B		* DEC	HL
CB	7D	BIT	7,L	DD	2B	DEC	IX
DC	nnnn	* CALL	C,nnnn	FD	2B	DEC	IY
FC	nnnn	* CALL	M,nnnn	2D		* DEC	L
D4	nnnn	* CALL	NC,nnnn	3B		* DEC	SP
CD	nnnn	* CALL	nnnn	F3		* DI	
C4	nnnn	* CALL	NZ,nnnn	10	dd	DJNZ	dd
F4	nnnn	* CALL	P,nnnn	FB		* EI	
EC	nnnn	* CALL	PE,nnnn	E3		* EX	(SP),HL
E4	nnnn	* CALL	PO,nnnn	DD	E3	EX	(SP),IX
CC	nnnn	* CALL	Z,nnnn	FD	E3	EX	(SP),IY
3F		* CCF		08		EX	AF,AF'
BE		* CP	(HL)	EB		* EX	DE,HL
DD	BEdd	CP	(IX+dd)	D9		EXX	
FD	BEdd	CP	(IY+dd)	76		* HALT	
BF		* CP	A	ED	46	IM	0
B8		* CP	B	ED	56	IM	1
B9		* CP	C	ED	5E	IM	2
BA		* CP	D	ED	78	IN	A,(C)
BB		* CP	E	DB	nn	* IN	A,(nn)
BC		* CP	H	ED	40	IN	B,(C)
BD		* CP	L	ED	48	IN	C,(C)
FE	nn	* CP	nn	ED	50	IN	D,(C)
ED	A9	* CPD		ED	58	IN	E,(C)
ED	B9	* CPDR		ED	60	IN	H,(C)
ED	A1	* CPI		ED	68	IN	L,(C)
ED	B1	* CPIR		34		* INC	(HL)
2F		* CPL		DD	34dd	INC	(IX+dd)
27		* DAA		FD	34dd	INC	(IY+dd)
35		* DEC	(HL)	3C		* INC	A
DD	35dd	DEC	(IX+dd)	04		* INC	B
FD	35dd	DEC	(IY+dd)	03		* INC	BC
3D		* DEC	A	0C		* INC	C

Hex		Befehl		Hex		Befehl	
14		* INC	D	DD	77dd	LD	(IX+dd),A
13		* INC	DE	DD	70dd	LD	(IX+dd),B
1C		* INC	E	DD	71dd	LD	(IX+dd),C
24		* INC	H	DD	72dd	LD	(IX+dd),D
23		* INC	HL	DD	73dd	LD	(IX+dd),E
DD	23	INC	IX	DD	74dd	LD	(IX+dd),H
FD	23	INC	IY	DD	75dd	LD	(IX+dd),L
2C		* INC	L	DD	36ddnn	LD	(IX+dd),nn
33		* INC	SP	FD	77dd	LD	(IY+dd),A
ED	AA	IND		FD	70dd	LD	(IY+dd),B
ED	BA	INDR		FD	71dd	LD	(IY+dd),C
ED	A2	INI		FD	72dd	LD	(IY+dd),D
ED	B2	INIR		FD	73dd	LD	(IY+dd),E
E9		* JP	(HL)	FD	74dd	LD	(IY+dd),H
DD	E9	JP	(IX)	FD	75dd	LD	(IY+dd),L
FD	E9	JP	(IY)	FD	36ddnn	LD	(IY+dd),nn
DA	nnnn	* JP	C,nnnn	32	nnnn	* LD	(nnnn),A
FA	nnnn	* JP	M,nnnn	ED	43nnnn	LD	(nnnn),BC
D2	nnnn	* JP	NC,nnnn	ED	53nnnn	LD	(nnnn),DE
C3	nnnn	* JP	nnnn	22	nnnn	* LD	(nnnn),HL
C2	nnnn	* JP	NZ,nnnn	DD	22nnnn	LD	(nnnn),IX
F2	nnnn	* JP	P,nnnn	FD	22nnnn	LD	(nnnn),IY
EA	nnnn	* JP	PE,nnnn	ED	73nnnn	LD	(nnnn),SP
E2	nnnn	* JP	PO,nnnn	0A		* LD	A,(BC)
CA	nnnn	* JP	Z,nnnn	1A		* LD	A,(DE)
38	dd	JR	C,dd	7E		* LD	A,(HL)
18	dd	JR	dd	DD	7Edd	LD	A,(IX+dd)
30	dd	JR	NC,dd	FD	7Edd	LD	A,(IY+dd)
20	dd	JR	NZ,dd	3A	nnnn	* LD	A,(nnnn)
28	dd	JR	Z,dd	7F		* LD	A,A
02		* LD	(BC),A	78		* LD	A,B
12		* LD	(DE),A	79		* LD	A,C
77		* LD	(HL),A	7A		* LD	A,D
70		* LD	(HL),B	7B		* LD	A,E
71		* LD	(HL),C	7C		* LD	A,H
72		* LD	(HL),D	ED	57	LD	A,I
73		* LD	(HL),E	7D		* LD	A,L
74		* LD	(HL),H	3E	nn	* LD	A,nn
75		* LD	(HL),L	ED	5F	LD	A,R
36	nn	* LD	(HL),nn	46		* LD	B,(HL)

Hex		Befehl		Hex		Befehl	
DD	46dd		LD B,(IX+dd)	58		* LD	E,B
FD	46dd		LD B,(IY+dd)	59		* LD	E,C
47		* LD	B,A	5A		* LD	E,D
40		* LD	B,B	5B		* LD	E,E
41		* LD	B,C	5C		* LD	E,H
42		* LD	B,D	5D		* LD	E,L
43		* LD	B,E	1E	nn	* LD	E,nn
44		* LD	B,H	66		* LD	H,(HL)
45		* LD	B,L	DD	66dd	LD	H,(IX+dd)
06	nn	* LD	B,n	FD	66dd	LD	H,(IY+dd)
ED	4Bnnnn	LD	BC,(nnnn)	67		* LD	H,A
01	nnnn	* LD	BC,nnnn	60		* LD	H,B
4E		* LD	C,(HL)	61		* LD	H,C
DD	4Edd	LD	C,(IX+dd)	62		* LD	H,D
FD	4Edd	LD	C,(IY+dd)	63		* LD	H,E
4F		* LD	C,A	64		* LD	H,H
48		* LD	C,B	65		* LD	H,L
49		* LD	C,C	26	nn	* LD	H,nn
4A		* LD	C,D	2A	nnnn	* LD	HL,(nnnn)
4B		* LD	C,E	21	nnnn	* LD	HL,nnnn
4C		* LD	C,H	ED	47	LD	I,A
4D		* LD	C,L	DD	2Annnn	LD	IX,(nnnn)
0E	nn	* LD	C,nn	DD	21nnnn	LD	IX,nnnn
56		* LD	D,(HL)	FD	2Annnn	LD	IY,(nnnn)
DD	56dd	LD	D,(IX+dd)	FD	21nnnn	LD	IY,nnnn
FD	56dd	LD	D,(IY+dd)	6E		* LD	L,(HL)
57		* LD	D,A	DD	6Edd	LD	L,(IX+dd)
50		* LD	D,B	FD	6Edd	LD	L,(IY+dd)
51		* LD	D,C	6F		* LD	L,A
52		* LD	D,D	68		* LD	L,B
53		* LD	D,E	69		* LD	L,C
54		* LD	D,H	6A		* LD	L,D
55		* LD	D,L	6B		* LD	L,E
16	nn	* LD	D,nn	6C		* LD	L,H
ED	5Bnnnn	LD	DE,(nnnn)	6D		* LD	L,L
11	nnnn	* LD	DE,nnnn	2E	nn	* LD	L,nn
5E		* LD	E,(HL)	ED	4F	LD	R,A
DD	5Edd	LD	E,(IX+dd)	ED	7Bnnnn	LD	SP,(nnnn)
FD	5Edd	LD	E,(IY+dd)	F9		* LD	SP,HL
5F		* LD	E,A	DD	F9	LD	SP,IX

Hex			Befehl	
FD	F9		LD	SP,IY
31	nnnn	*	LD	SP,nnnn
ED	A8		LDD	
ED	B8		LDDR	
ED	A0		LDI	
ED	B0		LDIR	
ED	44		NEG	
00		*	NOP	
B6		*	OR	(HL)
DD	B6dd		OR	(IX+dd)
FD	B6dd		OR	(IY+dd)
B7		*	OR	A
B0		*	OR	B
B1		*	OR	C
B2		*	OR	D
B3		*	OR	E
B4		*	OR	H
B5		*	OR	L
F6	nn	*	OR	nn
ED	BB		OTDR	
ED	B3		OTIR	
ED	79		OUT	(C),A
ED	41		OUT	(C),B
ED	49		OUT	(C),C
ED	51		OUT	(C),D
ED	59		OUT	(C),E
ED	61		OUT	(C),H
ED	69		OUT	(C),L
D3	nn	*	OUT	(nn),A
ED	AB		OUTD	
ED	A3		OUTI	
F1		*	POP	AF
C1		*	POP	BC
D1		*	POP	DE
E1		*	POP	HL
DD	E1		POP	IX
FD	E1		POP	IY
F5		*	PUSH	AF
C5		*	PUSH	BC
D5		*	PUSH	DE

Hex			Befehl	
E5		*	PUSH	HL
DD	E5		PUSH	IX
FD	E5		PUSH	IY
CB	86		RES	0,(HL)
DD	CBdd86		RES	0,(IX+dd)
FD	CBdd86		RES	0,(IY+dd)
CB	87		RES	0,A
CB	80		RES	0,B
CB	81		RES	0,C
CB	82		RES	0,D
CB	83		RES	0,E
CB	84		RES	0,H
CB	85		RES	0,L
CB	8E		RES	1,(HL)
DD	CBdd8E		RES	1,(IX+dd)
FD	CBdd8E		RES	1,(IY+dd)
CB	8F		RES	1,A
CB	88		RES	1,B
CB	89		RES	1,C
CB	8A		RES	1,D
CB	8B		RES	1,E
CB	8C		RES	1,H
CB	8D		RES	1,L
CB	96		RES	2,(HL)
DD	CBdd96		RES	2,(IX+dd)
FD	CBdd96		RES	2,(IY+dd)
CB	97		RES	2,A
CB	90		RES	2,B
CB	91		RES	2,C
CB	92		RES	2,D
CB	93		RES	2,E
CB	94		RES	2,H
CB	95		RES	2,L
CB	9E		RES	3,(HL)
DD	CBdd9E		RES	3,(IX+dd)
FD	CBdd9E		RES	3,(IY+dd)
CB	9F		RES	3,A
CB	98		RES	3,B
CB	99		RES	3,C
CB	9A		RES	3,D

Hex		Befehl		Hex		Befehl	
CB	9B	RES	3,E	CB	BB	RES	7,E
CB	9C	RES	3,H	CB	BC	RES	7,H
CB	9D	RES	3,L	CB	BD	RES	7,L
CB	A6	RES	4,(HL)	C9		* RET	
DD	CBddA6	RES	4,(IX+dd)	D8		* RET	C
FD	CBddA6	RES	4,(IY+dd)	F8		* RET	M
CB	A7	RES	4,A	D0		* RET	NC
CB	A0	RES	4,B	C0		* RET	NZ
CB	A1	RES	4,C	F0		* RET	P
CB	A2	RES	4,D	E8		* RET	PE
CB	A3	RES	4,E	E0		* RET	PO
CB	A4	RES	4,H	C8		* RET	Z
CB	A5	RES	4,L	ED	4D	RETI	
CB	AE	RES	5,(HL)	ED	45	RETN	
DD	CBddAE	RES	5,(IX+dd)	CB	16	RL	(HL)
FD	CBddAE	RES	5,(IY+dd)	DD	CBdd16	RL	(IX+dd)
CB	AF	RES	5,A	FD	CBdd16	RL	(IY+dd)
CB	A8	RES	5,B	CB	17	RL	A
CB	A9	RES	5,C	CB	10	RL	B
CB	AA	RES	5,D	CB	11	RL	C
CB	AB	RES	5,E	CB	12	RL	D
CB	AC	RES	5,H	CB	13	RL	E
CB	AD	RES	5,L	CB	14	RL	H
CB	B6	RES	6,(HL)	CB	15	RL	L
DD	CBddB6	RES	6,(IX+dd)	17		* RLA	
FD	CBddB6	RES	6,(IY+dd)	CB	06	RLC	(HL)
CB	B7	RES	6,A	DD	CBdd06	RLC	(IX+dd)
CB	B0	RES	6,B	FD	CBdd06	RLC	(IY+dd)
CB	B1	RES	6,C	CB	07	RLC	A
CB	B2	RES	6,D	CB	00	RLC	B
CB	B3	RES	6,E	CB	01	RLC	C
CB	B4	RES	6,H	CB	02	RLC	D
CB	B5	RES	6,L	CB	03	RLC	E
CB	BE	RES	7,(HL)	CB	04	RLC	H
DD	CBddBE	RES	7,(IX+dd)	CB	05	RLC	L
FD	CBddBE	RES	7,(IY+dd)	07		* RLCA	
CB	BF	RES	7,A	ED	6F	RLD	
CB	B8	RES	7,B	CB	1E	RR	(HL)
CB	B9	RES	7,C	DD	CBdd1E	RR	(IX+dd)
CB	BA	RES	7,D	FD	CBdd1E	RR	(IY+dd)

Hex			Befehl	
CB	1F		RR	A
CB	18		RR	B
CB	19		RR	C
CB	1A		RR	D
CB	1B		RR	E
CB	1C		RR	H
CB	1D		RR	L
1F		*	RRA	
CB	0E		RRC	(HL)
DD	CBdd0E		RRC	(IX+dd)
FD	CBdd0E		RRC	(IY+dd)
CB	0F		RRC	A
CB	08		RRC	B
CB	09		RRC	C
CB	0A		RRC	D
CB	0B		RRC	E
CB	0C		RRC	H
CB	0D		RRC	L
0F		*	RRCA	
ED	67		RRD	
C7		*	RST	0
CF		*	RST	8
D7		*	RST	10H
DF		*	RST	18H
E7		*	RST	20H
EF		*	RST	28H
F7		*	RST	30H
FF		*	RST	38H
9E		*	SBC	A,(HL)
DD	9Edd		SBC	A,(IX+dd)
FD	9Edd		SBC	A,(IY+dd)
9F		*	SBC	A,A
98		*	SBC	A,B
99		*	SBC	A,C
9A		*	SBC	A,D
9B		*	SBC	A,E
9C		*	SBC	A,H
9D		*	SBC	A,L
DE	nn	*	SBC	A,nn
ED	42		SBC	HL,BC
ED	52		SBC	HL,DE
ED	62		SBC	HL,HL
ED	72		SBC	HL,SP
37		*	SCF	
CB	C6		SET	0,(HL)
DD	CBddC6		SET	0,(IX+dd)
FD	CBddC6		SET	0,(IY+dd)
CB	C7		SET	0,A
CB	C0		SET	0,B
CB	C1		SET	0,C
CB	C2		SET	0,D
CB	C3		SET	0,E
CB	C4		SET	0,H
CB	C5		SET	0,L
CB	CE		SET	1,(HL)
DD	CBddCE		SET	1,(IX+dd)
FD	CBddCE		SET	1,(IY+dd)
CB	CF		SET	1,A
CB	C8		SET	1,B
CB	C9		SET	1,C
CB	CA		SET	1,D
CB	CB		SET	1,E
CB	CC		SET	1,H
CB	CD		SET	1,L
CB	D6		SET	2,(HL)
DD	CBddD6		SET	2,(IX+dd)
FD	CBddD6		SET	2,(IY+dd)
CB	D7		SET	2,A
CB	D0		SET	2,B
CB	D1		SET	2,C
CB	D2		SET	2,D
CB	D3		SET	2,E
CB	D4		SET	2,H
CB	D5		SET	2,L
CB	DE		SET	3,(HL)
DD	CBddDE		SET	3,(IX+dd)
FD	CBddDE		SET	3,(IY+dd)
CB	DF		SET	3,A
CB	D8		SET	3,B
CB	D9		SET	3,C

Hex		Befehl		Hex		Befehl	
CB	DA	SET	3,D	CB	FA	SET	7,D
CB	DB	SET	3,E	CB	FB	SET	7,E
CB	DC	SET	3,H	CB	FC	SET	7,H
CB	DD	SET	3,L	CB	FD	SET	7,L
CB	E6	SET	4,(HL)	CB	26	SLA	(HL)
DD	CBddE6	SET	4,(IX+dd)	DD	CBdd26	SLA	(IX+dd)
FD	CBddE6	SET	4,(IY+dd)	FD	CBdd26	SLA	(IY+dd)
CB	E7	SET	4,A	CB	27	SLA	A
CB	E0	SET	4,B	CB	20	SLA	B
CB	E1	SET	4,C	CB	21	SLA	C
CB	E2	SET	4,D	CB	22	SLA	D
CB	E3	SET	4,E	CB	23	SLA	E
CB	E4	SET	4,H	CB	24	SLA	H
CB	E5	SET	4,L	CB	25	SLA	L
CB	EE	SET	5,(HL)	CB	2E	SRA	(HL)
DD	CBddEE	SET	5,(IX+dd)	DD	CBdd2E	SRA	(IX+dd)
FD	CBddEE	SET	5,(IY+dd)	FD	CBdd2E	SRA	(IY+dd)
CB	EF	SET	5,A	CB	2F	SRA	A
CB	E8	SET	5,B	CB	28	SRA	B
CB	E9	SET	5,C	CB	29	SRA	C
CB	EA	SET	5,D	CB	2A	SRA	D
CB	EB	SET	5,E	CB	2B	SRA	E
CB	EC	SET	5,H	CB	2C	SRA	H
CB	ED	SET	5,L	CB	2D	SRA	L
CB	F6	SET	6,(HL)	CB	3E	SRL	(HL)
DD	CBddF6	SET	6,(IX+dd)	DD	CBdd3E	SRL	(IX+dd)
FD	CBddF6	SET	6,(IY+dd)	FD	CBdd3E	SRL	(IY+dd)
CB	F7	SET	6,A	CB	3F	SRL	A
CB	F0	SET	6,B	CB	38	SRL	B
CB	F1	SET	6,C	CB	39	SRL	C
CB	F2	SET	6,D	CB	3A	SRL	D
CB	F3	SET	6,E	CB	3B	SRL	E
CB	F4	SET	6,H	CB	3C	SRL	H
CB	F5	SET	6,L	CB	3D	SRL	L
CB	FE	SET	7,(HL)	96		* SUB	(HL)
DD	CBddFE	SET	7,(IX+dd)	DD	96dd	SUB	(IX+dd)
FD	CBddFE	SET	7,(IY+dd)	FD	96dd	SUB	(IY+dd)
CB	FF	SET	7,A	97		* SUB	A
CB	F8	SET	7,B	90		* SUB	B
CB	F9	SET	7,C	91		* SUB	C

Hex		Befehl		Hex		Befehl	
92		* SUB	D	AF		* XOR	A
93		* SUB	E	A8		* XOR	B
94		* SUB	H	A9		* XOR	C
95		* SUB	L	AA		* XOR	D
D6	nn	* SUB	nn	AB		* XOR	E
AE		* XOR	(HL)	AC		* XOR	H
DD	AEdd	XOR	(IX+dd)	AD		* XOR	L
FD	AEdd	XOR	(IY+dd)	EE	nn	* XOR	nn

Anhang F

Der Z80-Befehlssatz *(numerisch)*

Der Z80-Befehlssatz ist in numerischer Folge mit den entsprechenden hexadezimalen Werten aufgelistet. Es gilt:

nn	8-Bit-Parameter
nnnn	16-Bit-Parameter
dd	8-Bit-Distanz mit Vorzeichen
*	Befehl gibt es auch beim 8080

Hex		Befehl		Hex		Befehl	
00		* NOP		14		* INC	D
01	nnnn	* LD	BC,nnnn	15		* DEC	D
02		* LD	(BC),A	16	nn	* LD	D,nn
03		* INC	BC	17		* RLA	
04		* INC	B	18	dd	JR	
05		* DEC	B	19		* ADD	HL,DE
06	nn	* LD	B,nn	1A		* LD	A,(DE)
07		* RLCA		1B		* DEC	DE
08		EX	AF,AF'	1C		* INC	E
09		* ADD	HL,BC	1D		* DEC	E
0A		* LD	A,(BC)	1E	nn	* LD	E,nn
0B		* DEC	BC	1F		* RRA	
0C		* INC	C	20	dd	JR	NZ,dd
0D		* DEC	C	21	nnnn	* LD	HL,nnnn
0E	nn	* LD	C,nn	22	nnnn	* LD	(nnnn),HL
0F		* RRCA		23		* INC	HL
10	dd	DJNZ	dd	24		* INC	H
11	nnnn	* LD	DE,nnnn	25		* DEC	H
12		* LD	(DE),A	26	nn	* LD	H,nn
13		* INC	DE	27		* DAA	

Hex		Befehl		Hex	Befehl	
28	dd	JR	Z,dd	50	* LD	D,B
29		* ADD	HL,HL	51	* LD	D,C
2A	nnnn	* LD	HL,(nnnn)	52	* LD	D,D
2B		* DEC	HL	53	* LD	D,E
2C		* INC	L	54	* LD	D,H
2D		* DEC	L	55	* LD	D,L
2E	nn	* LD	L,nn	56	* LD	D,(HL)
2F		* CPL		57	* LD	D,A
30	dd	JR	NC,dd	58	* LD	E,B
31	nnnn	* LD	SP,nnnn	59	* LD	E,C
32	nnnn	* LD	(nnnn),A	5A	* LD	E,D
33		* INC	SP	5B	* LD	E,E
34		* INC	(HL)	5C	* LD	E,H
35		* DEC	(HL)	5D	* LD	E,L
36	nn	* LD	(HL),nn	5E	* LD	E,(HL)
37		* SC		5F	* LD	E,A
38	dd	JR	C,dd	60	* LD	H,B
39		* ADD	HL,SP	61	* LD	H,C
3A	nnnn	* LD	A,(nnnn)	62	* LD	H,D
3B		* DEC	SP	63	* LD	H,E
3C		* INC	A	64	* LD	H,H
3D		* DEC	A	65	* LD	H,L
3E	nn	* LD	A,nn	66	* LD	H,(HL)
3F		* CCF		67	* LD	H,A
40		* LD	B,B	68	* LD	L,B
41		* LD	B,C	69	* LD	L,C
42		* LD	B,D	6A	* LD	L,D
43		* LD	B,E	6B	* LD	L,E
44		* LD	B,H	6C	* LD	L,H
45		* LD	B,L	6D	* LD	L,L
46		* LD	B,(HL)	6E	* LD	L,(HL)
47		* LD	B,A	6F	* LD	L,A
48		* LD	C,B	70	* LD	(HL),B
49		* LD	C,C	71	* LD	(HL),C
4A		* LD	C,D	72	* LD	(HL),D
4B		* LD	C,E	73	* LD	(HL),E
4C		* LD	C,H	74	* LD	(HL),H
4D		* LD	C,L	75	* LD	(IIL),L
4E		* LD	C,(HL)	76	* HALT	
4F		* LD	C,A	77	* LD	(HL),A

Hex	Befehl		Hex	Befehl	
78	* LD	A,B	A0	* AND	B
79	* LD	A,C	A1	* AND	C
7A	* LD	A,D	A2	* AND	D
7B	* LD	A,E	A3	* AND	E
7C	* LD	A,H	A4	* AND	H
7D	* LD	A,L	A5	* AND	L
7E	* LD	A,(HL)	A6	* AND	(HL)
7F	* LD	A,A	A7	* AND	A
80	* ADD	A,B	A8	* XOR	B
81	* ADD	A,C	A9	* XOR	C
82	* ADD	A,D	AA	* XOR	D
83	* ADD	A,E	AB	* XOR	E
84	* ADD	A,H	AC	* XOR	H
85	* ADD	A,L	AD	* XOR	L
86	* ADD	A,(HL)	AE	* XOR	(HL)
87	* ADD	A,A	AF	* XOR	A
88	* ADC	A,B	B0	* OR	B
89	* ADC	A,C	B1	* OR	C
8A	* ADC	A,D	B2	* OR	D
8B	* ADC	A,E	B3	* OR	E
8C	* ADC	A,H	B4	* OR	H
8D	* ADC	A,L	B5	* OR	L
8E	* ADC	A,(HL)	B6	* OR	(HL)
8F	* ADC	A,A	B7	* OR	A
90	* SUB	B	B8	* CP	B
91	* SUB	C	B9	* CP	C
92	* SUB	D	BA	* CP	D
93	* SUB	E	BB	* CP	E
94	* SUB	H	BC	* CP	H
95	* SUB	L	BD	* CP	L
96	* SUB	(HL)	BE	* CP	(HL)
97	* SUB	A	BF	* CP	A
98	* SBC	A,B	C0	* RET	NZ
99	* SBC	A,C	C1	* POP	BC
9A	* SBC	A,D	C2 nnnn	* JP	NZ,nnnn
9B	* SBC	A,E	C3 nnnn	* JP	nnnn
9C	* SBC	A,H	C4 nnnn	* CALL	NZ,nnnn
9D	* SBC	A,L	C5	* PUSH	BC
9E	* SBC	A,(HL)	C6 nn	* ADD	A,nn
9F	* SBC	A,A	C7	* RST	0

Hex		Befehl		Hex		Befehl	
C8		* RET	Z	CB	25	SLA	L
C9		* RET		CB	26	SLA	(HL)
CA	nnnn	* JP	Z,nnnn	CB	27	SLA	A
CB	00	RLC	B	CB	28	SRA	B
CB	01	RLC	C	CB	29	SRA	C
CB	02	RLC	D	CB	2A	SRA	D
CB	03	RLC	E	CB	2B	SRA	E
CB	04	RLC	H	CB	2C	SRA	H
CB	05	RLC	L	CB	2D	SRA	L
CB	06	RLC	(HL)	CB	2E	SRA	(HL)
CB	07	RLC	A	CB	2F	SRA	A
CB	08	RRC	B	CB	38	SRL	B
CB	09	RRC	C	CB	39	SRL	C
CB	0A	RRC	D	CB	3A	SRL	D
CB	0B	RRC	E	CB	3B	SRL	E
CB	0C	RRC	H	CB	3C	SRL	H
CB	0D	RRC	L	CB	3D	SRL	L
CB	0E	RRC	(HL)	CB	3E	SRL	(HL)
CB	0F	RRC	A	CB	3F	SRL	A
CB	10	RL	B	CB	40	BIT	0,B
CB	11	RL	C	CB	41	BIT	0,C
CB	12	RL	D	CB	42	BIT	0,D
CB	13	RL	E	CB	43	BIT	0,E
CB	14	RL	H	CB	44	BIT	0,H
CB	15	RL	L	CB	45	BIT	0,L
CB	16	RL	(HL)	CB	46	BIT	0,(HL)
CB	17	RL	A	CB	47	BIT	0,A
CB	18	RR	B	CB	48	BIT	1,B
CB	19	RR	C	CB	49	BIT	1,C
CB	1A	RR	D	CB	4A	BIT	1,D
CB	1B	RR	E	CB	4B	BIT	1,E
CB	1C	RR	H	CB	4C	BIT	1,H
CB	1D	RR	L	CB	4D	BIT	1,L
CB	1E	RR	(HL)	CB	4E	BIT	1,(HL)
CB	1F	RR	A	CB	4F	BIT	1,A
CB	20	SLA	B	CB	50	BIT	2,B
CB	21	SLA	C	CB	51	BIT	2,C
CB	22	SLA	D	CB	52	BIT	2,D
CB	23	SLA	E	CB	53	BIT	2,E
CB	24	SLA	H	CB	54	BIT	2,H

Hex	Befehl	Hex	Befehl
CB 55	BIT 2,L	CB 7D	BIT 7,L
CB 56	BIT 2,(HL)	CB 7E	BIT 7,(HL)
CB 57	BIT 2,A	CB 7F	BIT 7,A
CB 58	BIT 3,B	CB 80	RES 0,B
CB 59	BIT 3,C	CB 81	RES 0,C
CB 5A	BIT 3,D	CB 82	RES 0,D
CB 5B	BIT 3,E	CB 83	RES 0,E
CB 5C	BIT 3,H	CB 84	RES 0,H
CB 5D	BIT 3,L	CB 85	RES 0,L
CB 5E	BIT 3,(HL)	CB 86	RES 0,(HL)
CB 5F	BIT 3,A	CB 87	RES 0,A
CB 60	BIT 4,B	CB 88	RES 1,B
CB 61	BIT 4,C	CB 89	RES 1,C
CB 62	BIT 4,D	CB 8A	RES 1,D
CB 63	BIT 4,E	CB 8B	RES 1,E
CB 64	BIT 4,H	CB 8C	RES 1,H
CB 65	BIT 4,L	CB 8D	RES 1,L
CB 66	BIT 4,(HL)	CB 8E	RES 1,(HL)
CB 67	BIT 4,A	CB 8F	RES 1,A
CB 68	BIT 5,B	CB 90	RES 2,B
CB 69	BIT 5,C	CB 91	RES 2,C
CB 6A	BIT 5,D	CB 92	RES 2,D
CB 6B	BIT 5,E	CB 93	RES 2,E
CB 6C	BIT 5,H	CB 94	RES 2,H
CB 6D	BIT 5,L	CB 95	RES 2,L
CB 6E	BIT 5,(HL)	CB 96	RES 2,(HL)
CB 6F	BIT 5,A	CB 97	RES 2,A
CB 70	BIT 6,B	CB 98	RES 3,B
CB 71	BIT 6,C	CB 99	RES 3,C
CB 72	BIT 6,D	CB 9A	RES 3,D
CB 73	BIT 6,E	CB 9B	RES 3,E
CB 74	BIT 6,H	CB 9C	RES 3,H
CB 75	BIT 6,L	CB 9D	RES 3,L
CB 76	BIT 6,(HL)	CB 9E	RES 3,(HL)
CB 77	BIT 6,A	CB 9F	RES 3,A
CB 78	BIT 7,B	CB A0	RES 4,B
CB 79	BIT 7,C	CB A1	RES 4,C
CB 7A	BIT 7,D	CB A2	RES 4,D
CB 7B	BIT 7,E	CB A3	RES 4,E
CB 7C	BIT 7,H	CB A4	RES 4,H

Hex		Befehl		Hex		Befehl	
CB	A5	RES	4,L	CB	CD	SET	1,L
CB	A6	RES	4,(HL)	CB	CE	SET	1,(HL)
CB	A7	RES	4,A	CB	CF	SET	1,A
CB	A8	RES	5,B	CB	D0	SET	2,B
CB	A9	RES	5,C	CB	D1	SET	2,C
CB	AA	RES	5,D	CB	D2	SET	2,D
CB	AB	RES	5,E	CB	D3	SET	2,E
CB	AC	RES	5,H	CB	D4	SET	2,H
CB	AD	RES	5,L	CB	D5	SET	2,L
CB	AE	RES	5,(HL)	CB	D6	SET	2,(HL)
CB	AF	RES	5,A	CB	D7	SET	2,A
CB	B0	RES	6,B	CB	D8	SET	3,B
CB	B1	RES	6,C	CB	D9	SET	3,C
CB	B2	RES	6,D	CB	DA	SET	3,D
CB	B3	RES	6,E	CB	DB	SET	3,E
CB	B4	RES	6,H	CB	DC	SET	3,H
CB	B5	RES	6,L	CB	DD	SET	3,L
CB	B6	RES	6,(HL)	CB	DE	SET	3,(HL)
CB	B7	RES	6,A	CB	DF	SET	3,A
CB	B8	RES	7,B	CB	E0	SET	4,B
CB	B9	RES	7,C	CB	E1	SET	4,C
CB	BA	RES	7,D	CB	E2	SET	4,D
CB	BB	RES	7,E	CB	E3	SET	4,E
CB	BC	RES	7,H	CB	E4	SET	4,H
CB	BD	RES	7,L	CB	E5	SET	4,L
CB	BE	RES	7,(HL)	CB	E6	SET	4,(HL)
CB	BF	RES	7,A	CB	E7	SET	4,A
CB	C0	SET	0,B	CB	E8	SET	5,B
CB	C1	SET	0,C	CB	E9	SET	5,C
CB	C2	SET	0,D	CB	EA	SET	5,D
CB	C3	SET	0,E	CB	EB	SET	5,E
CB	C4	SET	0,H	CB	EC	SET	5,H
CB	C5	SET	0,L	CB	ED	SET	5,L
CB	C6	SET	0,(HL)	CB	EE	SET	5,(HL)
CB	C7	SET	0,A	CB	EF	SET	5,A
CB	C8	SET	1,B	CB	F0	SET	6,B
CB	C9	SET	1,C	CB	F1	SET	6,C
CB	CA	SET	1,D	CB	F2	SET	6,D
CB	CB	SET	1,E	CB	F3	SET	6,E
CB	CC	SET	1,H	CB	F4	SET	6,H

Hex			Befehl		Hex		Befehl	
CB	F5		SET	6,L	DD	46dd	LD	B,(IX+dd)
CB	F6		SET	6,(HL)	DD	4Edd	LD	C,(IX+dd)
CB	F7		SET	6,A	DD	56dd	LD	D,(IX+dd)
CB	F8		SET	7,B	DD	5Edd	LD	E,(IX+dd)
CB	F9		SET	7,C	DD	66dd	LD	H,(IX+dd)
CB	FA		SET	7,D	DD	6Edd	LD	L,(IX+dd)
CB	FB		SET	7,E	DD	70dd	LD	(IX+dd),B
CB	FC		SET	7,H	DD	71dd	LD	(IX+dd),C
CB	FD		SET	7,L	DD	72dd	LD	(IX+dd),D
CB	FE		SET	7,(HL)	DD	73dd	LD	(IX+dd),E
CB	FF		SET	7,A	DD	74dd	LD	(IX+dd),H
CC	nnnn	*	CALL	Z,nnnn	DD	75dd	LD	(IX+dd),L
CD	nnnn	*	CALL	nnnn	DD	77dd	LD	(IX+dd),A
CE	nn	*	ADC	A,nn	DD	7Edd	LD	A,(IX+dd)
CF		*	RST	8	DD	86dd	ADD	A,(IX+dd)
D0		*	RET	NC	DD	8Edd	ADC	A,(IX+dd)
D1		*	POP	DE	DD	96dd	SUB	(IX+dd)
D2	nnnn	*	JP	NC,nnnn	DD	9Edd	SBC	A,(IX+dd)
D3	nn	*	OUT	(nn),A	DD	A6dd	AND	(IX+dd)
D4	nnnn	*	CALL	NC,nnnn	DD	AEdd	XOR	(IX+dd)
D5		*	PUSH	DE	DD	B6dd	OR	(IX+dd)
D6	nn	*	SUB	nn	DD	BEdd	CP	(IX+dd)
D7		*	RST	10H	DD	CBdd06	RLC	(IX+dd)
D8		*	RET	C	DD	CBdd0E	RRC	(IX+dd)
D9			EXX		DD	CBdd16	RL	(IX+dd)
DA	nnnn	*	JP	C,nnnn	DD	CBdd1E	RR	(IX+dd)
DB	nn	*	IN	A,(nn)	DD	CBdd26	SLA	(IX+dd)
DC	nnnn	*	CALL	C,nnnn	DD	CBdd2E	SRA	(IX+dd)
DD	09		ADD	IX,BC	DD	CBdd3E	SRL	(IX+dd)
DD	19		ADD	IX,DE	DD	CBdd46	BIT	0,(IX+dd)
DD	21nnnn		LD	IX,nnnn	DD	CBdd4E	BIT	1,(IX+dd)
DD	22nnnn		LD	(nnnn),IX	DD	CBdd56	BIT	2,(IX+dd)
DD	23		INC	IX	DD	CBdd5E	BIT	3,(IX+dd)
DD	29		ADD	IX,IX	DD	CBdd66	BIT	4,(IX+dd)
DD	2Annnn		LD	IX,(nnnn)	DD	CBdd6E	BIT	5,(IX+dd)
DD	2B		DEC	IX	DD	CBdd76	BIT	6,(IX+dd)
DD	34dd		INC	(IX+dd)	DD	CBdd7E	BIT	7,(IX+dd)
DD	35dd		DEC	(IX+dd)	DD	CBdd86	RES	0,(IX+dd)
DD	36ddnn		LD	(IX+dd),nn	DD	CBdd8E	RES	1,(IX+dd)
DD	39		ADD	IX,SP	DD	CBdd96	RES	2,(IX+dd)

Hex		Befehl		Hex		Befehl	
DD	CBdd9E	RES	3,(IX+dd)	ED	47	LD	I,A
DD	CBddA6	RES	4,(IX+dd)	ED	48	IN	C,(C)
DD	CBddAE	RES	5,(IX+dd)	ED	49	OUT	(C),C
DD	CBddB6	RES	6,(IX+dd)	ED	4A	ADC	HL,BC
DD	CBddBE	RES	7,(IX+dd)	ED	4Bnnnn	LD	BC,(nnnn)
DD	CBddC6	SET	0,(IX+dd)	ED	4D	RETI	
DD	CBddCE	SET	1,(IX+dd)	ED	4F	LD	R,A
DD	CBddD6	SET	2,(IX+dd)	ED	50	IN	D,(C)
DD	CBddDE	SET	3,(IX+dd)	ED	51	OUT	(C),D
DD	CBddE6	SET	4,(IX+dd)	ED	52	SBC	HL,DE
DD	CBddEE	SET	5,(IX+dd)	ED	53nnnn	LD	(nnnn),DE
DD	CBddF6	SET	6,(IX+dd)	ED	56	IM	1
DD	CBddFE	SET	7,(IX+dd)	ED	57	LD	A,I
DD	E1	POP	IX	ED	58	IN	E,(C)
DD	E3	EX	(SP),IX	ED	59	OUT	(C),E
DD	E5	PUSH	IX	ED	5A	ADC	HL,DE
DD	E9	JP	(IX)	ED	5Bnnnn	LD	DE,(nnnn)
DD	F9	LD	SP,IX	ED	5E	IM	2
DE	nn	* SBC	A,nn	ED	5F	LD	A,R
DF		* RST	18H	ED	60	IN	H,(C)
E0		* RET	PO	ED	61	OUT	(C),H
E1		* POP	HL	ED	62	SBC	HL,HL
E2	nnnn	* JP	PO,nnnn	ED	67	RRD	
E3		* EX	(SP),HL	ED	68	IN	L,(C)
E4	nnnn	* CALL	PO,nnnn	ED	69	OUT	(C),L
E5		* PUSH	HL	ED	6A	ADC	HL,HL
E6	nn	* AND	nn	ED	6F	RLD	
E7		* RST	20H	ED	72	SBC	HL,SP
E8		* RET	PE	ED	73nnnn	LD	(nnnn),SP
E9		* JP	(HL)	ED	78	IN	A,(C)
EA	nnnn	* JP	PE,nnnn	ED	79	OUT	(C),A
EB		* EX	DE,HL	ED	7A	ADC	HL,SP
EC	nnnn	* CALL	PE,nnnn	ED	7Bnnnn	LD	SP,(nnnn)
ED	40	IN	B,(C)	ED	A0	LDI	
ED	41	OUT	(C),B	ED	A1	CPI	
ED	42	SBC	HL,BC	ED	A2	INI	
ED	43nnnn	LD	(nnnn),BC	ED	A3	OUTI	
ED	44	NEG		ED	A8	LDD	
ED	45	RETN		ED	A9	CPD	
ED	46	IM	0	ED	AA	IND	

Hex		Befehl		Hex		Befehl	
ED	AB		OUTD	FD	66dd	LD	H,(IY+dd)
ED	B0		LDIR	FD	6Edd	LD	L,(IY+dd)
ED	B1		CPIR	FD	70dd	LD	(IY+dd),B
ED	B2		INIR	FD	71dd	LD	(IY+dd),C
ED	B3		OTIR	FD	72dd	LD	(IY+dd),D
ED	B8		LDDR	FD	73dd	LD	(IY+dd),E
ED	B9		CPDR	FD	74dd	LD	(IY+dd),H
ED	BA		INDR	FD	75dd	LD	(IY+dd),L
ED	BB		OTDR	FD	77dd	LD	(IY+dd),A
EE	nn	*	XOR N	FD	7Edd	LD	A,(IY+dd)
EF		*	RST 28H	FD	86dd	ADD	A,(IY+dd)
F0		*	RET P	FD	8Edd	ADC	A,(IY+dd)
F1		*	POP AF	FD	96dd	SUB	(IY+dd)
F2	nnnn	*	JP P,nnnn	FD	9Edd	SBC	A,(IY+dd)
F3		*	DI	FD	A6dd	AND	(IY+dd)
F4	nnnn	*	CALL P,nnnn	FD	AEdd	XOR	(IY+dd)
F5		*	PUSH AF	FD	B6dd	OR	(IY+dd)
F6	nn	*	OR nn	FD	BEdd	CP	(IY+dd)
F7		*	RST 30H	FD	CBdd06	RLC	(IY+dd)
F8		*	RET M	FD	CBdd0E	RRC	(IY+dd)
F9		*	LD SP,HL	FD	CBdd16	RL	(IY+dd)
FA	nnnn	*	JP M,nnnn	FD	CBdd1E	RR	(IY+dd)
FB		*	EI	FD	CBdd26	SLA	(IY+dd)
FC	nnnn	*	CALL M,nnnn	FD	CBdd2E	SRA	(IY+dd)
FD	09		ADD IY,BC	FD	CBdd3E	SRL	(IY+dd)
FD	19		ADD IY,DE	FD	CBdd46	BIT	0,(IY+dd)
FD	21nnnn		LD IY,nnnn	FD	CBdd4E	BIT	1,(IY+dd)
FD	22nnnn		LD (nnnn),IY	FD	CBdd56	BIT	2,(IY+dd)
FD	23		INC IY	FD	CBdd5E	BIT	3,(IY+dd)
FD	29		ADD IY,IY	FD	CBdd66	BIT	4,(IY+dd)
FD	2Annnn		LD IY,(nnnn)	FD	CBdd6E	BIT	5,(IY+dd)
FD	2B		DEC IY	FD	CBdd76	BIT	6,(IY+dd)
FD	34dd		INC (IY+dd)	FD	CBdd7E	BIT	7,(IY+dd)
FD	35dd		DEC (IY+dd)	FD	CBdd86	RES	0,(IY+dd)
FD	36ddnn		LD (IY+dd),nn	FD	CBdd8E	RES	1,(IY+dd)
FD	39		ADD IY,SP	FD	CBdd96	RES	2,(IY+dd)
FD	46dd		LD B,(IY+dd)	FD	CBdd9E	RES	3,(IY+dd)
FD	4Edd		LD C,(IY+dd)	FD	CBddA6	RES	4,(IY+dd)
FD	56dd		LD D,(IY+dd)	FD	CBddAE	RES	5,(IY+dd)
FD	5Edd		LD E,(IY+dd)	FD	CBddB6	RES	6,(IY+dd)

Hex		Befehl		Hex		Befehl	
FD	CBddBE	RES	7,(IY+dd)	FD	CBddFE	SET	7,(IY+dd)
FD	CBddC6	SET	0,(IY+dd)	FD	E1	POP	IY
FD	CBddCE	SET	1,(IY+dd)	FD	E3	EX	(SP),IY
FD	CBddD6	SET	2,(IY+dd)	FD	E5	PUSH	IY
FD	CBddDE	SET	3,(IY+dd)	FD	E9	JP	(IY)
FD	CBddE6	SET	4,(IY+dd)	FD	F9	LD	SP,IY
FD	CBddEE	SET	5,(IY+dd)	FE	nn	* CP	nn
FD	CBddF6	SET	6,(IY+dd)	FF		* RST	38H

Anhang G

Einzelheiten zum 8080-Befehlssatz

Eine Übersicht über den 8080-Befehlssatz ist in diesem Anhang wiedergegeben. Die Befehle werden alphabetisch nach den offiziellen Intel-Bezeichnungen aufgelistet. Die Z80-Version des Befehls ist in spitzen Klammern angegeben.

Die Buchstaben A, B, C, D, E, H, L, und die Bezeichnung SP werden für die Standard-8080-Registernamen verwendet. Zusätzlich werden die Symbole BC, DE und HL benutzt für Registerpaare. Folgende Symbole werden für allgemeine Parameter verwendet:

r, r2	8-Bit-CPU-Register
nn	Allgemeine 8-Bit-Konstante
nnnn	16-Bit-Konstante

Die Flagbits werden mit den folgenden Symbolen bezeichnet:

C	Carry
H	Half-Carry
N	Add/subtract
P	Parity
S	Sign
Z	Zero

Bei den Zilog-Befehlen werden Speicheradressen und Ein-/Ausgabe-Portadressen in Klammern eingeschlossen.

ACI nn <ADC A,nn>

Addiere die Konstante nn und das Carry-Flag zum A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gelöscht: N

ADC M <ADC A,(HL)>

Addiere das durch das HL-Register adressierte Byte im Speicher und das Carryflag zum A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gelöscht: N

ADC r <ADC A,r>

Addiere den Wert in Register r und das Carryflag zum A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gelöscht: N

ADD M <ADD A,(HL)>

Addiere das durch das HL-Register adressierte Byte zum A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gelöscht: N

ADD r <ADD A,r>

Addiere den Wert in Register r zum A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gelöscht: N

ADI nn <ADD A,nn>

Addiere die Konstante nn zum A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gelöscht: N

ANA M <AND (HL)>

Führe ein logisches UND zwischen dem A-Register und dem durch das HL-Register adressierten Byte aus. Das Ergebnis steht dann im A-Register.

Flags verändert: P, S, Z
Flags gelöscht: C, N
Flags gesetzt: H

ANA r <AND r>

Führe ein logisches UND zwischen dem A-Register und Register r aus. Das Ergebnis steht dann im A-Register. Mit dem Befehl ANA A kann man leicht Parity, Vorzeichen und das Zeroflag testen, da dieser Befehl den Wert in A nicht verändert.

Flags verändert: P, S, Z
Flags gelöscht: C, N
Flags gesetzt: H

ANI nn <AND nn>

Führe ein logisches UND zwischen dem A-Register und der Konstanten nn aus. Das Ergebnis steht dann im A-Register. Mit diesem Befehl kann man einzelne Bits im A-Register löschen. Z. B. löscht der Befehl ANI 7FH Bit 7.

Flags verändert: P, S, Z
Flags gelöscht: C, N
Flags gesetzt: H

CALL nnnn <CALL nnnn>

Unbedingter Subroutine-Call nach Adresse nnnn. Die Adresse des folgenden Befehls wird im Stack gespeichert.

Flags verändert: keine

CC	nnnn	<CALL C,nnnn>
CM	nnnn	<CALL M,nnnn>
CNC	nnnn	<CALL NC,nnnn>
CNZ	nnnn	<CALL NZ,nnnn>
CP	nnnn	<CALL P,nnnn>
CPE	nnnn	<CALL PE,nnnn>
CPO	nnnn	<CALL PO,nnnn>
CZ	nnnn	<CALL Z,nnnn>

Bedingter Subroutine-Call nach Adresse nnnn. Die Adresse des folgenden Befehls wird im Stack gespeichert. Die Bedingungen sind:

C	Bedeutet Carry-Flag gesetzt	(Carry)
M	Bedeutet Vorzeichen-Flag gesetzt	(Minus)
NC	Bedeutet Carry-Flag gelöscht	(Not carry)
NZ	Bedeutet Null-Flag gelöscht	(Not zero)
P	Bedeutet Vorzeichen-Flag gelöscht	(Plus)
PE	Bedeutet Parity-Flag gesetzt	(Parity even)
PO	Bedeutet Parity-Flag gelöscht	(Parity odd)
Z	Bedeutet Null-Flag gesetzt	(Zero)

CMA <CPL>

Komplementiere das A-Register. Dieser Befehl führt ein Einer-Komplement im A-Register aus: jedes Bit, das 0 ist, wird 1, und jedes Bit, das 1 ist, wird 0.

Flags gesetzt: H, N

CMC <CCF>

Komplementiere das Carry-Flag. Nach dem Befehl STC ausgeführt, löscht dieser Befehlt das Carry-Flag.

Flag verändert: C
Flags gelöscht: N

CMP M <CP (HL)>

Vergleiche das durch das HL-Register adressierte Byte mit dem A-Register, das der implizite Operand ist. Das Zero-Flag wird gesetzt, wenn das A-Register gleich dem Operanden ist. Das Carry-Flag wird gesetzt, wenn das A-Register kleiner ist als der Operand.

Flags verändert: C, H, O, S, Z
Flags gesetzt: N

CMP r <CP r>

Vergleiche das Register r mit dem A-Register, das der implizite Operand ist. Das Zero-Flag wird gesetzt, wenn das A-Register gleich dem Operanden ist. Das Carry-Flag wird gesetzt, wenn das A-Register kleiner ist als der Operand.

Flags verändert: C, H, O, S, Z
Flags gesetzt: N

CPI nn <CP nn>

Vergleiche die Konstante nn mit dem A-Register, das der implizite Operand ist. Das Zero-Flag wird gesetzt, wenn das A-Register gleich dem Operanden ist. Das Carry-Flag wird gesetzt, wenn das A-Register kleiner ist als der Operand.

Flags verändert: C, H, O, S, Z
Flags gesetzt: N

DAA <DAA>

Dezimale Korrektur des A-Registers. Dieser Befehl wird nach jeder Addition mit BCD-Zahlen benutzt. Der Z80 führt diese Operation bei

Addition und Subtraktion richtig aus. Der 8080 jedoch liefert ein falsches Ergebnis bei Subtraktion.

Flags verändert: C, H, O, S, Z

```
DAD   B      <ADD   HL,BC>
DAD   D      <ADD   HL,DE>
DAD   H      <ADD   HL,HL>
DAD   SP     <ADD   HL,SP>
```

Addiere das angegebene Doppel-Register zum HL-Register. Das Ergebnis steht dann im HL-Register. Dies ist eine doppelt-genaue Addition. Das Carry-Flag wird gesetzt, wenn das Ergebnis größer als 16 Bits ist (Überlauf). Der Befehl DAD H ist eine arithmetische 16-Bit-Linksverschiebung, verdoppelt also den HL-Wert. Mit dem Befehl DAD SP kann man auch den Stack-Pointer retten:

```
LXI   H,0
DAD   SP
SHLD  OLDSTK
```

Flags verändert: C, H, O, S, Z
Flags gelöscht: N

```
DCR  M      <DEC    (HL)>
```

Vermindere das Byte, das durch das HL-Register adressiert wird, um 1.

Flags verändert: H, O, S, Z
Flags gesetzt: N
Flags nicht verändert: C

```
DCR  r      <DEC    r>
```

Vermindere Register r um 1. Achten Sie darauf, daß das Carry-Flag durch diese Operation nicht verändert wird.

Flags verändert: H, O, S, Z
Flags gesetzt: N
Flags nicht verändert: C

```
DCX  B     <DEC   BC>
DCX  D     <DEC   DE>
DCX  H     <DEC   HL>
DCX  SP    <DEC   SP>
```

Vermindere das angegebene Doppel-Register um 1. Durch diesen Befehl wird keines der Flags verändert. Für einen Null-Test laden Sie ein Byte des Doppel-Registers in das A-Register und führen ein logisches OR mit dem anderen Byte aus:

```
REPEAT:
            ...
            MOV    A,C
            ORA    B
            JNZ    REPEAT
```

Flags verändert: keine

```
DI    <DI>
```

Sperre Unterbrechungen.

```
EI    <EI>
```

Unterbrechungssperre aufheben.

```
HLT   <HALT>
```

Stoppe die CPU, bis ein Reset oder ein Interrupt auftritt.

```
IN    nn     <IN     A,(nn)>
```

Lies ein Byte in das A-Register von der Port-Adresse nn.

Flags verändert: keine

INR M <INC (HL)>

Erhöhe das Byte, das durch das HL-Register adressiert wird, um 1.

Flags verändert: H, O, S, Z
Flags gesetzt: N
Flags nicht verändert: C

INR r <INC r>

Erhöhe das 8-Bit-Register um 1. Beachten Sie, daß das Carry-Flag nicht verändert wird durch diesen Befehl.

Flags verändert: H, O, S, Z
Flags gesetzt: N
Flags nicht verändert: C

INX B <INC BC>
INX D <INC DE>
INX H <INC HL>
INX SP <INC SP>

Erhöhe das angegebene Doppel-Register um 1.

Flags verändert: keine

JMP nnnn <JP nnnn>

Unbedingter Sprung zu Adresse nnnn.

Flags verändert: keine

JC nnnn <JP C,nnnn>
JM nnnn <JP M,nnnn>
JNC nnnn <JP NC,nnnn>

```
JNZ   nnnn   <JP     NZ,nnnn>
JP    nnnn   <JP     P,nnnn>
JPE   nnnn   <JP     PE,nnnn>
JPO   nnnn   <JP     PO,nnnn>
JZ    nnnn   <JP     Z,nnnn>
```

Bedingter Sprung zu Adresse nnnn, wobei:

C	Bedeutet Carry-Flag gesetzt	(Carry)
M	Bedeutet Vorzeichen-Flag gesetzt	(Minus)
NC	Bedeutet Carry-Flag gelöscht	(Not carry)
NZ	Bedeutet Zero-Flag gelöscht	(Not zero)
P	Bedeutet Vorzeichen-Flag gelöscht	(Plus)
PE	Bedeutet Parity-Flag gesetzt	(Parity even)
PO	Bedeutet Parity-Flag gelöscht	(Parity odd)
Z	Bedeutet Zero-Flag gesetzt	(Zero)

```
LDA   nnnn   <LD     A,(nnnn)>
```

Lade das A-Register mit dem Byte, das durch die 16-Bit-Adresse nnnn adressiert wird.

```
LDAX  B      <LD     A,(BC)>
LDAX  D      <LD     A,(DE)>
```

Speichere das Byte, das durch das angegebene Doppel-Register BC oder DE adressiert wird, in das A-Register. (Siehe STAX B.)

```
LHLD  nnnn   <LD     HL,(nnnn)>
```

Lade Register L mit dem Wert, der durch den 16-Bit-Wert nnnn adressiert wird. Lade Register H aus der Adresse nnnn + 1.

```
LXI   B,nnnn <LD     BC,nnnn>
LXI   D,nnnn <LD     DE,nnnn>
LXI   H,nnnn <LD     HL,nnnn>
LXI   SP,nnnn <LD    SP,nnnn>
```

Lade das angegebene Doppel-Register mit der 16-Bit-Konstanten nnnn.

MOV M,r <LD (HL),r>

Speichere das Byte in Register r in das Byte, das durch das HL-Register adressiert wird.

MOV r,M <LD r,(HL)>

Speichere das Byte, das durch das HL-Register adressiert wird, in Register r.

MOV r,r2 <LD r,r2>

Speichere das Byte aus Register r2 in r.

MVI M,nn <LD (HL),nn>

Speichere das Byte nn in das Byte, das durch das HL-Register adressiert wird.

MVI r,nn <LD r,nn>

Lade Register r mit der 8-Bit-Konstanten nn.

NOP <NOP>

Dieser Befehl hat keine Wirkung.

Flags verändert: keine

ORA M <OR (HL)>

Führe ein logisches ODER zwischen A-Register und dem Byte, das durch

das HL-Register adressiert wird, aus. Das Ergebnis steht dann im A-Register.

Flags verändert: P, S, Z
Flags gelöscht: C, H, N

ORA r <OR r>

Führe ein logisches ODER zwischen dem A-Register und Register r aus. Das Ergebnis steht dann im A-Register. Der Befehl ORA A kann Parity, Vorzeichen und Zero-Flags testen, da dieser Befehl nicht den Wert in A ändert.

Flags verändert: P, S, Z
Flags gelöscht: C, H, N

ORI nn <OR nn>

Führe ein logisches ODER zwischen dem A-Register und dem Byte nn aus. Das Ergebnis steht dann im A-Register. Mit diesem Befehl kann man einzelne Bits des A-Registers setzen. Z. B. macht der Befehl ORI 20H das Bit 5 zu 1.

Flags verändert: P, S, Z
Flags gelöscht: C, H, N

OUT nn <OUT (nn),A>

Gibt das Byte im A-Register auf den Port bei Adresse nn aus.

Flags verändert: keine

PCHL <JP (HL)>

Lade das HL-Register in den Programmzähler, und setze das Programm bei der Adresse fort, die in HL stand.

Flags verändert: keine

```
POP  B      <POP   BC>
POP  D      <POP   DE>
POP  H      <POP   HL>
```

Lade zwei Bytes aus dem Speicher in das angegebene Doppel-Register. Das Byte, das durch den Stackpointer adressiert wird, wird in das niedere Byte (C, E, oder L) geladen, dann wird der Stackpointer erhöht. Das Byte, das durch den neuen Stackpointer adressiert wird, wird dann in das höhere Byte (B, D, oder H) geladen. Der Stackpointer wird erneut erhöht.

Flags verändert: keine

```
POP  PSW    <POP   AF>
```

Lade das Byte, das durch den Stackpointer adressiert wird, in das Flag-Register (PSW), und erhöhe den Stackpointer. Dann lade das Byte, das durch den neuen Stackpointer adressiert wird, in das A-Register und erhöhe den Stackpointer erneut.

Flags verändert: alle

```
PUSH B      <PUSH  BC>
PUSH D      <PUSH  DE>
PUSH H      <PUSH  HL>
```

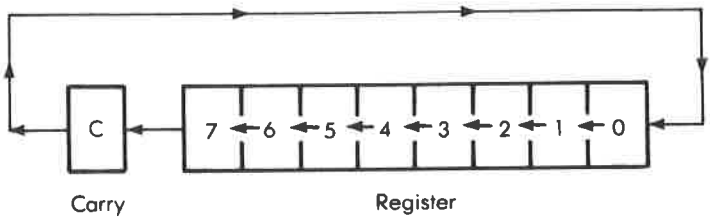
Speichere das angegebene Doppel-Register im Speicher. Der Stackpointer wird vermindert, dann wird das höhere Byte des angegebenen Registers B, D, oder H in den Speicher geschrieben, der durch den Stackpointer adressiert wird. Der Stackpointer wird erneut vermindert. Das niedere Byte C, E, oder L wird nun in das Byte, das durch den Wert des Stackpointers adressiert wird, geschrieben.

Flags verändert: keine

PUSH PSW <PUSH AF>

Speichere das A-Register und das Flag-Register im Speicher. Der Stackpointer wird vermindert, dann wird das A-Register in den Speicher geschrieben, der durch den Stackpointer adressiert wird. Der Stackpointer wird erneut vermindert. Das Flag-Register wird nun in das Byte, das durch den Wert des Stackpointers adressiert wird, geschrieben.

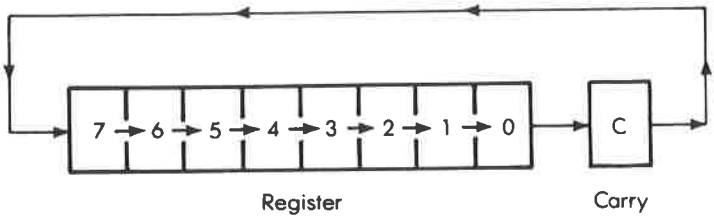
Flags verändert: keine



RAL <RLA>

Der Befehl verschiebt zyklisch Bits nach links durch das Carry um eine Position. Das Byte im A-Register wird zyklisch nach links verschoben durch das Carry. Das Carry-Flag geht in Bit 0. Bit 7 des A-Registers geht in das Carry-Flag.

Flags verändert: C
Flags gelöscht: H, N



RAR <RRA>

Der Befehl verschiebt zyklisch Bits nach rechts durch das Carry um eine

Position. Das A-Register wird zyklisch nach rechts verschoben durch das Carry. Das Carry-Flag geht in Bit 7. Bit 0 geht in das Carry-Flag.

Flag verändert: C
Flags gelöscht: H, N

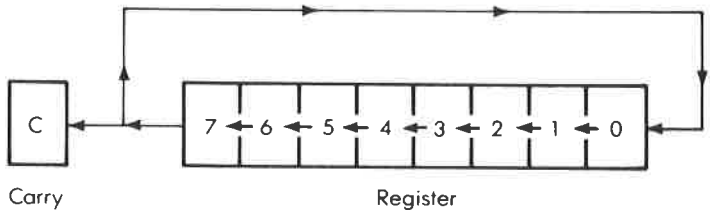
RET <RET>

Rückkehr von einer Subroutine. Der oberste Eintrag des Stack wird in den Programmzähler geladen. Der Stackpointer wird zweimal erhöht.

RC <RET C>
RM <RET M>
RNC <RET NC>
RNZ <RET NZ>
RP <RET P>
RPE <RET PE>
RPO <RET PO>
RZ <RET Z>

Bedingte Rückkehr von einer Subroutine. Der oberste Eintrag des Stack wird in den Programmzähler geladen. Der Stackpointer wird zweimal erhöht.

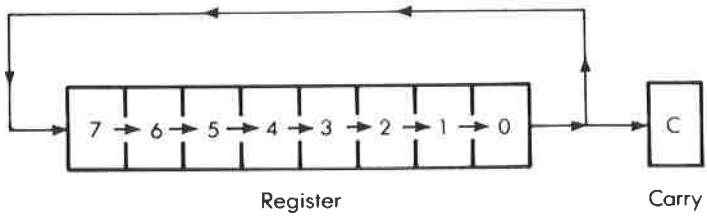
C	Bedeutet Carry-Flag gesetzt	(Carry)
M	Bedeutet Vorzeichen-Flag gesetzt	(Minus)
NC	Bedeutet Carry-Flag gelöscht	(Not carry)
NZ	Bedeutet Zero-Flag gelöscht	(Not zero)
P	Bedeutet Vorzeichen-Flag gelöscht	(Plus)
PE	Bedeutet Parity-Flag gesetzt	(Parity even)
PO	Bedeutet Parity-Flag gelöscht	(Parity odd)
Z	Bedeutet Zero-Flag gesetzt	(Zero)



RLC <RLCA>

Der Befehl verschiebt zyklisch Bits nach links um eine Position. Das Byte im A-Register wird zyklisch nach links verschoben durch das Carry. Bit 7 des A-Registers geht sowohl nach Bit 0 als auch in das Carry-Flag.

Flags verändert: C
Flags gelöscht: H, N



RRC <RRCA>

Der Befehl verschiebt zyklisch Bits nach rechts um eine Position. Das Byte im A-Register wird zyklisch nach rechts verschoben durch das Carry. Bit 0 des A-Registers geht sowohl nach Bit 7 als auch in das Carry-Flag.

Flag verändert: C
Flags gelöscht: H, N

RST	0	<RST	00H>
RST	1	<RST	08H>
RST	2	<RST	10H>
RST	3	<RST	18H>
RST	4	<RST	20H>
RST	5	<RST	28H>
RST	6	<RST	30H>
RST	7	<RST	38H>

Diese Restart-Befehle generieren Ein-Byte-Subroutine-Calls zu der Adresse im Z80-Operanden. Z. B. ruft RST 7 die Adresse 38 hex auf.

SBB M <SBC A,(HL)>

Subtrahiere das Carry-Flag und das Byte, das durch das HL-Register adressiert wird, vom A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gesetzt: N

SBB r <SBC A,r>

Subtrahiere das Carry-Flag und das angegebene CPU-Register vom A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gesetzt: N

SBI nn <SBC A,nn>

Subtrahiere das Byte nn und das Carry-Flag vom A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gesetzt: N

SHLD nnnn <LD (nnnn),HL>

Speichere Register L bei Adresse nnnn. Speichere Register H bei Adresse nnnn + 1.

SPHL <LD SP,HL>

Lade den Stackpointer vom HL-Register. Mit diesem Befehl kann man den Stackpointer restaurieren.

LHLD nnnn
SPHL

STA nnnn <LD (nnnn),A>

Speichere das A-Register in der Adresse nnnn.

STAX B <LD (BC),A>
STAX D <LD (DE),A>

Speichere das Byte im A-Register in das Byte, das durch das angegebene Register-Paar adressiert wird. (Siehe LDAX B.)

STC <SCF>

Setze das Carry-Flag. Es gibt keinen Befehl, um das Flag zu löschen. Jedoch kann das Carry-Flag gelöscht werden mit dem Befehl XRA A oder mit dem Befehlspaar SCF und CCF.

Flags gesetzt: C
Flags gelöscht: H, N

SUB M <SUB (HL)>

Subtrahiere das Byte, das durch das HL-Register adressiert wird, vom A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gesetzt: N

SUB r <SUB r>

Subtrahiere das angegebene Register vom A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gesetzt: N

SUI nn <SUB nn>

Subtrahiere das Byte nn vom A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gesetzt: N

XCHG<EX DE,HL>

Vertausche die Doppel-Register DE und HL.

Flags verändert: keine

XRA M <XOR (HL)>

Führe ein logisches Exclusiv-ODER zwischen dem A-Register und dem Byte, das durch das HL-Register adressiert wird, aus. Das Ergebnis steht dann im A-Register.

Flags verändert: P, S, Z
Flags gelöscht: C, H, N

XRA r <XOR r>

Führe ein logisches Exclusiv-ODER zwischen dem A-Register und Register r aus. Das Ergebnis steht dann im A-Register. Mit dem Befehl XRA A kann man das A-Register löschen, es werden jedoch dabei auch alle Flags gelöscht. XRA A wird auch gelegentlich verwendet, um das Carry-Flag zu löschen, da es hierfür keinen eigenen Befehl gibt.

Flags verändert: P, S, Z
Flags gelöscht: C, H, N

XRI nn <XOR nn>

Führe ein logisches Exclusiv-ODER zwischen dem A-Register und dem Byte nn aus. Das Ergebnis steht dann im A-Register.

Flags verändert: P, S, Z
Flags gelöscht: C, H, N

XTHL <EX (SP),HL>

Vertausche das Byte im Speicher, das durch den Stackpointer adressiert wird, mit Register L. Vertausche das Byte, das durch den Stackpointer + 1 adressiert wird, mit Register H.

Flags verändert: keine

Anhang H
Einzelheiten zum Z80-Befehlssatz

Eine Zusammenfassung des Z80-Befehlsatzes ist in diesem Anhang wiedergegeben. Die Befehle sind alphabetisch nach den offiziellen Zilog-Bezeichnungen geordnet. Wenn es einen entsprechenden 8080-Befehl gibt, ist der Intel-Befehl in spitzen Klammern gezeigt; Anhang G enthält Details zu diesem Befehl. Gibt es kein 8080-Äquivalent, steht „kein 8080" in spitzen Klammern. Die Z80-Befehle in numerischer Folge stehen in Anhang F. Das Z80 Äquivalent eines 8080-Befehls kann man ebenfalls Anhang G entnehmen.

Die Buchstaben A, B, C, D, E, H, I, L, IX, IY, R und SP bezeichnen die Standard-Z80-Registernamen. Zusätzlich werden die Symbole BC, DE, und HL für die Registerpaare benutzt. Die folgenden Symbole bezeichnen allgemeine Parameter:

8r, r2	8-Bit-CPU-Register
dd	8-Bit-Distanz mit Vorzeichen
nn	Allgemeine 8-Bit-Konstante
nnnn	16-Bit-Konstante

Die Flag-Bits werden mit den folgenden Symbolen bezeichnet:

C	Carry	(Übertrag)
H	Half Carry	(Halbübertrag)
N	Add/subtract	(Adittion/Subtraktion)
P/O	Parity/overflow	(Parität/Überlauf)
S	Sign	(Vorzeichen)
Z	Zero	(Null)

Speicheradressen und Ein-/Ausgabe-Portadressen werden in Klammern eingeschlossen.

ADC A,(HL) <ADC M>

```
ADC   A,(IX+dd) <kein 8080>
ADC   A,(IY+dd) <kein 8080>
```

Addiere das Byte, das durch die Summe des angegebenen Index-Registers und der Distanz bezeichnet wird, und das Carry-Flag zum A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gelöscht: N

```
ADC   A,r       <ADC     r>
```

```
ADC   A,nn      <ACI     nn>
```

```
ADC   HL,BC     <kein 8080>
ADC   HL,DE     <kein 8080>
ADC   HL,HL     <kein 8080>
ADC   HL,SP     <kein 8080>
```

Addiere das angegebene Doppel-Register und das Carry-Flag zum HL-Register. Das Ergebnis steht in HL.

Flags verändert: C, H, O, S, Z
Flags gelöscht: N

```
ADD   A,(HL)  <ADD     M>
```

```
ADD   A,(IX+dd)<kein 8080>
ADD   A,(IY+dd) <kein 8080>
```

Addiere das Byte, das durch die Summe des angegebenen Index-Registers und der Distanz bezeichnet wird, zum A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gelöscht: N

ADD A,r <ADD r>

ADD A,nn <ADI nn>

ADD HL,BC <DAD B>
ADD HL,DE <DAD D>
ADD HL,HL <DAD H>
ADD HL,SP <DAD SP>

ADD IX,BC <kein 8080>
ADD IX,DE <kein 8080>
ADD IX,IX <kein 8080>
ADD IX,SP <kein 8080>
ADD IY,BC >kein 8080>
ADD IY,DE <kein 8080>
ADD IY,IY <kein 8080>
ADD IY,SP <kein 8080>

Addiere das angegebene Doppel-Register zum angegebenen Index-Register. Das Ergebnis steht in dem Index-Register. Das HL-Register kann hier nicht angegeben werden. Es gibt auch keine äquivalente Reihe von ADC-Befehlen.

Flags verändert: C, O, S, Z
Flags gelöscht: N

AND (HL) <ANA M>

AND (IX+dd)<kein 8080>
AND (IY+dd) <kein 8080>

Führe ein logisches UND zwischen dem A-Register und dem Byte, das durch die Summe des Index-Registers und der Distanz bezeichnet wird, aus. Das Ergebnis steht im A-Register.

Flags verändert: P, S, Z
Flags gelöscht: C, N
Flags gesetzt: H

```
AND  r        <ANA    r>
```

```
AND  nn       <ANI    nn>
```

```
BIT  b,(HL)   <kein 8080>
BIT  b,(IX+dd) <kein 8080>
BIT  b,(IY+dd) <kein 8080>
```

Teste Bit b des Bytes, das durch den zweiten Operanden bezeichnet wird. b kann 0 bis 7 sein. Das Zero-Flag wird gesetzt, wenn das angegebene Bit 1 ist, sonst wird es gelöscht. Das Zero-Flag wird also das Komplement des angegebenen Bits.

Flags verändert: Z
Flags gesetzt: H
Flags gelöscht: N

```
BIT  b,r      <kein 8080>
```

Teste Bit b von Register r. b kann 0 bis 7 sein. Das Zero-Flag wird gesetzt, wenn das angegebene Bit 1 ist, sonst wird es gelöscht. Das Zero-Flag wird also das Komplement des angegebenen Bits.

Flags verändert: Z
Flags gesetzt: H
Flags gelöscht: N

```
CALL nnnn   <CALL   nnnn>
```

```
CALL C,nnnn   <CC     nnnn>
CALL M,nnnn   <CM     nnnn>
CALL NC,nnnn  <CNC    nnnn>
CALL NZ,nnnn  <CNZ    nnnn>
CALL P,nnnn   <CP     nnnn>
CALL PE,nnnn  <CPE    nnnn>
CALL PO,nnnn  <CPO    nnnn>
CALL Z,nnnn   <CZ     nnnn>
```

```
CCF   <CMC>
```

```
CP    (HL)    <CMP    M>
```

```
CP    (IX+dd)  <kein 8080>
CP    (IY+dd)  <kein 8080>
```

Vergleiche das Byte, das durch die Summe des Index-Registers und der Distanz bezeichnet wird, mit dem A-Register, das der implizite Operand ist. Das Zero-Flag wird gesetzt, wenn das A-Register gleich dem Operanden ist. Das Carry-Flag wird gesetzt, wenn das A-Register kleiner als der Operand ist.

Flags verändert: C, H, O, S, Z
Flags gesetzt: N

```
CP    r       <CMP    r>
```

```
CP    nn      <CPI    nn>
```

CPD <kein 8080>
CPDR <kein 8080>
CPI <kein 8080>
CPIR <kein 8080>

Vergleiche das Byte, das durch HL bezeichnet wird, mit dem A-Register. Vermindere HL (bei D) oder erhöhe HL (wenn I). Vermindere die Bytezahl im BC-Register. Wiederhole dies für CPDR und CPIR, bis Gleichheit auftritt oder das BC-Registerpaar Null geworden ist. Das Zero-Flag wird bei Gleichheit gesetzt. Das Parity-Flag ist an, wenn BC Null wurde.

Flags verändert: H, S
Flags gesetzt: N, Z wenn A = (HL), P wenn BC = 0

CPL <CMA>

DAA <DAA>

DEC (HL) <DCR M>

DEC (IX+dd)<kein 8080>
DEC (IY+dd) <kein 8080>

Vermindere das Byte, das durch die Summe des Index-Registers und der Distanz bezeichnet wird, um 1.

Flags verändert: H, O, S, Z
Flags gesetzt: N
Flags nicht verändert: C

DEC r <DCR r>

```
DEC  BC     <DCX   B>
DEC  DE     <DCX   D>
DEC  HL     <DCX   H>
DEC  SP     <DCX   SP>
```

```
DEC  IX     <kein 8080>
DEC  IY     <kein 8080>
```

Vermindere das Index-Register um 1.

Flags verändert: keine

```
DI    <DI>
```

```
DJNZ dd     <kein 8080>
```

Vermindere Register B und springe relativ zur Distanz dd, wenn das B-Register nicht 0 ist.

Flags verändert: keine

```
EI    <EI>
```

```
EX    (SP),HL <XTHL>
```

```
EX    (SP),IX  <kein 8080>
EX    (SP),IY  <kein 8080>
```

Vertausche die 16 Bits, die durch den Stackpointer bezeichnet werden, mit dem angegebenen Index-Register.

Flags verändert: keine

```
EX    AF,AF'  <kein 8080>
```

Vertausche das A-Register und Flag-Register mit dem zweiten Satz.

Flags verändert: alle

```
EX    DE,HL  <XCHG>
```

```
EXX   <kein 8080>
```

Vertausche BC, DE und HL mit dem zweiten Satz.

Flags verändert: keine

```
HALT <HLT>
```

```
IM    0       <kein 8080>
IM    1       <kein 8080>
IM    2       <kein 8080>
```

Setze Interruptmodus 0, 1 oder 2. Interruptmodus 0 ist automatisch selektiert bei Grundstellung des Z80. Dies ist auch der Modus der 8080-CPU. Interruptmodus 1 führt einen RST 38H-Befehl aus. Interruptmodus 2 ermöglicht viele Interrupt-Adressen.

```
IN    r,(C)    <kein 8080>
```

Lies ein Byte von der Port-Adresse in Register C in das Register r.

Flags verändert: P, S, Z
Flags gelöscht: H, N

IN A,(nn) <IN nn>

INC (HL) <INR M>

INC (IX+dd)<kein 8080>
INC (IY+dd) <kein 8080>

Erhöhe das Byte, das durch die Summe des Index-Registers und der Distanz bezeichnet wird, um 1.

Flags verändert: H, O, S, Z
Flags gesetzt: N
Flags nicht verändert: C

INC r <INR r>

INC BC <INX B>
INC DE <INX D>
INC HL <INX H>
INC SP <INX SP>

INC IX <kein 8080>
INC IY <kein 8080>

Erhöhe das angegebene Index-Register um 1.

Flags verändert: keine

IND <kein 8080>
INDR <kein 8080>
INI <kein 8080>
INIR <kein 8080>

Lies ein Byte von der Port-Adresse in Register C in das Byte, das durch HL bezeichnet wird. Vermindere Register B. Das HL-Register wird erhöht (wenn I) oder vermindert (wenn D). Bei INDR und INIR wird dies wiederholt, bis das 8-Bit-Register B 0 wird.

Flags verändert: Z (wenn B = 0)
Flags gesetzt: N

```
JP    (HL)    <PCHL>
```

```
JP    (IX)    <kein 8080>
JP    (IY)    <kein 8080>
```

Lade den Inhalt des angegebenen Index-Registers in den Programmzähler; dies ist dann die Adresse für den nächsten Befehl.

Flags verändert: keine

```
JP    nnnn    <JMP    nnnn>
```

```
JP    C,nnnn    <JC     nnnn>
JP    M,nnnn    <JM     nnnn>
JP    NC,nnnn   <JNC    nnnn>
JP    NZ,nnnn   <JNZ    nnnn>
JP    P,nnnn    <JP     nnnn>
JP    PE,nnnn   <JPE    nnnn>
JP    PO,nnnn   <JPO    nnnn>
JP    Z,nnnn    <JZ     nnnn>
```

```
JR    nn      <kein 8080>
```

Unbedingter relativer Sprung mit einer vorzeichenbehafteten Distanz nn. Der Sprung geht maximal 129 Bytes vorwärts und 126 Bytes rückwärts im Speicher.

Flags verändert: keine

```
JR    C,nn    <kein 8080>
JR    NC,nn   <kein 8080>
JR    NZ,nn   <kein 8080>
JR    Z,nn    <kein 8080>
```

Bedingter relativer Sprung zu Adresse nn, wobei:

C	Bedeutet Carry-Flag gesetzt	(Carry)
NC	Bedeutet Carry-Flag gelöscht	(Not Carry)
NZ	Bedeutet Zero-Flag gelöscht	(Not Zero)
Z	Bedeutet Zero-Flag gesetzt	(Zero)

```
LD    (BC),A  <STAX   B>
LD    (DE),A  <STAX   D>
```

```
LD    (HL),r   <MOV    M,r>
```

```
LD    (HL),nn <MVI     M,nn>
```

```
LD     (IX+dd),r   <kein 8080>
LD     (IX+dd),nn  <kein 8080>
LD     (IY+dd),r   <kein 8080>
LD     (IY+dd),nn  <kein 8080>
```

Speichere das Byte in Register r oder die Konstante nn in das Byte, das durch die Summe des Index-Registers und der Distanz bezeichnet wird.

```
LD    (nnnn),A  <STA     nnnn>
```

```
LD     (nnnn),BC  <kein 8080>
LD     (nnnn),DE  <kein 8080>
```

Speichere das niedere Byte (C oder E) des angegebenen Doppel-Registers bei Adresse nnnn. Speichere das obere Byte (B oder D) bei nnnn + 1.

```
LD    (nnnn),HL  <SHLD   nnnn>
```

```
LD    (nnnn),IX  <kein 8080>
LD    (nnnn),IY  <kein 8080>
LD    (nnnn),SP  <kein 8080>
```

Speichere das niedere Byte des angegebenen Registers IX, IY oder SP bei Adresse nnnn. Speichere das obere Byte bei nnnn + 1. Mit dem Befehl LD (nnnn),SP kann man temporär den Stack-Pointer retten. Er kann später mit LD SP,(nnnn) wieder geladen werden.

```
LD    A,(BC)  <LDAX   B>
LD    A,(DE)  <LDAX   D>
```

```
LD    A,(DE)  <LDAX   D>
```

```
LD    A,I     <kein 8080>
```

Lade das A-Register mit dem Interrupt-Vektor-Register. Das Parity-Flag zeigt den Stand des Interrupt-Enable Flip-Flops.

Flags verändert: P, S, Z
Flags gelöscht: H, N

```
LD    A,R     <kein 8080>
```

Lade das A-Register mit dem Refresh-Register. Das Parity-Flag zeigt den Stand des Interrupt-Enable Flip-Flop. Mit diesem Befehl erhält man eine einfache Zufallszahl.

Flags verändert: P, S, Z
Flags gelöscht: H, N

LD I,A <kein 8080>

Lade das A-Register in das Interrupt-Vektor-Register.

Flags verändert: keine

LD R,A <kein 8080>

Lade das A-Register in das Refresh-Register.

Flags verändert: keine

LD r,(HL) <MOV r,M>

LD r,(IX+dd) <kein 8080>
LD r,(IY+dd) <kein 8080>

Lade das Byte aus der Adresse, die durch die Summe des Index-Registers und der Distanz bezeichnet wird, in Register r.

LD r,r2 <MOV r,r2>

LD r,nn <MVI r,nn>

LD A,(nnnn) <LDA nnnn>

LD BC,(nnnn) <kein 8080>
LD DE,(nnnn) <kein 8080>

Lade das niedere Byte (C oder E) aus der Adresse, die durch den 16-Bit-Zeiger nnnn bezeichnet wird. Lade das obere Byte (B oder D) von nnnn + 1.

```
LD    HL,(nnnn)  <LHLD   nnnn>
```

```
LD    BC,nnnn    <LXI B,nnnn>
LD    DE,nnnn    <LXI D,nnnn>
LD    HL,nnnn    <LXI H,nnnn>
LD    SP,nnnn    <LXI SP,nnnn>
LD    IX,nnnn    <kein 8080>
LD    IY,nnnn    <kein 8080>
```

Lade das angegebene Doppel-Register mit der 16-Bit-Konstanten nnnn. Verwechseln Sie nicht die Befehle LD HL,(nnnn) und LD HL,nnnn.

```
LD    IX,(nnnn)  <kein 8080>
LD    IY,(nnnn)  <kein 8080>
LD    SP,(nnnn)  <kein 8080>
```

Lade das niedere Byte von IX, IY oder SP mit dem Wert an der Adresse nnnn. Lade das obere Byte mit dem Wert bei nnnn + 1. Mit dem Befehl LD SP,(nnnn) kann man einen geretteten Stack-Pointer wieder laden.

```
LD    SP,HL   <SPHL>
LD    SP,IX   <kein 8080>
LD    SP,IY   <kein 8080>
```

Lade den Stack-Pointer mit dem Wert des angegebenen 16-Bit-Registers. Mit dem Befehl SPHL kann man einen geretteten Stack-Pointer wieder laden, wenn eine 8080-CPU benutzt wird.

```
LHLD  nnnn
SPHL
```

LDD <kein 8080>
LDDR <kein 8080>
LDI <kein 8080>
LDIR <kein 8080>

Speichere das durch HL bezeichnete Byte an die durch DE bezeichnete Adresse. Vermindere den 16-Bit-Byte-Zähler in BC. Erhöhe (wenn I) oder vermindere (wenn D) HL und DE. Wiederhole dies bei LDDR und LDIR, bis das BC-Register Null geworden ist.

NEG <kein 8080>

Dieser Befehl führt ein Zweier-Komplement im A-Register aus, in dem er das A-Register von Null subtrahiert. Mit einem 8080 benutzt man hierfür die Befehle CMA und INR A.

Flags verändert: alle

NOP <NOP>

OR (HL) <ORA M>

OR (IX+dd) <kein 8080>
OR (IY+dd) <kein 8080>

Führe ein logisches ODER zwischen dem A-Register und dem Byte, das durch das angegebene Index-Register plus der Distanz bezeichnet wird, aus. Das Ergebnis steht dann im A-Register.

Flags verändert: P, S, Z
Flags gelöscht: C, H, N

OR r <ORA r>

OR nn <ORI nn>

OTDR <kein 8080>
OTIR <kein 8080>

Gib ein Byte von der durch HL bezeichneten Adresse aus. Die Port-Adresse steht in Register C. Register B wird vermindert. Das HL-Register wird erhöht (wenn I) oder vermindert (wenn D). Dies wird wiederholt, bis Register B Null geworden ist.

Flags gesetzt: N, Z

OUT (C),r <kein 8080>

Gib das Byte in Register r zu der Port-Adresse in Register C aus.

Flags verändert: keine

OUT (nn),A <OUT nn>

OUTD <kein 8080>
OUTI <kein 8080>

Gib ein Byte von der durch HL bezeichneten Adresse aus. Die Port-Adresse steht in Register C. Register B wird vermindert. Das HL-Register wird erhöht (wenn I) oder vermindert (wenn D).

Flags verändert: Z
Flags gesetzt: N

POP AF <POP PSW>

```
POP  BC     <POP   B>
POP  DE     <POP   D>
POP  HL     <POP   H>
```

```
POP  IX     <kein 8080>
POP  IY     <kein 8080>
```

Lade die beiden obersten Bytes vom Stack in das angegebene Index-Register. Erhöhe den Stack-Pointer zweimal.

Flags verändert: keine

```
PUSH AF     <PUSH  PSW>
```

```
PUSH BC     <PUSH  B>
PUSH DE     <PUSH  D>
PUSH HL     <PUSH  H>
```

```
PUSH IX     <kein 8080>
PUSH IY     <kein 8080>
```

Speichere das angegebene Index-Register in den Stack. Der Stack-Pointer wird zweimal vermindert.

Flags verändert: keine

```
RES  b,(HL)      <kein 8080>
RES  b,(IX+dd)   <kein 8080>
RES  b,(IY+dd)   <kein 8080>
```

Lösche Bit b des im zweiten Operanden angegebenen Bytes. b kann 0 bis 7 sein.

Flags verändert: kcinc

RES b,r <kein 8080>

Lösche Bit b des Registers r. b kann 0 bis 7 sein.

Flags verändert: keine

RET <RET>

RET C <RC>
RET M <RM>
RET NC <RNC>
RET NZ <RNZ>
RET P <RP>
RET PE <RPE>
RET PO <RPO>
RET Z <RZ>

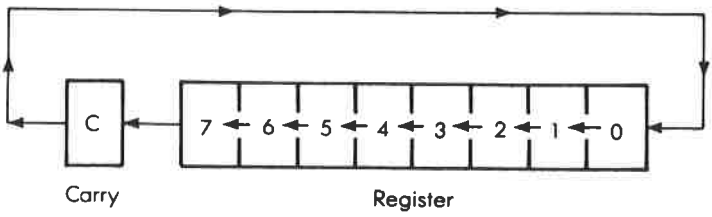
RETI <kein 8080>

Rückkehr von einem maskierbaren Interrupt.

RETN <kein 8080>

Rückkehr von einem nicht-maskierbaren Interrupt.

Die folgenden RL- und RLA-Befehle rotieren Bits nach links durch Carry.



RL (HL) <kein 8080>

Das Speicher-Byte, das durch HL bezeichnet wird, rotiert nach links durch Carry. Das Carry-Flag geht nach Bit 0. Bit 7 geht in das Carry-Flag.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

RL (IX+dd) <kein 8080>
RL (IY+dd) <kein 8080>

Das Speicher-Byte, das durch die Summe des Index-Registers und der Distanz bezeichnet wird, rotiert nach links durch Carry. Das Carry-Flag geht nach Bit 0. Bit 7 geht in das Carry-Flag.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

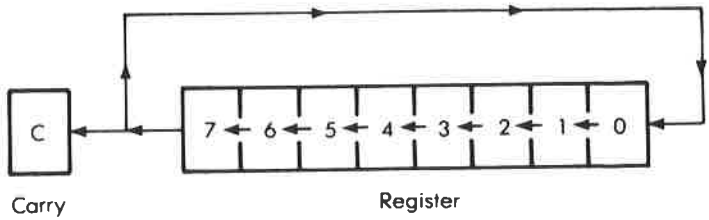
RL r <kein 8080>

Das Byte in Register r rotiert nach links durch Carry. Das Carry-Flag geht nach Bit 0. Bit 7 geht in das Carry-Flag. Der Befehl RL A macht genau dasselbe wie der Befehl RLA, dauert jedoch doppelt solange.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

RLA <RAL>

Die folgenden Befehle RLC- und RLCA- rotieren Bits nach links.



RLC (HL) <kein 8080>

Das Byte, das durch HL bezeichnet wird, rotiert zyklisch nach links. Bit 7 geht in das Bit 0 und das Carry-Flag.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

RLC (IX+dd) <kein 8080>
RLC (IY+dd) <kein 8080>

Das Byte, das durch das angegebene Index-Register plus der Distanz bezeichnet wird, rotiert zyklisch nach links. Bit 7 gcht in das Bit 0 und das Carry-Flag.

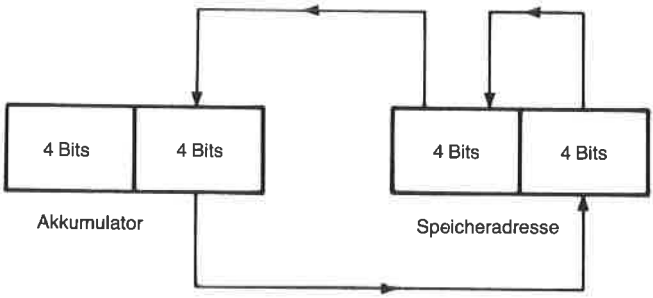
Flags verändert: C, P, S, Z
Flags gelöscht: H, N

RLC r <kein 8080>

Das Byte in Register r rotiert zyklisch nach links. Bit 7 geht in das Bit 0 und das Carry-Flag. RLC A macht genau dasselbe wie der Befehl RLCA, der Befehl RLCA ist aber doppelt so schnell.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

RLCA <RLC>

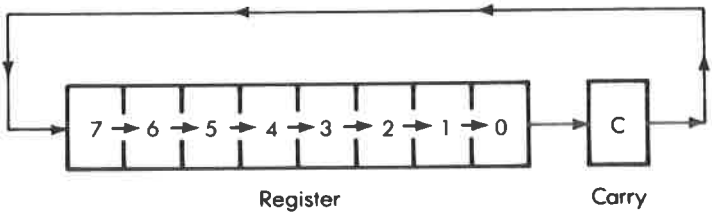


RLD <kein 8080>

Eine Vier-Bit-Rotation über 12 Bits. Die niederen vier Bits von A gehen in die niederen vier Bits des Speichers, bezeichnet durch HL. Die alten vier niederen Bits des Speichers gehen in die vier oberen Bits. Die alten oberen vier Bits gehen in die niederen vier Bits von A. Dieser Befehl wird für BCD-Operationen benutzt.

Flags verändert: P, S, Z
Flags gelöscht: H, N

Die folgenden RR- und RRA-Befehle rotieren Bits nach rechts durch Carry.



RR (HL) <kein 8080>

Das Byte, bezeichnet durch HL, rotiert nach rechts durch Carry. Carry geht zu Bit 7. Bit 0 geht in das Carry-Flag.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

RR (IX+dd) <kein 8080>
RR (IY+dd) <kein 8080>

Das Byte, das durch das angegebene Index-Register plus der Distanz bezeichnet wird, rotiert nach rechts durch Carry. Das Carry-Flag geht zu Bit 7. Bit 0 geht in das Carry-Flag.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

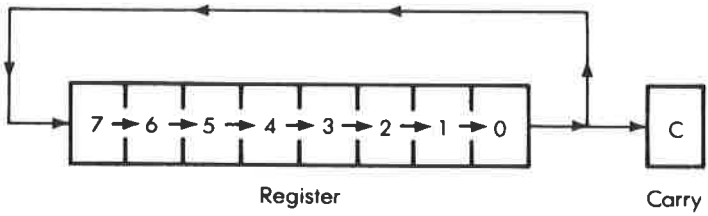
RR r <kein 8080>

Das Byte in Register r rotiert nach rechts durch Carry. Carry geht zu Bit 7. Bit 0 geht in das Carry-Flag. RR A macht genau dasselbe wie der Befehl RRA, aber der Befehl RRA ist zweimal so schnell.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

RRA <RAR>

Die folgenden RRC- und RRCA-Befehle rotieren Bits nach rechts.



RRC (HL) <kein 8080>

Das Byte, das durch HL bezeichnet wird, rotiert nach rechts. Bit 0 geht in das Carry-Flag und Bit 7.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

RRC (IX+dd) <kein 8080>
RRC (IY+dd) <kein 8080>

Das Byte, das durch das Index-Register plus der Distanz bezeichnet wird, rotiert nach rechts. Bit 0 geht in das Carry-Flag und Bit 7.

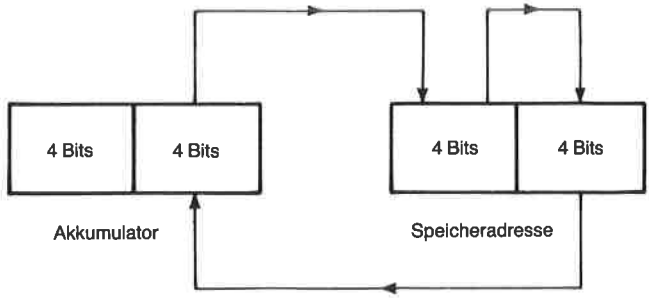
Flags verändert: C, P, S, Z
Flags gelöscht: H, N

RRC r <kein 8080>

Das Byte in Register r rotiert nach rechts. Bit 0 geht in das Carry-Flag und Bit 7. RRC A macht genau dasselbe wie der Befehl RRCA, der Befehl RRCA ist aber doppelt so schnell.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

RRCA <RRC>



RRD <kein 8080>

Eine Vier-Bit-Rotation über 12 Bits. Die niederen vier Bits von A gehen zu den oberen vier Bits des Speicher-Bytes, bezeichnet durch HL. Die alten oberen vier Bits des Speichers gehen in die unteren vier Bits. Die alten niederen vier Bits gehen in die niederen vier Bits von A. Dieser Befehl wird für BCD-Operationen benutzt.

Flags verändert: P, S, Z
Flags gelöscht: H, N

```
RST  00H   <RST   0>
RST  08H   <RST   1>
RST  10H   <RST   2>
RST  18H   <RST   3>
RST  20H   <RST   4>
RST  28H   <RST   5>
RST  30H   <RST   6>
RST  38H   <RST   7>
```

```
SBC  A,(HL)  <SBB   M>
```

```
SBC  A,(IX+dd)  <kein 8080>
SBC  A,(IY+dd)  <kein 8080>
```

Subtrahiere das Carry-Flag und das Byte, das durch die Summe des Index-Registers und der Distanz bezeichnet wird, vom A-Register. Das Ergebnis steht dann im A-Register.

Flags verändert: C, H, O, S, Z
Flags gesetzt: N

```
SBC  A,r     <SBB   r>
```

```
SBC  A,nn    <SBI   nn>
```

```
SBC  HL,BC   <kein 8080>
SBC  HL,DE   <kein 8080>
SBC  HL,HL   <kein 8080>
SBC  HL,SP   <kein 8080>
```

Subtrahiere das angegebene Doppel-Register und das Carry-Flag vom HL-Register. Das Ergebnis steht dann in HL. Meist muß man das Carry-Flag mit einem Befehl wie OR A vorher löschen.

Flags verändert: C, H, O, S, Z
Flags gesetzt: N

SCF <STC>

SET b,(HL) <kein 8080>
SET b,(IX+dd) <kein 8080>
SET b,(IY+dd) <kein 8080>

Setze Bit b des Speicher-Bytes, das durch den zweiten Operanden bezeichnet wird. b kann 0 bis 7 sein.

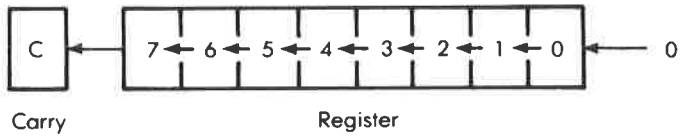
Flags verändert: keine

SET b,r <kein 8080>

Setze Bit b von Register r. b kann 0 bis 7 sein.

Flags verändert: keine

Die folgenden SLA-Befehle schieben Bits nach links.



SLA (HL) <kein 8080>

Führe einen arithmetischen Shift nach links aus mit dem Byte, das durch HL bezeichnet wird. Bit 7 geht in das Carry-Flag. Eine Null geht zu Bit 0. Hierdurch wird der alte Wert verdoppelt.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

SLA (IX+dd) <kein 8080>
SLA (IY+dd) <kein 8080>

Führe einen arithmetischen Shift nach links aus mit dem Byte, das durch das Index-Register plus der Distanz bezeichnet wird. Bit 7 geht in das Carry-Flag. Eine Null geht in das Bit 0. Hierdurch wird der alte Wert verdoppelt.

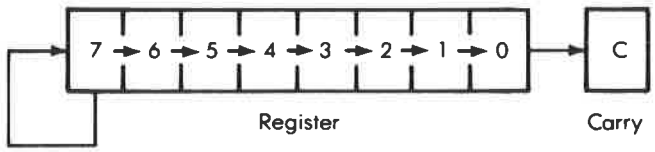
Flags verändert: C, P, S, Z
Flags gelöscht: H, N

SLA r <kein 8080>

Führe einen arithmetischen Shift nach links aus in Register r. Bit 7 geht in das Carry-Flag. Eine Null geht zu Bit 0. Dieser Befehl verdoppelt den alten Wert. SLA A macht genau dasselbe wie der Befehl ADD A,A, aber der Befehl ADD A,A ist doppelt so schnell.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

Die folgenden SRA-Befehle schieben Bits nach rechts.



SRA (HL) <kein 8080>

Führe einen arithmetischen Shift nach rechts aus mit dem Byte, das durch HL bezeichnet wird. Bit 0 geht in das Carry-Flag. Bit 7 wird nach Bit 6 kopiert.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

SRA (IX+dd) <kein 8080>
SRA (IY+dd) <kein 8080>

Führe einen arithmetischen Shift nach rechts aus mit dem Byte, das durch das Index-Register plus der Distanz bezeichnet wird. Bit 0 geht ins Carry. Bit 7 wird nach Bit 6 kopiert.

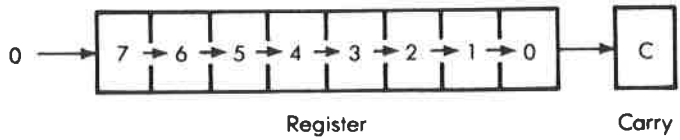
Flags verändert: C, P, S, Z
Flags gelöscht: H, N

SRA r <kein 8080>

Führe einen arithmetischen Shift nach rechts aus in Register r. Bit 0 geht ins Carry und Bit 7 wird nach Bit 6 kopiert. Dieser Befehl halbiert den Register Wert. Das Carry-Flag repräsentiert den Rest. Es ist gesetzt, wenn die alte Zahl ungerade war.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

Die folgenden SRL-Befehle schieben Bits nach rechts.



SRL (HL) <kein 8080>

Führe einen logischen Shift nach rechts aus mit dem Byte, das durch HL bezeichnet wird. Eine Null geht nach Bit 7. Bit 0 geht in das Carry-Flag.

Flags verändert: C, P, Z
Flags gelöscht: H, N, S

SRL (IX+dd) <kein 8080>
SRL (IY+dd) <kein 8080>

Führe einen logischen Shift nach rechts aus mit dem Byte, das durch das Index-Register plus der Distanz bezeichnet wird. Eine Null geht in das Bit 7. Bit 0 geht in das Carry-Flag.

Flags verändert: C, P, Z
Flags gelöscht: H, N, S

SRL r <kein 8080>

Führe einen logischen Shift nach rechts aus in Register r. Eine Null geht in das Bit 7. Bit 0 geht in das Carry-Flag.

Flags verändert: C, P, S, Z
Flags gelöscht: H, N

SUB (HL) <SUB M>

SUB (IX+dd) <kein 8080>
SUB (IY+dd) <kein 8080>

Subtrahiere das Byte, das durch das Index-Register plus der Distanz bezeichnet wird, vom Wert im A-Register. Das Ergebnis steht dann in A.

Flags verändert: C, H, O, S, Z
Flags gesetzt: N

SUB r <SUB r>

SUB nn <SUI nn>

XOR (HL) <XRA M>

XOR (IX+dd) <kein 8080>
XOR (IY+dd) <kein 8080>

Führe ein logisches Exclusiv-ODER aus zwischen dem A-Register und dem Byte, das durch die Summe des angegebenen Index-Registers und der Distanz bezeichnet wird. Das Ergebnis steht dann im A-Register.

Flags verändert: P, S, Z
Flags gelöscht: C, H, N

XOR r <XRA r>

XOR nn <XRI nn>

Weitere Einzelheiten kann man dem Buch „Z80 Assembly Language Programming Manual" der Firma Zilog Inc., 1977, entnehmen.

Anhang I
Die CP/M-BDOS-Funktionen

Die einfachen BDOS-Funktionen

Funktions-nummer (in C)	Operation	Übergebener Wert	Erhaltener Wert
1	Konsoleingabe		Zeichen in A
2	Konsolausgabe	Zeichen in E	
3	Lesereingabe		Zeichen in A
4	Stanzerausgabe	Zeichen in E	
5	Druckerausgabe	Zeichen in E	
6	Direkte Konsol-Ein-Ausgabe	FF (Eingabe) Zeichen (Ausgabe)	0 = nicht bereit oder Zeichen in A
7	Lies IOBYTE		Byte in A
8	Setze IOBYTE	in E	
9	Drucke Puffer	Adresse in DE	
10	Lies Puffer	Adresse in DE	
11	Lies Konsolstatus		Byte in A
12	Lies CP/M-Version		Byte in A und L

Die Datei-bezogenen BDOS-Funktionen

Funktions-nummer (in C)	Operation	Übergebener Wert	Erhaltener Wert
13	Laufwerk Grundstellung		
14	Laufwerkanwahl	E = Laufwerk-Nr.	
15	Eröffnc Datci	DE = FCB	Λ = Fehlercode
16	Schließe Datei	DE = FCB	A = Fehlercode

Funktions-nummer (in C)	Operation	Übergebener Wert	Erhaltener Wert
17	Suche erste Datei	DE = FCB	A = Fehlercode
18	Suche nächste		A = Fehlercode
19	Lösche Datei	DE = FCB	A = Fehlercode
20	Lies sequentiell	DE = FCB	A = Fehlercode
21	Schreibe sequentiell	DE = FCB	A = Fehlercode
22	Erstelle Datei	DE = FCB	A = Fehlercode
23	Umbenenne Datei	DE = Original-FCB	A = Fehlercode
24	Bestimme momentane Laufwerke		HL = Vektor
25	Finde Standardlaufwerk		A = Laufwerk
26	Setze DMA-Adresse	DE = Adresse	
27	Lies Belegtabelle		HL = Vektor
28	Setze Schreibschutz		
29	Finde R/O-Laufwerke		HL = Vektor
30	Setze Dateiattribute	DE = FCB	
31	Lies Disk-Parameter-Block		HL = Block
32	Lies oder setze Benutzernummer	E = FF E = neue Ben.-Nr.	A = Ben.-Nr.
33	Lies direkt	DE = FCB	A = Fehlercode
34	Schreibe direkt	DE = FCB	A = Fehlercode
35	Lies Dateigröße	DE = FCB	
36	Setze Direkt-Satznummer	DE = FCB	

Stichwortverzeichnis

Die SYBEX Bibliothek

Einführende Literatur

VORSICHT! Computer brauchen Pflege
von Rodnay Zaks – das Buch, das Ihnen die Handhabung eines Computersystems erklärt – vor allem, was Sie damit nicht machen sollten. Allgemeingültige Regeln für die pflegliche Behandlung Ihres Systems. 240 Seiten, 96 Abbildungen, Best.-Nr.: **3013** (1983)

CHIP UND SYSTEM: Einführung in die Mikroprozessoren-Technik
von Rodnay Zaks – eine sehr gut lesbare Einführung in die faszinierende Welt der Computer, vom Microprozessor bis hin zum vollständigen System. 576 Seiten, 325 Abbildungen, Best.-Nr.: **3017** (1984)

SYBEX MIKROCOMPUTER LEXIKON
– die schnelle Informationsbörse! Über 1500 Definitionen, Kurzformeln, Begriffsschema der Mikroprozessor-Technik, englisch/deutsches und französisch/deutsches Wörterbuch, Bezugsquellen. 192 Seiten, Format 12,5 x 18 cm, Best.-Nr.: **3035** (1984)

MEIN ERSTER COMPUTER
von Rodnay Zaks – Der unentbehrliche Wegweiser für jeden, der den Kauf oder den Gebrauch eines Mikrocomputers erwägt, das Standardwerk in 3., überarbeiteter Ausgabe. 304 Seiten, 150 Abbildungen, zahlreiche Illustrationen, Best.-Nr.: **3040** (1984)

COMPUTER TOTAL VERRÜCKT
von Daniel Le Noury – mit diesem Buch kommen Sie wieder zur Besinnung, nachdem Sie sich halbtot gelacht haben. Ca. 100 Cartoons rund um den Computer. 96 Seiten, Best.-Nr. **3042** (1984)

MIT DEM COMPUTER UNTERWEGS
von W. Höfs – für alle, die einen netzunabhängigen Rechner benötigen; alles über Handheld-Computer. 144 S., 27 Abb., Best.-Nr. **3067** (1984)

Sprachen

BASIC

BASIC COMPUTER SPIELE/Band 1
herausgegeben von David H. Ahl – die besten Mikrocomputerspiele aus der Zeitschrift „Creative Computing" in deutscher Fassung mit Probelauf und Programmlisting. 208 Seiten, 56 Abbildungen, Best.-Nr. **3009**

BASIC COMPUTER SPIELE/Band 2
herausgegeben von David H. Ahl – 84 weitere Mikrocomputerspiele aus „Creative Computing". Alle in Microsoft-BASIC geschrieben mit Listing und Probelauf. 224 Seiten, 61 Abbildungen, Best.-Nr.: **3010**

BASIC PROGRAMME – MATHEMATIK, STATISTIK, INFORMATIK
von Alan Miller – eine Bibliothek von Programmen zu den wichtigsten Problemlösungen mit numerischen Verfahren, alle in BASIC geschrieben, mit Musterlauf und Programmlisting. 352 Seiten, 147 Abbildungen, Best.-Nr.: **3015** (1983)

PLANEN UND ENTSCHEIDEN MIT BASIC
von X. T. Bui – eine Sammlung von interaktiven, kommerziell-orientierten BASIC-Programmen für Management- und Planungsentscheidungen. 200 Seiten, 53 Abbildungen, Best.-Nr.: **3025** (1983)

BASIC FÜR DEN KAUFMANN
von D. Hergert – das BASIC-Buch für Studenten und Praktiker im kaufmännischen Bereich. Enthält Anwendungsbeispiele für Verkaufs- und Finanzberichte, Grafiken, Abschreibungen u.v.m. 208 Seiten, 85 Abbildungen, Best.-Nr.: **3026** (1983)

MEIN ERSTES BASIC PROGRAMM
von Rodnay Zaks – das Buch für Einsteiger! Viele farbige Illustrationen und leichtverständliche Diagramme bringen Spaß am Lernen. In wenigen Stunden schreiben Sie Ihr erstes nützliches Programm. 208 Seiten, illustriert, Best.-Nr.: **3033** (1983)

Pascal

EINFÜHRUNG IN PASCAL UND UCSD/PASCAL
von Rodnay Zaks – das Buch für jeden, der die Programmiersprache PASCAL lernen möchte. Vorkenntnisse in Computerprogrammierung werden nicht vorausgesetzt. Eine schrittweise Einführung mit vielen Übungen und Beispielen. 535 Seiten, 130 Abbildungen, Best.-Nr.: **3004** (1982)

DAS PASCAL HANDBUCH
von Jacques Tiberghien – ein Wörterbuch mit jeder Pascal-Anweisung und jedem Symbol, reservierten Wort, Bezeichner und Operator, für beinahe alle bekannten Pascal-Versionen. 480 Seiten, 270 Abbildungen, Format 23 x 18 cm, Best.-Nr.: **3005** (1982)

PASCAL PROGRAMME – MATHEMATIK, STATISTIK, INFORMATIK
von Alan Miller – eine Sammlung von 60 der wichtigsten wissenschaftlichen Algorithmen samt Programmauflistung und Musterdurchlauf. Ein wichtiges Hilfsmittel für Pascal-Benutzer mit technischen Anwendungen. 398 Seiten, 120 Abbildungen, Format 23 x 18 cm, Best.-Nr.: **3007** (1982)

GRUNDKURS IN PASCAL Bd. 1
von K.-H. Rollke – der sichere Einstieg in Pascal, speziell für Schule und Fortbildung (Reihe SYBEX Informatik). 224 Seiten, mit Abb., Format 17,5x25 cm, Best.-Nr. **3046** (1984), Lehrerbegleitheft Best.-Nr. **3059**

Assembler

PROGRAMMIERUNG DES Z80
von Rodnay Zaks – ein kompletter Lehrgang in der Programmierung des Z80 Mikroprozessors und eine gründliche Einführung in die Maschinensprache. 608 Seiten, 176 Abbildungen, Format DIN A5, Best.-Nr.: **3006** (1982)

Z80 ANWENDUNGEN
von J. W. Coffron – vermittelt alle nötigen Anweisungen, um Peripherie-Bausteine mit dem Z80 zu steuern und individuelle Hardware-Lösungen zu realisieren. 296 Seiten, 204 Abbildungen, Best.-Nr.: **3037** (1984)

PROGRAMMIERUNG DES 6502 (2. überarbeitete Ausgabe)
von Rodnay Zaks – Programmierung in Maschinensprache mit dem Mikroprozessor 6502, von den Grundkonzepten bis hin zu fortgeschrittenen Informationsstrukturen. 368 Seiten, 160 Abbildungen, Format DIN A5, Best.-Nr.: **3011** (1982)

6502 ANWENDUNGEN
von Rodnay Zaks – das Eingabe-/Ausgabe-Buch für Ihren 6502-Microprozessor. Stellt die meistgenutzten Programme und die dafür notwendigen Hardware-Komponenten vor. 288 Seiten, 213 Abbildungen, Best.-Nr.: **3014** (1983)

FORTGESCHRITTENE 6502-PROGRAMMIERUNG
von Rodnay Zaks – hilft Ihnen, schwierige Probleme mit dem 6502 zu lösen, stellt Ihnen Maschinenroutinen zum Arbeiten mit einem Hobbyboard vor. 288 Seiten, 140 Abbildungen, Best.-Nr.: **3047** (1984)

PROGRAMMIERUNG DES 8086/8088
von J. W. Coffron – lehrt Sie Programmierung, Kontrolle und Anwendung dieses 16-Bit-Mikroprozessors; vermittelt Ihnen das notwendige Wissen zu optimaler Nutzung Ihrer Maschine, von der internen Architektur bis hin zu fortgeschrittenen Adressierungstechniken. 312 Seiten, 107 Abbildungen, Best.-Nr.: **3050** (1984)

C

ERFOLGREICH PROGRAMMIEREN MIT C
von J. A. Illik – ein unentbehrliches Handbuch für jeden, der mit der universellen Sprache C erfolgreich programmieren will. Aussagekräftige Beispiele, auf verschiedenen Mini- und Mikrocomputern getestet. 408 Seiten, Best.-Nr.: **3055** (1984)

Spezielle Geräte

Apple

BASIC ÜBUNGEN FÜR DEN APPLE
von J.-P. Lamoitier – das Buch für APPLE-Nutzer, die einen schnellen Zugang zur Programmierung in BASIC suchen. Abgestufte Übungen mit zunehmendem Schwierigkeitsgrad. 256 Seiten, 190 Abbildungen, Best.-Nr.: **3016** (1983)

PROGRAMME FÜR MEINEN APPLE II
von S. R. Trost – enthält eine Reihe von lauffähigen Programmen samt Listing und Beispiellauf. Hilft Ihnen, viele neue Anwendungen für Ihren APPLE II zu entdecken und erfolgreich einzusetzen. 192 Seiten, 158 Abbildungen, Best.-Nr.: **3029** (1983)

APPLE II LEICHT GEMACHT
von J. Kascmer – macht Sie schnell mit Tastatur, Bildschirm und Diskettenlaufwerken vertraut. Sie lernen, wie leicht es ist, Ihr eigenes BASIC-Programm zu schreiben. 192 Seiten, mit 43 Abbildungen, Best.-Nr.: **3031** (1984)

APPLE II BASIC HANDBUCH
von D. Hergert – ein handliches Nachschlagewerk, das neben Ihren Apple II, II+ oder IIe stehen sollte. Dank vieler Tips und Vorschläge eine wesentliche Erleichterung fürs Programmieren. 304 Seiten, 116 Abbildungen, Best.-Nr. **3036** (1984)

V24/RS-232 KOMMUNIKATION
von J. Campbell – zeigt Ihnen, wie Sie mit den Schnittstellen V24 und RS-232 notwendiges Zubehör an Ihren Rechner anschließen; mit praktischen Fallbeispielen. 224 Seiten, 97 Abb., Best.-Nr. **3075** (1984)

APPLE II/IIe ASSEMBLER KURS
Reihe MISTER MICRO – Assembler-Programmierung auf dem Apple leicht gemacht. Das Buch vermittelt alle Instruktionen für den 6502-Prozessor. Der Assembler kann jederzeit für eigene Programme eingesetzt werden. 240 Seiten, Buch und Diskette, Best.-Nr. **3408** (1984)

Commodore

COMMODORE 64 – LEICHT GEMACHT
von J. Kascmer – führt Sie schnell in die Bedienung von Tastatur, Bildschirm und Diskettenlaufwerken ein, macht Sie zum BASIC-Programmierer Ihres C64! 176 Seiten, 36 Abbildungen, Best.-Nr.: **3038** (1984)

FARBSPIELE MIT DEM COMMODORE 64
von W. Black und M. Richter – 20 herrliche Farbspiele für Ihren C64, mit Beschreibung, Programmlisten und Bildschirm-Darstellungen. Für mehr Freizeit-Spaß mit Ihrem Commodore! 176 Seiten, 58 Abbildungen, Best.-Nr.: **3044** (1984)

COMMODORE 64 BASIC HANDBUCH
von D. Hergert – zeigt Ihnen alle Anwendungsmöglichkeiten Ihres C64 und beschreibt das vollständige BASIC-Vokabular anhand von praktischen Beispielen. 208 Seiten, 92 Abbildungen, Best.-Nr.: **3048** (1984)

COMMODORE 64 PROGRAMMSAMMLUNG
von S. R. Trost – mehr als 70 getestete Anwenderprogramme, die direkt eingegeben werden können. Erläuterungen gewährleisten eine optimale Nutzung. 192 Seiten, 160 Abbildungen, Best.-Nr.: **3051** (1983)

MEIN ERSTES COMMODORE 64-PROGRAMM
von R. Zaks – sollte Ihr erstes Buch zum Commodore 64 sein. Viel Spaß am Lernen durch farbige Illustrationen und leichtverständliche Diagramme, Programmieren mit sofortigen Resultaten. 208 Seiten, illustriert, Best.-Nr. **3062** (1984)

MEIN ZWEITES COMMODORE 64 PROGRAMM
von Gary Lippman – für alle, die bereits ein Grundwissen in BASIC haben und mit ihrem C 64 den nächsten Schritt machen wollen – und das mit viel Spaß. 240 Seiten, zahlr. witzige Illustr., Best.-Nr. **3086** (erscheint 1985)

COMMODORE 64 – GRAFIK + DESIGN
von Ch. Platt – Eine Schritt-für-Schritt-Einführung in die Grafik-Programmierung Ihres C 64. Tips, die Sie in keinem Handbuch finden. 280 S., 150 Abb., teils vierfarbig. Best.-Nr. **3073** (1984)

COMMODORE 64 BASIC-KURS MIT HONEY-AID
Reihe MISTER MICRO – BASIC auf dem C64 durch Praxis lernen; mit dem integrierten Lernpaket (Buch + Software). Außer vielen Übungsprogrammen: Honey-Aid – eine universell einsetzbare BASIC-Erweiterung mit 28 zusätzlichen Befehlen. 352 Seiten, Buch und Kassette, Best.-Nr. **3400**, Buch und Diskette, Best.-Nr. **3401** (1984)

COMMODORE 64 ASSEMBLER-KURS
Reihe MISTER MICRO – zeigt in Theorie und Praxis, wie Sie den 6510-Prozessor Ihres C64 programmieren. Der mitgelieferte Assembler ist universell einsetzbar. 296 Seiten, Buch und Kassette, Best.-Nr. **3402**, Buch und Diskette, Best.-Nr. **3403** (1984)

IBM

BASIC ÜBUNGEN FÜR DEN IBM PERSONAL COMPUTER
von J.-P. Lamoitier – vermittelt Ihnen BASIC durch praktische und umfassende Übungen anhand von realistischen Programmen: Datenverarbeitung, Statistik, kommerzielle Programme, Spiele u.v.m. 256 Seiten, 192 Abbildungen, Best.-Nr.: **3023** (1983)

PROGRAMMSAMMLUNG ZUM IBM PERSONAL COMPUTER
von S. R. Trost – mehr als 65 getestete, direkt einzugebende Anwenderprogramme, die eine weite Palette von kaufmännischen, persönlichen und schulischen Anwendungen abdecken. 192 Seiten, 158 Abbildungen, Best.-Nr.: **3024** (1983)

IBM PC-DOS HANDBUCH
von R. A. King – eine umfassende Einführung in das Disketten-Betriebssystem Ihres IBM PC, seine grundsätzlichen Möglichkeiten und Funktionen sowie auch fortgeschrittene Funktionen (einschließlich der Version 2.0). 320 Seiten, 50 Abbildungen, Best.-Nr.: **3034** (1984)

ARBEITEN MIT DEM IBM PC
von J. Lasselle und C. Ramsay – zeigt Ihnen Schritt für Schritt, wie Sie den IBM PC ohne Vorkenntnisse einsetzen, die speziellen Eigenschaften dieses Computers für Druck, Grafik und Kommunikation nutzen können. 160 Seiten, 25 Abbildungen, Best.-Nr.: **3056** (1984)

IBM PC – GRAFIK FÜR DEN KAUFMANN
von N. Ford – komplette Beispielprogramme zeigen Ihnen, wie Sie Ihre eigenen Programme für kommerzielle Grafiken auf dem IBM PC erstellen. Ca. 280 Seiten, 74 Abb., Best.-Nr. **3076** (erscheint 1985)

Schneider CPC 464

SCHNEIDER CPC 464: MEIN ERSTES BASIC PROGRAMM
von Rodnay Zaks – zahlreiche farbige Illustrationen und viele Diagramme helfen Ihnen, auf spielerische Weise in BASIC zu programmieren; ohne Vorkenntnisse nutzbar. Ca. 208 Seiten, zahl. farb. Abb., Best.-Nr. **3096** (erscheint 1985)

SVI

SVI PROGRAMM-SAMMLUNG
von S. R. Trost – Knapp 70 ausgetestete Anwenderprograme, u. a. für kommerzielle Berechnungen, Dateiverwaltung und mathematische Übungen; ohne Vorkenntnisse nutzbar. 192 Seiten, 160 Abb., Best.-Nr. **3074** (1984)

Systemsoftware

CP/M-HANDBUCH
von Rodnay Zaks – das Standardwerk über CP/M, das meistgebrauchte Betriebssystem für Mikrocomputer. Für Anfänger eine verständliche Einführung, für Fortgeschrittene ein umfassendes Nachschlagewerk über die CP/M-Versionen 2.2, 3.0 und CCP/M-86 sowie MP/M., 2. überarbeitete Ausgabe. 356 Seiten, 56 Abbildungen, Best.-Nr.: **3053** (1984)

UNIX-HANDBUCH
von R. Detering – eine systematische Einführung in UNIX, das kommende Betriebssystem für 16-bit-Rechner. Lernen Sie, Ihren Prozessor optimal einzusetzen! 392 Seiten, 37 Abbildungen, Best.-Nr.: **3054** (1984)

MIKROPROZESSOR INTERFACE TECHNIKEN (3. überarbeitete Ausgabe)
von Rodnay Zaks/Austin Lesea – Hardware- und Software-Verbindungstechniken samt Digital/Analog-Wandler, Peripheriegeräte, Standard-Busse und Fehlersuchtechniken. 432 Seiten, 400 Abbildungen, Format DIN A5, Best.-Nr.: **3012** (1982)

V24/RS-232 KOMMUNIKATION
von J. Campbell – zeigt Ihnen, wie Sie mit den Schnittstellen V24 und RS-232 notwendiges Zubehör an Ihren Rechner anschließen; mit praktischen Fallbeispielen. 224 Seiten, 97 Abb., Best.-Nr. **3075** (1984)

Anwendungssoftware

EINFÜHRUNG IN DIE TEXTVERARBEITUNG
von Hal Glatzer – beschreibt, woraus eine Textverarbeitungsanlage besteht, wie man sie nutzen kann und wozu sie fähig ist. Beispiele verschiedener Anwendungen und Kriterien für den Kauf eines Systems. 248 Seiten, 67 Abbildungen, Best.-Nr. **3018** (1983)

EINFÜHRUNG IN WORDSTAR
von Arthur Naiman – eine klar gegliederte Einführung, die aufzeigt, wie das Textbearbeitungsprogramm WORDSTAR funktioniert, was man damit tun kann und wie es eingesetzt wird. 240 Seiten, 36 Abbildungen, Best.-Nr.: **3019** (1983)

ERFOLG MIT VisiCalc
von D. Hergert – umfassende Einführung in VisiCalc und seine Anwendung. Zeigt Ihnen u. a.: Aufstellung eines Verteilungsbogens, Benutzung von VisiCalc-Formeln, Verwendung der DIF-Datei-Funktion. 224 Seiten, 58 Abbildungen, Best.-Nr.: **3030** (1983)

ERFOLG MIT MULTIPLAN
von Th. Ritter – das Tabellenkalkulations-Programm Multiplan hilft Ihnen bei der Lösung kommerzieller, wissenschaftlicher und allgemeiner Probleme. Lernen Sie die Möglichkeiten kennen, Ihre Software optimal zu nutzen! 208 Seiten, 68 Abbildungen, Best.-Nr.: **3043** (1984)

ARBEITEN MIT dBase II
von A. Simpson – Grundlagen und Programmiertechniken für die Datenbank-Verwaltung mit dBASE II. Zahlreiche praktische Tips. 264 Seiten, 50 Abb., Best.-Nr. **3070** (1984)

LOTUS 1-2-3. DATENVERARBEITUNG OHNE VORKENNTNISSE
von S. Heine – für alle, die ohne DV-Kenntnisse das starke Software-Paket LOTUS 1-2-3 für berufliche oder private Anwendungen einsetzen möchten. 264 Seiten, 64 Abb., Best.-Nr. **3052** (1985)

ARBEITEN MIT LOTUS 1-2-3
von B. F. Kehlmann – die wichtigsten Anwendungsfunktionen von LOTUS 1-2-3 im Betrieb anhand praktischer Fallstudien. 232 Seiten, 115 Abb., Best.-Nr. **3078** (1985)

SYBEX

Programmieren mit CP/M

Für fortgeschrittene CP/M-Anwender und Systemprogrammierer.

„Programmieren mit CP/M" erläutert Ihnen Techniken für den einfachen Gebrauch, die Änderung und Erweiterung des Betriebssystems CP/M. Das Buch, aus der Praxis heraus geschrieben, vermittelt ein umfassendes Verständnis der CP/M-Module, insbesondere des BIOS und des BDOS. Auch Makros, wichtige Instrumente für die effektive Entwicklung von Programmen in der Assemblersprache, werden anschaulich dargestellt.

Fortgeschrittene CP/M-Anwender und Systemprogrammierer lernen, die Feinheiten von CP/M zu erforschen und die Möglichkeiten der CP/M-Kommandos zu erweitern. Zahlreiche Tabellen im Anhang helfen dabei.

Über den Autor:

Alan R. Miller ist Professor am New Mexico Institute of Technology und arbeitet als Software-Editor. Der Doktor der Ingenieurwissenschaften unterrichtet seit 1967 Ingenieure in Methoden der Programmierung. Er arbeitet mit CP/M seit der Entwicklung dieses Betriebssystems und hat mit einer Reihe von Techniken dessen Brauchbarkeit erhöht. Die Erfahrungen aus seiner Arbeit veröffentlichte Alan R. Miller bereits in mehreren SYBEX-Büchern.

ISBN 3-88745-077-9